CAR DEALERSHIP WEB APPLICATION

BY

YAP JHENG KHIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2022

CAR DEALERSHIP WEB APPLICATION

By

Yap Jheng Khin

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2022

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: CAR DEALERSHIP WEB APPLICATION

Academic Session: JAN 2022

I YAP JHENG KHIN

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

(Author's signature)

Sun Teik Heng

(Supervisor's signature)

Address:
12A JALAN 4/2
TAMAN PRIMA SAUJANA
SEKSYEN 4 43000 KAJANG
SELANGOR

Sun Teik Heng @ San Teik Heng
Supervisor's name

Date: 21-04-2022

Date: 22/4/2022

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 21-04-2022

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that YAP JHENG KHIN (ID No: 1800224) has completed this final year project entitled "*CAR DEALERSHIP WEB APPLICATION*" under the supervision of Ts Sun Teik Heng @ San Teik Heng (Supervisor) from the Department of Information Systems, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

(Yap Jheng Khin)

DECLARATION OF ORIGINALITY

I declare that this report entitled "**CAR DEALERSHIP WEB APPLICATION**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : Yap Jheng Khin

Date : 21-04-2022

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Ts Sun Teik Heng @ San Teik Heng, for being patient and understanding in guiding the final year project I and final year project II. I learnt a lot on how to write a good proposal and report under my supervisor's guidance.

ABSTRACT

In countries like the US and Europe, explainable AI and AI monitoring had become well-received as commercial companies would soon be mandated to assess AI model risk and continuously review AI systems [1]. Thus, explainable AI and AI monitoring become the integral components for any new development of commercial applications, including used car dealership web application. In this project, a web application and a web service were proposed and implemented. The used car dealership web application was implemented with ASP.NET Core. The web application was published and deployed to the Azure App Service. At the same time, the application data was seeded to the Azure SQL Database by applying the Entity Framework Core migrations. The web service performed model explaining and monitoring by querying the application data from the Azure SQL Database.

In the web service, adaptive random forest regressor and classifier, which were implemented by third-party River Python library, were used to train models that automatically detected and adapted to drift over time. Tree SHAP, which was implemented by third-party SHAP Python library, were used in model monitoring and made the models interpretable. Explainable models could enhance business value and application users' trusts in machine learning with the aid of effective visualizations like beeswarm plots. On the other hand, the data scientists could monitor and debug the models using a SHAP monitoring function, which was improved by the author, with the aid of effective visualizations like model loss bar plot. By making two initially incompatible Python libraries interoperable, the web service enhanced the functionalities of lead management application module and car inventory application module with car price analytics and lead scoring analytics, respectively.

Besides, it was observed that the initial performance of a model that was trained on one data instance at a time could never be as good as a model that was trained on the whole batch of the training set at one time. Hence, two transfer learning algorithms were proposed and implemented to provide initial performance boost to the River adaptive random forest regressor and classifier, respectively. The transfer learning algorithm pre-trained the River adaptive random forest regressor and classifier by transferring the tree structures and weights from the Scikit-learn fitted

random forest regressor and classifier, respectively. Validations were conducted to prove the correctness of the transfer learning algorithm. Experiment results proved that the offline performance of pre-trained adaptive random forest models was always as good as or better than traditional random forest models. The experiments also proved that the adaptive random forest models performed better than the traditional random forest models under the influence of drift.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADWIN	Adaptive Windowing
AI	Artificial Intelligence
API	Application programming interface
ARF	Adaptive random forest
CRUD	Create, Read, Updated, Delete
LIME	Local Interpretable Model-agnostic Explanations
ML	Machine learning
PSI	Population stability index
PWA	Progressive web application
RF	Random forest
ROC AUC	Area Under the Receiver Operating Characteristic (ROC) Curve
SDK	Software development kit
SHAP	Shapley Additive Explanations
Tree SHAP	Tree Shapley Additive Explanations
TRF	Traditional random forest
VFDT	Hoeffding tree
XAI	Explainable Artificial Intelligence

CHAPTER 1 INTRODUCTION

1.1 Problem Statement and Motivation

In 2021, according to a survey conducted by Carsome Sdn. Bhd., the intention to purchase cars among Malaysian increased by 32% during the COVID-19 post-lockdown [2]. It was because most Malaysians were unwilling to use ride-sharing services or public transports due to fear of getting COVID-19 [2]. However, the intention to sell cars among Malaysian increased by 133% at the same period. The used car market was becoming more competitive since the supply of used cars exceeds the demand for used cars. Hence, the author was motivated to develop a car dealership web application with enhanced functionalities to provide competitive advantages to its system users.

According to research conducted by Brennen in 2020, most industry people or the end-user did not understand how AI works and did not trust them [3]. This discouraged the adoption of sophisticated AI applications like self-driving vehicles and financial robo-advisors in business. In the interviews conducted by Brennen, explainable AI could help the non-tech people to see values in AI and feel comfortable with AI without understanding how AI works [3]. Hence, the author was motivated to implement explainable AI functionalities to promote application users' trust in both deployed models. For example, a used car dealer could trust a car price prediction even more if he or she understood which features that the model mainly used in predicting the car price.

Data pattern evolved from time to time. This was due to changes in business environment caused by multiple external factors such as pandemic or economic recession. Some machine learning algorithms no longer served the purpose of big data applications since they assumed that the data distribution was identical and independent [11]. As a result, deployed AI models would become obsolete unless the models retrained with the evolving data. Manual model performance monitoring and model retraining would be infeasible since the occurrence of changes in data patterns was often unpredictable. Therefore, the author was motivated to implement an automated AI performance monitoring and apply a machine learning algorithm that could automatically retrain themselves only when changes in data pattern was detected.

1.2 Objectives

The project objectives were:

1. To implement lead management that was enhanced with predictive analytics for spending less time and resources on converting leads.
2. To implement inventory management that was enhanced with predictive analytics for setting prices that were attractive and maximize profits.
3. To implement an online AI learning and monitoring system for automatically detecting and adapting to concept drift.
4. To implement explainable AI in predictive analytics for enhancing the business value of AI and promoting used car dealers' trust in AI.

1.3 Project Scope

The first deliverable of the project was a used car dealership web application was delivered. The web application consisted of three modules which were "lead management", "inventory management", and "model monitoring".

The lead management module consisted of two submodules, which were information management and analytics. The used car dealers could manage the lead information using CRUD operations. Besides, the used car dealers could request the interpretation on the lead scoring model's global prediction behaviour to gain data insights and validate the model. For example, the used car dealers would like to know which factors were correlated with higher likelihood of a successful lead conversion. When creating or updating a record, the dealers could request a local explanation on how the model used that record to predict the lead score, in order to check whether the prediction made sense.

The inventory management module consisted of two submodules, which were information management and analytics. The used car dealers could manage the car inventory information using CRUD operations. Besides, the used car dealers could request the interpretation on the car price model's general prediction behaviour to gain data insights and validate the model. For example, the used car dealers would like to know which factors were correlated with higher car price. When creating or updating a record, the dealers could request a local explanation on how the model used that record to predict the price, in order to check whether the prediction made sense.

The model monitoring module consisted of two submodules, which were model performance evaluation, and model monitoring and debugging. The first submodule provided visualization on the running metrics and the model loss using standards metrics and SHAP loss values, respectively. The data scientists could know the model's overall performance, and the positive and negative contribution of each feature to the model error. The second submodule displayed the alerts of potential drift or data error found on the application data under two different conditions. The first condition was when the application records had no truth, and the second condition was records containing the truth. The data scientists could click on each alert to visualize the potential drift or data error in the form of table and graph.

The second deliverable of the project was the deployment of a web service. The web service provided explainable AI and model monitoring functionalities backed by two optimal machine learning models that supported incremental learning. These models included an adaptive random forest regressor for car price prediction and an adaptive random forest classifier for lead scoring classification. The web application would then call the web service to perform various task like performing predictions and monitoring model performance.

1.4 Contributions

This project could be served as the stepping stone for the deployment of sophisticated machine learning algorithms or even deep learning architecture into the commercial application. More powerful models could mean providing higher potential business values, and the business values were effectively communicated to the business stakeholders using a suitable explainable AI approach like SHAP. According to Brennen's interviews, the interviewees demanded explainable AI to be able to debug black-box models, detect models bias, ensure model fairness and finally promoting AI transparency and AI trust [3]. The source code of this report would be published to a GitHub repository under MIT license after some time. This act was to provoke new ideas by pushing the innovations of explainable AI into commercial fields such as customer clustering analysis, churn prediction, credit risk scoring, fraud detection, inventory optimisation, sales forecasting, sentimental analysis, automated investing, and more.

Besides, this project also raised awareness of the importance of explainable AI. Over the recent years, giant tech companies like Google and Microsoft had been actively pushed forward the responsible AI principles [4], [22]. Not only that, according to Burt, government bodies from US and Europe had started or proposed to regulate artificial intelligence in commercial companies [1]. Some of the regulatory requirements included conducting independent AI risk assessments (AI auditing) and continuous review of AI systems [1]. These regulatory requirements justified the implementation of AI monitoring tools in this project. In short, though governments around the world were still in the infancy stage of regulating AI, companies and higher education should educate new generations on concepts and implementations of responsible AI to prepare for any future regulatory compliances.

1.5 Report Organization

Below showed the organization of reports by algorithm.

Algorithm	Sections
Adaptive random forest algorithm	Section 2.3, Appendix A, B, and C: Algorithm review Section 5.2: Performance evaluation
Transfer learning algorithm	Section 3.5: Algorithm design and design considerations Section 5.1: Validation
Tree SHAP algorithm	Section 2.4.1, 2.4.2: Algorithm review Section 2.4.3, 2.5.3: Review on the usage of algorithm Section 3.7: Implementation and usage of algorithm Section 5.4: Validation
Model tree weight extraction	Section 3.6: Algorithm design and design considerations Section 5.3: Validation
Drift monitoring	Section 2.5: Algorithm review Section 3.8: Algorithm design and implementations Section 5.5: Validation

Table 1.5.1: Report organization grouped by algorithm

Below showed the organization of reports by application type.

Application type	Sections
Database	Section 3.4: ERD Diagram Section 4.1: Dataset information Section 4.3.1, 4.3.2: Setup Section 4.3.3: Database procedure
Web application	Section 2.1: Review on existing car dealership systems Section 3.1, 3.2, 3.3: System design Section: Setup Section 4.5.1: Setup Section 4.5.2: Description of artifacts
Web service	Section 2.2: Review on existing AI cloud services Section 3.1, 3.2, 3.3: System design Section 4.4.1: Setup Section 4.4.2: Description of artifacts
Jupyter Notebook	Section 4.2.1, 4.2.2: Setup Section 4.2.3, 4.2.4: Description of artifacts

Table 1.5.2: Report organization grouped by application types

CHAPTER 2 LITERATURE REVIEW

First, one of the car dealership management systems was reviewed to understand the common functionalities found in these systems. Second, the third-party AI cloud services were reviewed to understand the difference between integrating the explainable AI and AI monitoring functionalities into commercial applications and custom development of these functionalities. Third, the global explanations methods were reviewed to justify the use of Tree SHAP method. Fourth, the drift detection methods were reviewed to inform the latest implementation of AI monitoring and drift detection. Currently, there was no paper, documentation, nor tutorial that suggested the uses of adaptive random forest with tree SHAP. Therefore, both the Tree SHAP algorithm and adaptive random forest algorithm were extensively validated to ensure that both algorithms could work together.

2.1 Review of Existing Car Dealership Management System

The author reviewed one of the most widely used dealership management systems which was DealerCenter. DealerCenter was a cloud-based dealership management system that offered full-fledged functionalities ranging from inventory management, customer relationship management, to digital marketing campaign management.

For the functionalities for managing customer relationships, DealerCenter could synchronize leads' information from multiple car advertisement posting websites such as Facebook Marketplace, Autotrader, CarGuru and more into a single platform. The dealer also could directly message customers on Facebook through the Facebook API within the system itself. Secondly, the system also allowed the sales representatives to automate workflows such as emailing and text messages (SMS), which enhanced the productivity of the sales representatives. For instance, the task automation provided custom templates for sending a batch of text messages (SMS) and emails to a pre-defined customer segment at the same time. Thirdly, the salesperson representative could also manage the prospect's profile in and out of the showroom using this system. The sales representative could check in the prospect when he/she visits the showroom and check out when he/she leaves the showroom. After meeting with a prospect, the sales representative manually inputted the interested vehicle by that prospect and set up reminders for any future appointment. Thus, the prospect's information was stored in the system and synchronized across all

sales representative's devices. Fourthly, the system automatically assigned leads to the sales representatives, and then the sales manager could track all sales activities and key performance metrics of each salesperson.

For the functionalities for managing inventory, DealerCenter had access to over thirty marketing sites like Facebook Marketplace, CarGurus, eBay, TrueCar, Craigslist, CarZing, and more. These integrations allowed sales representatives to directly schedule the posting of inventory advertisements to the respective marketing sites. Secondly, in order to make the car image stand out from the rest of the direct competitors, the system prepared customizable overlay templates to watermark car images by adding a custom dealership logo, address, phone number and logo. Thirdly, the system provided service pricing guide features. Specifically, the system fetched valuable vehicle data like fair purchase price, trade-in values, private party values, expert's reviews, and customers' reviews from automobile research websites like Kelley Blue Book through web service. Just as new car dealers and customers referred to the service price guide provided by the car automaker to determine the price of a new car; the same rule could be applied in the used car market as well. Nevertheless, for managing digital marketing campaigns, the same web services used for guiding service pricing also allowed the dealer to post advertisements to their websites in case the visitors would like to book a test drive for that car model. Secondly, the system could automate report generation by dynamically generating the buyer guide and car history report. This could be achieved by auto-populating the specified vehicle details. Thirdly, the sales manager could view real-time analytics about the marketing campaigns.

Indeed, DealerCenter had a lot of features to offer, however, the operation cost and maintenance cost were hefty due to the overuse of APIs within the system. For example, the dealer needed to pay $5 for each API transaction whenever they would like to advertise their inventory directly through third party marketing sites such as Facebook Marketplace. This was unnecessary, as the users could still manually post the advertisement via the respective marketing sites. In other words, the system provided greater convenience in exchange for higher fees. Furthermore, the developers must keep track of all updates of the integrated APIs to ensure that the system always remained operational, thus increasing the maintenance effort. Careful consideration of cost and convenience was needed when the author was trying to

integrate any new API into the system. Thus, to ensure the reusability and maintainability of the proposed system in the future, the author did not integrate any commercial API into the system.

2.2 Review of Existing AI Cloud Services

There were few big cloud service players like Microsoft, IBM and H2O that provided AI monitoring features and AI explainability features besides deployment options. Microsoft had introduced Azure Machine Learning; H2O had introduced H2O Driverless AI; IBM had introduced IBM Watson Studio. To enhance readability, the advantages and disadvantages of the system were summarised in point form as shown below.

Advantage

1. These systems could speed up AI development by automating some manual parts of AI development like data cleaning, feature selection, feature engineering, cross-validated hyperparameter tunning and many more. First, the data scientists must define an AI experiment by uploading a dataset, set a target variable, and define main metrics for model evaluation. Then, the data scientists could run the AI experiment by setting settings such as time limit and the cloud service provider would find the best model using the applied settings through trial-and-error methods like randomised search or Bayesian optimisation.
2. These systems could detect concept drift for the models in production. These systems used various techniques like monitoring performance metrics and detecting distribution changes in variables.
3. These systems could allow users to choose suitable explainable AI methods like Shapley values to provide both local and global explanations. This improved the users' trust and understanding of the deployed model and improve the quality of the decision making.
4. These systems allowed API access to the AI experiment and AI monitoring through SDKs in programming languages like Python. This facilitated system integration by incorporating these functionalities into another system.

Disadvantage

1. Users did not have the full flexibility and customisation over the ML models offered by the cloud service provider. For example, users might not be able to

configure specific things like configuring vendor-suppled Tree SHAP to calculate SHAP loss values for monitoring AI performance.

2. The third-party libraries that could be used in AI experiments and AI monitoring were limited and highly dependent on the vendor's support. The SDKs developed by these big companies were basically closed-sourced
3. Vendor-supplied packages might still be dependent on some old dependencies, limiting the benefits gained from the new functionalities, bug fixes, and security patches. For example, for the initial prototype, the author tried to develop a Flask web service that performed a prediction using a model trained by Azure Machine Learning SDK. The development of the web service was difficult and filled with old bugs since the Flask framework was forced to downgrade to ensure compatibility with the SDK.
4. Based on the disadvantages above, the software developers were required to conduct extensive software requirements validation on these third-party services. Lack of customizability on these services stiffened the system integration and potentially inflated the future costs of system evolution. For example, the evolution cost would be high if the business decided to use latest cutting-edge deep learning frameworks and AI explainability approaches but the integrated third-party service did not support them yet.

In short, the initial prototype developed by the author in this project had provided very good insights in accessing the overall project risks. In the end, the authors avoided AI monitoring and explainable AI SDK provided by Microsoft, IBM, H2O, and others. Instead, open-source packages like River and SHAP were used so that source code could be directly modified to ensure compatibility between different packages while ensuring these packages were up to date.

2.3 Adaptive Machine Learning Algorithm

In this project, two analytics functionalities were implemented which were car price prediction and lead scoring classification. The reason that ensemble tree-based algorithms like random forests was chosen instead of the neural network was due to the limited amount of data available (less than 10,000 samples per dataset) for this project. Regardless, ensemble tree-based algorithms still proved to consistently outperform standard neural networks on structured datasets [10].

2.3.1 Adaptive random forests (ARF) algorithm

ARF algorithm was innovated by Gomes and other researchers and this algorithm was directly adapted from the classical Random Forest with ground-breaking improvements [5]. There were few notable characteristics that made the ARF algorithm suitable for this project. The first characteristic was that the ARF model could be trained incrementally with as low as one single data instance while predicting the input data at the same time. The second characteristic was that each base leaner in the ARF model had a drift warning detector and drift detector. There were two possible scenarios. Upon the first sign of data drift, a warning was issued, and a background tree was created and trained with the new data. As the drift continued, the background tree would eventually replace the existing corresponding tree to ensure the relevancy of the model in handling volatile data in the real world. The formula of the ARF algorithm was discussed in detail in APPENDIX A.

2.3.2 Hoeffding tree

Instead of using a decision tree, the base leaner of the ARF was the Hoeffding tree. Note that the inner working of the Hoeffding tree was largely similar to the decision tree, which was to split the current subset of data by choosing the feature with the lowest variance, lowest entropy, highest information gains or lowest Gini impurity at each level. Hoeffding tree replaced the classical machine learning algorithm for one major reason. The major reason was that the Hoeffding tree was designed to be compatible with the ARF algorithm. The Hoeffding tree was compatible because of two main reasons.

The first reason was that the Hoeffding tree could be incrementally trained with data of any size. The second reason was that the Hoeffding tree didn’t keep the original train data after finish training for every incoming data stream. Instead, Hoeffding tree kept important statistics of incoming data which provide minimal but sufficient information to perform split attempt on the leaf nodes [6]. As a result, this saved a lot of memory space and sped up the training process. To perform the split attempt, the Hoeffding tree had a hyperparameter called grace period to control if a split should continue based on how much data had been “kept”. The detailed formula of the Hoeffding tree was discussed in APPENDIX B.

2.4 Explainable AI

First, the right model must be used for the right kind of explainable AI task. If the objective of the task was to "educate the user", then random forest or its equivalent was used; if the objective of the task was to "help them to take an appropriate action", then causality machine learning model like structure causal models (SCM) was used [7]. Random forest could provide the features importance' information for educating the business stakeholders about the most important independent variables or factors in determining the output [7]; structural causal models (SCM) could provide actionable insights on how to improve the output [7].

In this project, an Adaptive random forest algorithm would be both used in car price prediction and lead scoring classification. Thus, the selected interpretability technique must be extensively validated to ensure that the adaptive machine learning algorithm and interpretability technique were both compatible with each other without losing any benefit from any side. Tree SHAP was selected for this project and the validation was shown in the next section.

2.4.1 Shapley values

In order to validate Tree SHAP, Shapley values must be first understood. Shapley value was the local explainability method to quantify how much each feature contributed to the difference between an individual prediction with the average predictions [8]. Shapley values were based on game theory [8]. Metaphorically, each player (a.k.a feature) would like to get a fair share of the gains (a.k.a difference between a prediction with the average prediction) based on how hard they work (a.k.a how big was the impact that feature had on the difference). In other words, important features had higher Shapley values due to a larger influence on the predicted output. One problem was that the interaction effect between two or more features caused the order to become a disruption factor in the fair distribution of the contributions among features [9]. To remove the interaction effect, the computation of Shapley values for each feature was computed 2^k times, where k was the number of features.

To demonstrate how to calculate Shapley value, take the car price prediction as an example, say the average price prediction was RM30,000, the current prediction to be explained was RM33,000, and the predictive features were manufacturing year (x_1), car brand (x_2), and transmission (x_3). To calculate the Shapley value of x_1, the

formula was given as follows, where n was the number of features, $|S|$ was the number of elements in set S, V was the payout function given a coalition of players [8].

$$\phi_{x_1} = \frac{1}{n!} \times \sum_{S \subseteq N \setminus \{x_1\}} |S|!\,(n-1-|S|)!\,[V(S \cup \{x_1\}) - V(S)]$$

S had four possible values since it meant the coalitions that did not include player x_1, which was $\{\}, \{x_2\}, \{x_3\}$ and $\{x_2, x_3\}$. Then, for each summation, the formula was substituted as shown below, before all these values were summed up and divided by $\frac{1}{3!}$.

Variables	Summation
$S = \{\}, \lvert S\rvert = 0$	$0!\,(3-1-0)!\,[V(\{x_1\}) - V(\{\})]$
$S = \{x_2\}, \lvert S\rvert = 1$	$1!\,(3-1-1)!\,[V(\{x_1, x_2\}) - V(\{x_2\})]$
$S = \{x_3\}, \lvert S\rvert = 1$	$1!\,(3-1-1)!\,[V(\{x_1, x_3\}) - V(\{x_3\})]$
$S = \{x_2, x_3\}, \lvert S\rvert = 2$	$1!\,(3-1-1)!\,[V(\{x_1, x_2, x_3\}) - V(\{x_2, x_3\})]$

Table 2.4.1.1: Summations for calculating Shapley value

In general, the Shapley value could produce accurate prediction since payout was fairly distributed by satisfying the properties namely Efficiency, Symmetry, Dummy and Additivity [8]. Using the example above, efficiency ensured that the difference between the individual predicted price RM33,000 and the average predicted price RM30,000 (RM3,000) was equivalent to $\phi_{x_1} + \phi_{x_2} + \phi_{x_3}$; symmetry ensured that $\phi_{x_1} = \phi_{x_2}$ if and only if both x_1 and x_2 had the same contribution for all possible coalitions; dummy ensured that $\phi_{x_1} = 0$ if and only if x_1 did not have contribution at all. Besides, Shapley values could be used in ensemble models since the additivity property ensured that the average of Shapley value for each base learner could be computed to explain the local prediction.

Though Shapley value was backed by solid theory, the author would need to consider other interpretability approaches due to overwhelming high computing time and the necessity to retain old data. First, the time complexity of the Shapley value was exponential. Say, the Shapley value was applied in tree-based models, the time complexity for computing Shapley value for each tree model would be $O(L2^M)$,

where L was the maximum number of leaves in the tree and M was the number of input features. Second, Shapley values were required to access the old data. This was because the values of the features that were not included in the coalition S were replaced by random values drawn from old data [8]. For example, the values for x_1 and x_3 in the coalition $\{x_2\}$ were randomly sampled from the old data. As mentioned earlier, the online AI learning system was expected to receive an infinite amount of data streams in production. Retaining data might not be a convenient option since an extra mechanism needed to be implemented to balance between storage cost and data availability.

2.4.2 Tree SHAP

Fortunately, in 2018, Tree SHAP came into existence to solve the two aforementioned limitations. Tree SHAP was the SHAP version specifically designed for tree-based models, including random forest [10]. The prerequisite of SHAP was Shapley value and thus the properties Efficiency, Symmetry, Dummy and Additivity were not violated.

Unlike Shapley value, Tree SHAP allowed faster computation time of local explanations by reducing the time complexity from exponential to polynomial time. The reduction in time complexity was because the Shapley values were computed using the internal structure of tree-based models [10]. The reliance on the tree structure justified the reduction of exponential time complexity $O(TL2^M)$ to polynomial time complexity of the algorithm, $O(TLD^2)$, where T was the number of trees, L was the number of leaves, D was the depth of the tree, and M was the number of input features [10].

Similar to the concept of Shapley values, SHAP values were computed for each feature by calculating the approximate conditional expectation function $f_x(S) = E[f(X)|do(X_s = x_s)]$ by using the paths in the tree instead of training data [10]. Thus, it was compatible with the adaptive random forest algorithm since the algorithm did not save the data after training, keeping the storage requirement to the bare minimum.

The intuition of the tree SHAP was only discussed since the exact pseudocode of tree SHAP was way too technical and complicated to be discussed in this project. Intuitively, the 2^M was reduce to D^2 since the calculation of SHAP values for all 2^M coalitions were simultaneous [10]. Since a decision tree reused features when splitting

nodes, a single coalition could reach multiple leaf nodes. According to Lundberg and other researchers [10], the weights of these leaf nodes for the SHAP values of each feature were computed by descending from the root node to the leaf node one time before reversing the traversal for one time, which explained the D^2 in the $O(TLD^2)$. This was because some of the information was not available for the first descend. The path to descend was determined by the following rules. When deciding which path to follow, the algorithm would traverse only one of the decision paths if the split feature of the current node (x_s) was in S [10]. This made sense since the current prediction x would only go to either one of the child nodes after the splitting. Else if the split feature of the current node (x_s) was not in S, then x did not split and went to both branches, each with a certain assigned weight [10].

2.4.3 Review of Existing Global Explanation Methods

Global methods described the average prediction behaviour of a machine learning model to convey useful insights such as the relative importance of each feature in overall predictions [8]. Global explanations using SHAP was the most accurate as compared to permutation importance recommend by Scikit-learn [12]. It was because Tree SHAP was not influenced by feature dependence since it used conditional expectation instead of the marginal expectation in calculating the value function, V [8]. Contrarily, permutation feature importance was influenced by feature dependence [8]. In other words, when calculating the permutation feature importance for a feature, the model prediction error diminished since other correlated features were used in the prediction, yielding a lower feature importance [8]. Basically, it could not be certain that the feature with the highest permutation feature importance was the most important feature unless there was no correlated feature.

Similar to permutation feature importance, tree SHAP global explanation could be used to plot a bar chart to convey the magnitude of the feature effects. Note that the length of the bar was representing the average absolute magnitude of SHAP values [10]. On top of that, the local explanations could be aggregated in to plot a beeswarm as shown in the diagram on the right to visualize additional information which was the direction of the effect. Taking the beeswarm plot below as an example, it could be observed that age was the most important feature with a strong positive relationship with the risk of mortality. The lower magnitude of age contributed to lower risk mortality and vice versa [10].

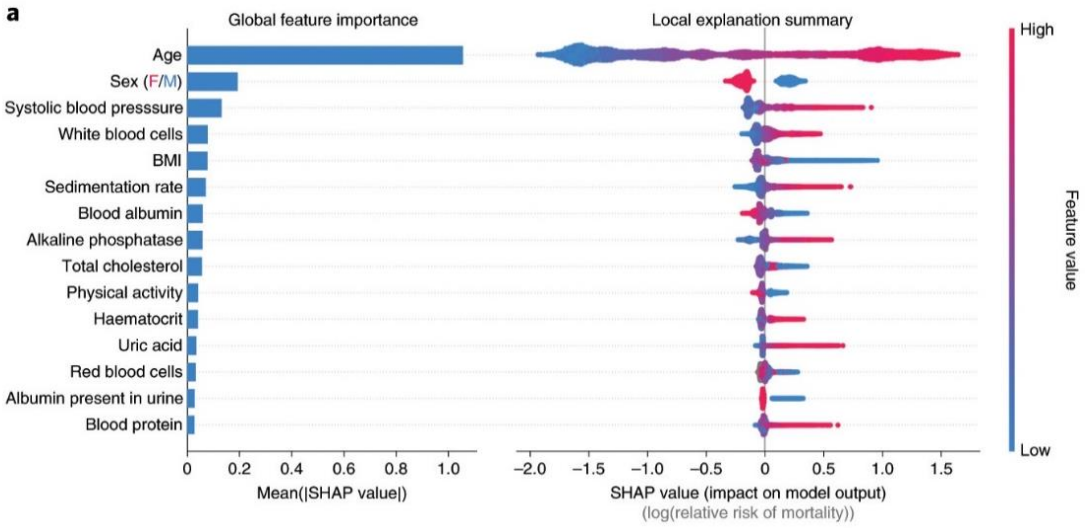


Figure 2.4.3.1: Tree SHAP global explanations: An example of bar chart (on the left) and an example of beeswarm plot (on the right)

2.5 Review of Existing Drift Detection Methods

Although a dealership management system like DealerCenter was a data-driven application with some AI capabilities, these systems cannot guarantee the dealer that the accuracy of the AI predictions in the real world. Thus, real-time AI monitoring was needed to validate whether the AI predictions were accurate and relevant over time.

There were many ways to detect the concept drift. The first way was simply by computing standard performance metrics of real-time prediction like accuracy, F1 score, recall and precision. If the standard performance metric fell below a certain threshold, the system triggered an alert and involve human intervention. Most often, the ground truth labels of the real-time prediction might not be available promptly or might not be available at all.

The second way was statistical tests. Statistical tests could represent proxy metrics by comparing distributions of the inputs and outputs to detect concept drift. According to Gama and other researchers, one way to detect the concept drift was to monitor the distributions on two different time windows [21]. To infer that there was a distribution difference between the two time windows, the null hypothesis that the two distributions were equal must be rejected with a statistically significant p-value. There were many types of statistical tests to detect whether the distribution between two-time windows was statistically different from one another. These tests were

Student's t-test, Population Stability Index (PSI), Kullback-Leibler divergence, and Jensen-Shannon Divergence, Wasserstein distance and more. The author decided to discuss Population Stability Index (PSI) and ADWIN. PSI was discussed since it was a standard measure recommended by most regulatory bodies in the field of credit risk [13], while ADWIN was discussed since ADWIN was the drift detector chosen for the Hoeffding tree and details were further discussed.

The third way was the calculation of Tree SHAP loss value to detect the drift. Lundberg and other researchers criticised that the statistical tests like PSI and ADWIN were prone to both false positives and false negatives and model performance monitoring could be prone to random fluctuations and noise data [10]. AI monitoring using Tree SHAP was free from these problems by directly calculating the contribution of each feature to the increase or loss of the model's performance. Conveniently, both the statistical tests and the Tree SHAP algorithm could be used to detect data errors as well as drift.

2.5.1 Population Stability Index

Population Stability Index (PSI) was a metric to measure the distribution shift of an independent or a dependent variable between two samples or over time [14]. The formula of population stability index was as followed, where A_i was the relative count of data in A in category i, B_i was the relative count of data in B, and K was the number of categories:

$$PSI = \sum_{i=1}^{K} \left[(A_i - B_i) \times \ln\left(\frac{A_i}{B_i}\right) \right]$$

E_i, non-relative frequency count	O_i, non-relative frequency count	A_i	B_i	$(A_i - B_i)$	$\ln\left(\frac{A_i}{B_i}\right)$	$(A_i - B_i) \times \ln\left(\frac{A_i}{B_i}\right)$
3718	154	0.11	0.06	0.05	0.5604	0.0255
3795	172	0.11	0.07	0.04	0.4704	0.0191
3239	141	0.09	0.06	0.04	0.5107	0.0188
3537	195	0.10	0.08	0.02	0.2745	0.0066
3320	189	0.09	0.07	0.02	0.2424	0.0049
3596	301	0.10	0.12	-0.02	-0.1431	0.0023

3457	298	0.10	0.12	-0.02	-0.1725	0.0032
3515	369	0.10	0.14	-0.04	-0.3696	0.0165
3444	412	0.10	0.16	-0.06	-0.5002	0.0318
3503	317	0.10	0.12	-0.02	-0.2211	0.0055
Sum = 35124	Sum = 2548	Sum = 1	Sum = 1			PSI = 0.1342

Table 2.5.1.1: Table for calculating PSI

The table above demonstrated the calculation of PSI. Given a variable of any type, 10 categories were defined, and the frequency was counted for each category in both distribution A and B. Second, for each category, the relative frequencies were obtained for both distribution A and B. Third, the $A - B$ and $\ln\left(\frac{A}{B}\right)$ were computed in each category. Finally, the PSI was obtained by the summation of $(A - B) \times \ln\left(\frac{A}{B}\right)$ for each category. The order to which the distribution was order was not important since the magnitude that quantified the difference of distributions was always positive as shown on the first column starting from the right.

Below was the interpretation of the population stability index by a range of values:

Range of value	Interpretation of distribution of the variable in 2 samples	Action
$\text{PSI} = 0$	Same	Ignore
$0 < \text{PSI} < 0.1$	Non-significant change	Ignore
$0.1 \leq \text{PSI} \leq 0.25$	Small change	Trigger warning of concept drift, further investigation was required
$\text{PSI} > 0.25$	Significant change	Alert concept drift and update model

Table 2.5.1.2: Table for interpreting PSI

The disadvantage of PSI was that PSI became unreasonably large when the binned category or category had frequency counts close to zero in either of the two distributions [15]. The second disadvantage was that choosing the number of bin categories required careful consideration and was specific to each variable. If the selected number of bin categories was too many, then the minor difference in the distribution could be more easily picked up and incorrectly classified as concept drift,

increasing the rate of false positives [15]. Else if the selected number of categories was too low, the difference in the distribution could be harder picked up and incorrectly dismissed as no difference, increasing the rate of false negative [15].

Based on the disadvantage of PSI mentioned above, for categorical variables that had less than 10 categories, other statistical measures like chi-square goodness of fit test were used to check the distribution difference. The formula for chi-squared statistics was shown as below:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

, where n was the total number of categories, O was the observed count in the category i, E was the expected count in category i, and χ^2 was the Chi-square value. Each E was readjusted such that the $\sum_{i=1}^{n} O_i$ was equal to $\sum_{i=1}^{n} E_i$, as shown below:

	Expected count, E_i	Observed count, O_i	Readjusted expected count
Category 1	3718	301	212.3188
Category 2	3795	172	216.7160
Category 3	3239	141	184.9652
Sum	10752	614	614

Table 2.5.1.3: Example of calculating Chi-squared statistics -Part I

The expected count was re-adjusted using the formula $\frac{E_i}{\sum_{i=1}^{n} E_i} \times \sum_{i=1}^{n} O_i$ for each category i. Then, the Chi squared value was calculated as shown blow:

	$E_i - O_i$	$(E_i - O_i)^2$	$\frac{(E_i - O_i)^2}{E_i}$
Category 1	-88.68	7864.35	37.0403
Category 2	44.72	1999.52	9.2264
Category 3	43.97	1932.94	10.4503
Sum	0.00	11796.81	56.7170

Table 2.5.1.4: Example of calculating Chi-squared statistics – Part II

Finally, the chi-squared value was calculated by summing the $\frac{(E_i - O_i)^2}{E_i}$ for all categories. In this case, the chi-squared value was 56.7170. The p-value of chi-square value of 56.7170 with the degree of freedom of 2 in the chi-squared distribution was $4.831e^{-13}$. The degree of freedom was 2 since the total number of

categories was three. Given that the p-value was less than 5%, an alarm was triggered since the distribution difference between the expected and observed categorical feature was significantly different.

2.5.2 **ADWIN**

The drift detector that existed in each of the base learners for the ARF algorithm was Adaptive windowing (ADWIN). ADWIN was one of the adaptive windows methods for checking distribution differences between two windows of data. Unlike most drift detectors, the window size was not a hyperparameter. Instead, as new data was added to the window, ADWIN would automatically adjust the window size by growing when the data was stationary or shrinking when a drift was detected [16]. As a result, the distribution of data after adjusting W would always represent the latest and most accurate distribution [16]. The formula of the ADWIN was discussed in depth in APPENDIX C.

However, there existed a major disadvantage. The disadvantage was that an arbitrary length of recently read data must be kept in order to check the distribution between old data and new data. This was unacceptable since each Hoeffding tree in the ARF leaner had one data drift detector and one data drift warning detector. Every unit increase in base learners meant a two unit increase in detectors. Assuming that new data was the same distribution as the existing data for quite some time, then more data had to be kept in each data drift detector, causing an explosion of storage consumptions.

Fortunately, Bifet and Gavald improved the first version of ADWIN by using a sophisticated data structure called exponential histograms [16]. The variation of this data structure improved both time and memory requirements. According to Bifet and Gavald, the data structure only tried up to $\log_2 n$ times instead up to of n times when determining the size of the new window, where n was the size of the current window [16]. The memory requirement was $O(\log_2 n)$. In short, some memory was still needed to store data to represent the latest distribution while discarding data that were not from the latest distribution.

2.5.3 Drift Monitoring using Tree SHAP

Tree SHAP could also be used to monitor AI performance and detect concept drift. The compositional approximation was implemented with Tree SHAP to

compute the model's loss function called as SHAP loss value [10]. This method would require iterating each data sample from the dataset used to compute the expectation. According to Lundberg and other researchers, this method quantified the impact of each feature on the performance without the influence from the global fluctuations of model performance caused by random noise [10]. An experiment was conducted by them and successfully proved the concept, as described below Lundberg et al.].

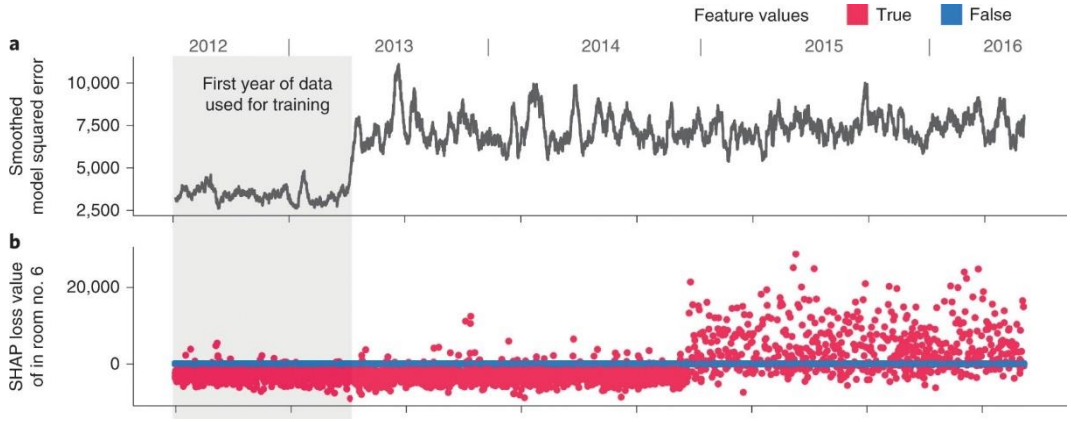


Figure 2.5.3.1: Monitoring plot using model performance (at the top) and monitoring plot using SHAP loss value (at the bottom)

The top plot showed the typical monitoring plot that visualized the model performance over time, while the bottom plot showed the monitoring plot that visualized the SHAP loss values of the feature that indicated whether the surgical procedure that happened in room no. 6 [10]. At the y-axis, the negative SHAP loss value increased the model's accuracy while the positive SHAP loss value decreased the model's accuracy. For the top plot, starting from the second year 2013 to 2016, the fluctuations of the model performance seemed natural and consistent. However, the bottom plot showed an alternative fact where there existed a clear shift from negative values to positive values for procedures that happened in room no. 6.

Regardless of the advantages, the major disadvantage was that the SHAP loss value needed truth labels and the drift detection was dependent on how fast the truth data was available. Fortunately, the true data was available after some time for car price analytics and lead scoring. If the used car was sold, then the final selling price was inserted into the system; if the lead conversion was successful or failed, then the result was inserted into the system.

2.6 Summarisation of Previous Works

In conclusion, the adaptive random forest algorithm could integrate with Tree SHAP without compromising too many memory requirements. The memory requirement was that ADWIN discarded old data and only kept data that represented the latest distribution, thus requiring some memory to store these data. Then for the online AI monitoring, it could be implemented by computing model performance, Population Stability Index (PSI), Chi-square goodness of fit test, and Tree SHAP loss. Inevitably, storage was required to store some old data for computing PSI, Chi-squared statistical test, and Tree SHAP loss. Online AI monitoring also worked during the absence of truth labels since the Population Stability Index and Chi-square goodness of fit test could serve as proxy metrics for monitoring concept drift.

CHAPTER 3 SYSTEM DESIGN

3.1 System Architecture Diagram

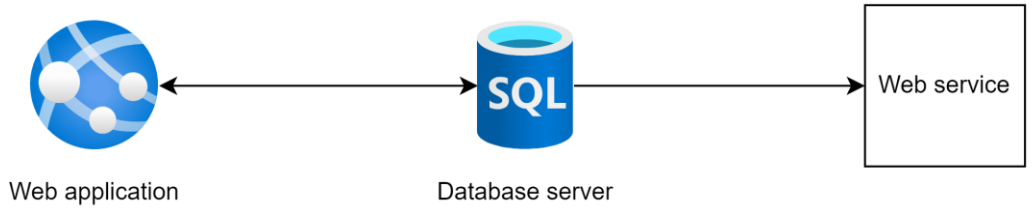


Figure 3.1.1: The architecture diagram of the car dealership system

The web application was deployed to the Azure App service. An Azure SQL server was also created to allow data communication between the web application and the web service. The web application created, read, updated, and deleted the database records to manage business information while the web service queried the most recent application data to train model, review the individual model prediction, review the individual model loss, review the model's average prediction behaviour, evaluate the model performance, and monitor drift.

3.2 Use Case Diagram for Web Application and Web Service

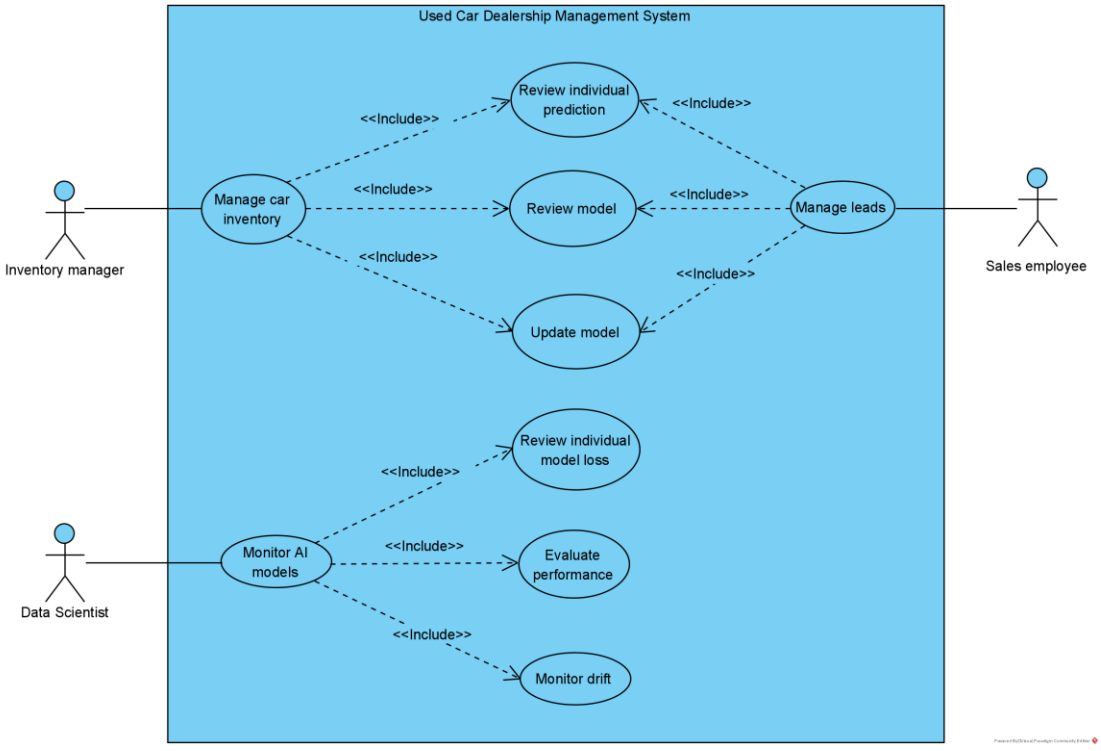


Figure 3.2.1: The use case diagram for used car dealership management system

1. Manage car inventory
    a) As an inventory manager, I want to create, read, update, and delete the car stocks to keep track of the car inventory.
    b) As an inventory manager, I want to determine the best price for a selling car with the aid of predictive analytics so that the determined price is more attractive to the prospects and fairer to justify the costs and profits of the business.
    c) As an inventory manager, I want to review the car price model so that I can trust the car price model to use the right features to predict the car prices for me.
    d) As an inventory manager, I want to update the car price model so that the model can remain relevant and high performing under the influence of drift.

2. Manage leads
    e) As a sales employee, I want to create, read, update, and delete the basic information about my potential customers so that I can convert them later.
    f) As a sales employee, I want to know which prospects that I can most likely successfully convert so that I can save my resources from serving non-potential customers.
    g) As a sales employee, I want to review the lead scoring model so that I can trust the lead scoring model to use the right features to predict the lead scores for me.
    h) As a sales employee, I want to update the lead scoring model so that the model can remain relevant and high performing under the influence of drift.

3. Monitor AI models
    a) As a data scientist, I want to review individual model loss so that I can know which features contribute the most to the prediction value and whether the features are accurate or inaccurate in predicting the value.
    b) As a data scientist, I want to evaluate the performance of the deployed AI models so that I can be reassured that the online performance of deployed models does not deteriorate, with or without the influence of drift.
    c) As a data scientist, I want to monitor the drifts so that I can observe the change in the distribution of variables overtime and determine whether to add or delete certain independent variables to further improve a model's performance.

3.3 Activity Diagram for Web Application and Web Service

3.3.1 Reviewing Individual Predictions

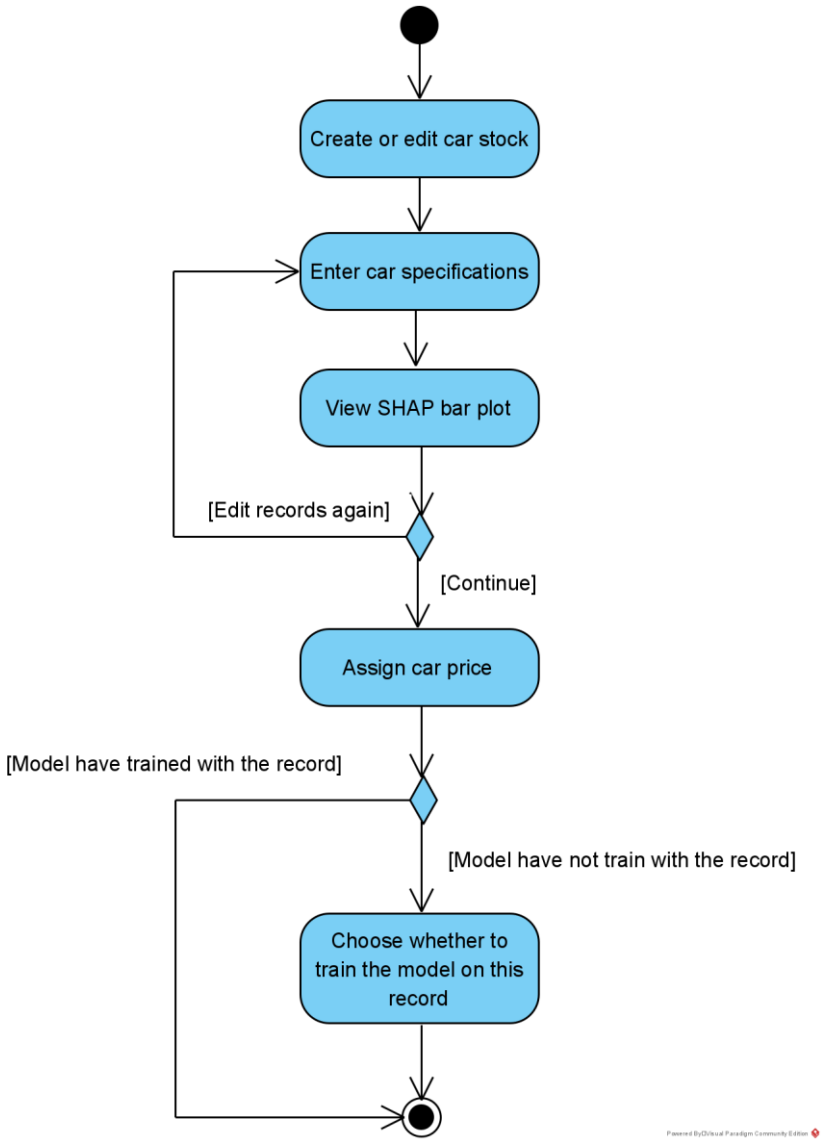


Figure 3.3.1.1: The activity diagram for reviewing individual predicted car price

Based on the diagram above, as the inventory manager was creating and editing the car stock, he or she could review the predicted car price by viewing the SHAP bar plot that was generated by the web service. Based on the predicted car price, the inventory manager could assign his or her own choice of car price as the selling car price. If the inventory manager would like to train the model on this record, then he or she could select the option “update the car price analytics” to do so. After the model had been trained with this record, the inventory manager could not choose the train the model with this record again even it has updated. It was because the

model could not untrain itself. Instead, the inventory manager could copy the car specifications to a new record and deleted the current record to retrain the model.

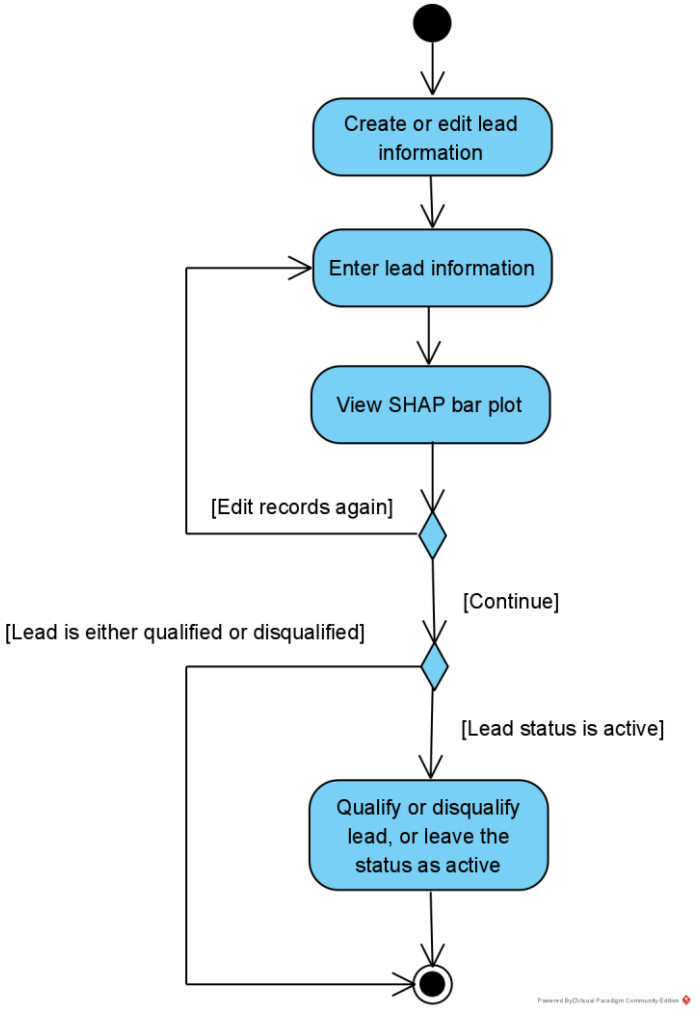


Figure 3.3.1.2: The activity diagram for reviewing individual predicted lead score

Based on the diagram above, as the sales employee was creating and editing the lead information, he or she could review the predicted lead score by viewing the SHAP bar plot that was generated by the web service. The sales employee could then use the predicted lead score to set the priorities on which leads to convert. If the attempts to convert the lead had failed, then the sales employee updated the lead status as “Disqualified”. Else if the lead conversion was successful, then the sales employee updated the lead status as “Qualified”. As soon as the lead status had been updated as “Qualified” or “Disqualified”, the model would automatically train on the current lead record. Any subsequent edits on the current lead record would not cause the model to re-train again since the model could not untrain the current record.

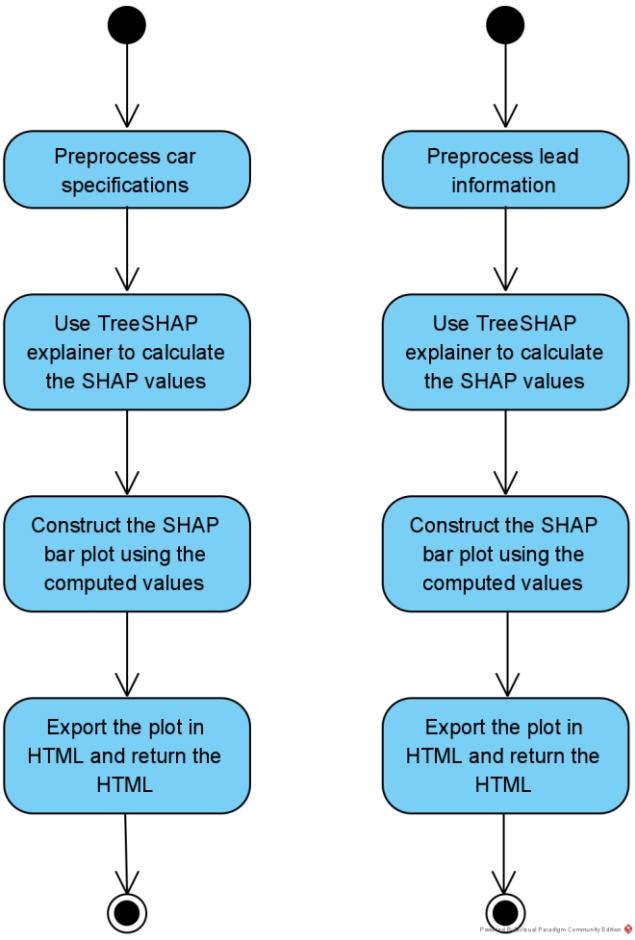


Figure 3.3.1.3: The activity diagram for constructing SHAP bar plots

Based on the diagram above, in order to generate the SHAP bar plots needed in the web application, the web service required the features that were submitted by the client when creating or editing the record. First, the web service preprocessed the car specifications and lead information, respectively. Then, the web service used the existing Tree SHAP explainers to calculate the SHAP values for both preprocessed car specifications and lead information, respectively. The SHAP values were then used to construct the SHAP bar plots. Finally, the plots were converted to HTML code and send back to the web application.

3.3.2 Reviewing Models

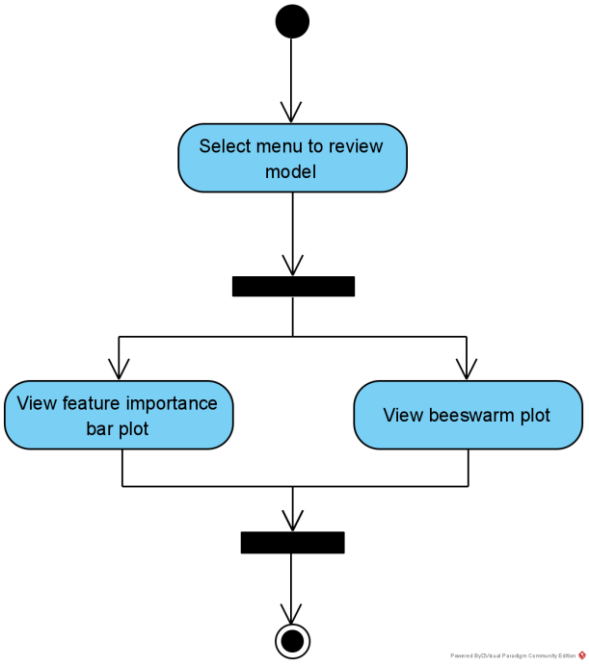


Figure 3.3.2.1: The activity diagram for reviewing car price model and lead scoring model

Based on the diagram above, the inventory manager and sales employee could navigate in the menu to review the car price model and lead scoring model, respectively. The web service then computed the SHAP value to display feature importance bar plot and beeswarm plot to visualize the average prediction behaviour of both models, respectively.

Based on the diagram below, in order to generate the beeswarm plots and feature importance bar plots that were needed in the web application, the web service was required to query all the recent application data that had the truth. For car inventory record, the records with truth were the records that had the UpdateAnalytics value set to “Yes”. For lead information, the records with truth were the records that had the Status value set to “Qualified” or “Disqualified”. First, the web service preprocessed the car specifications and lead information, respectively. Then, the web service used the existing Tree SHAP explainers to calculate the SHAP values for both preprocessed car specifications and lead information, respectively. The SHAP values were then used to construct the beeswarm plots and feature importance bar plots. Finally, the plots were converted to HTML code and send back to the web application.

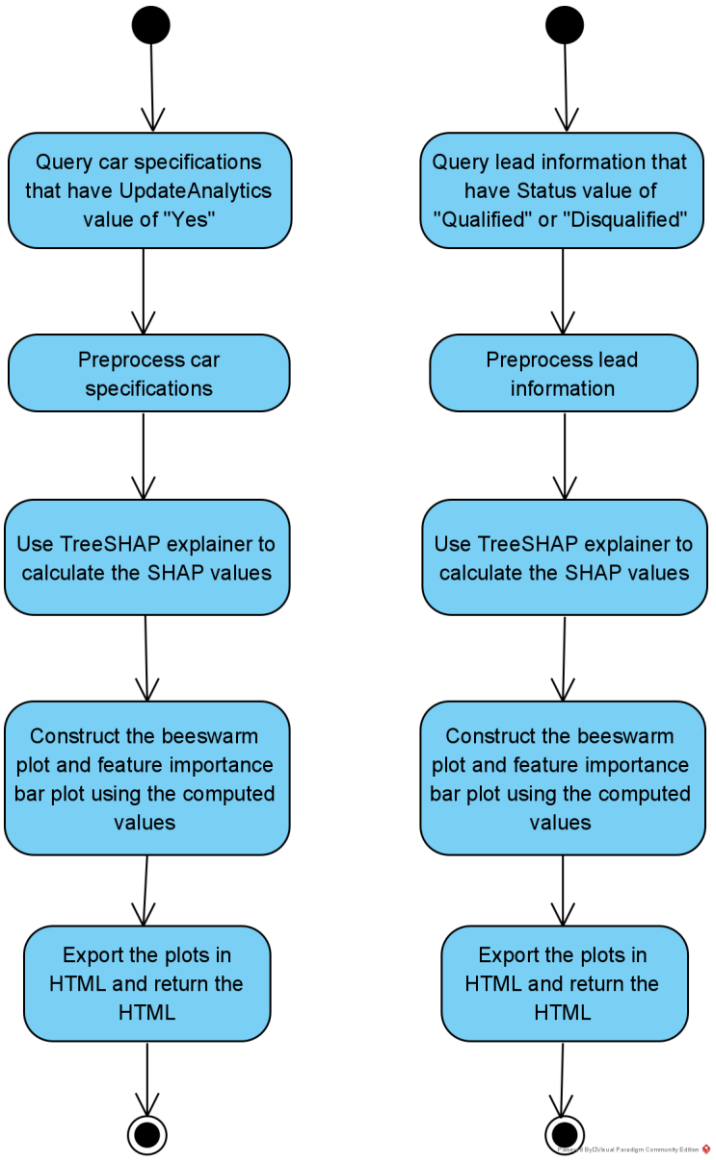


Figure 3.3.2.2: The activity diagram for constructing beeswarm plots and feature importance bar plots

3.3.3 Update Model

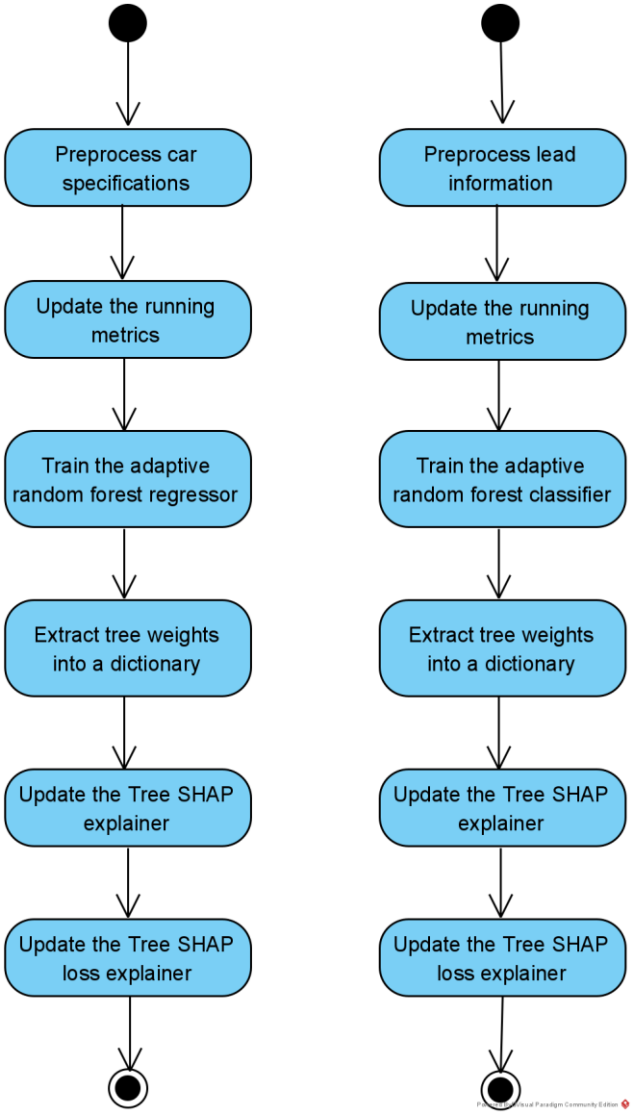


Figure 3.3.3.1: The activity diagram for incrementally training the adaptive random forest models

New samples were available to train car price adaptive random forest regressor whenever the inventory manager selected the option “update the car price analytics” in a car inventory record. On the other hand, new samples were available to train lead scoring adaptive random forest classifier whenever the sales employee update the lead status to either “Qualified” or “Disqualified”. Both adaptive random forest models only trained with one sample at a time. First, the sample was used to update test-then-train running metrics before training the models on the same sample. After the training, the tree weights were extracted from the models to a dictionary. It was because Tree SHAP explainer did not directly support River models but did support tree-like models by passing in a dictionary. The dictionary was then used to

update both Tree SHAP explainer and Tree SHAP loss explainer. The further detail of the tree weight extraction process was discussed in Section 3.6.

3.3.4 Reviewing Individual Model Loss

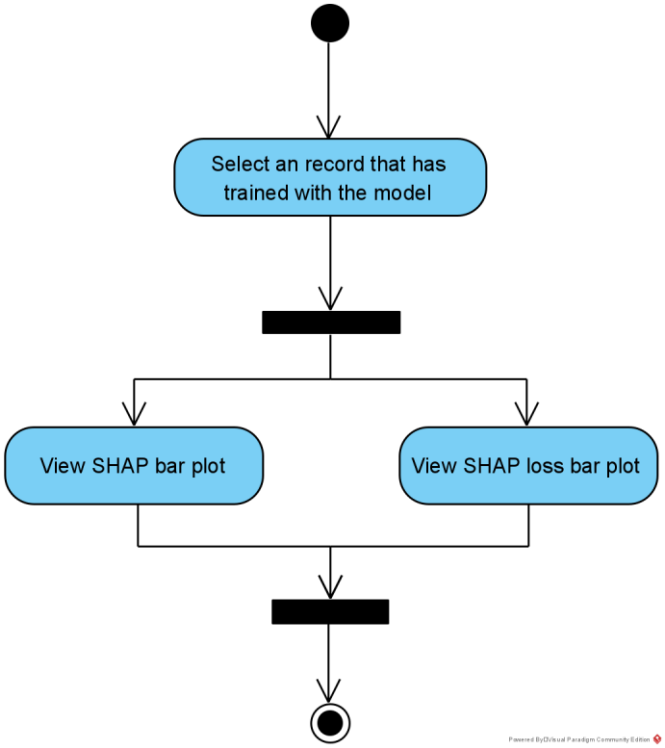


Figure 3.3.4.1: The activity diagram for reviewing individual model loss

Based on the diagram above, the data scientist could select a car inventory record or a lead record that had the truth to review the individual model loss of that record. The web service then displayed the SHAP bar plot and SHAP loss bar plot to visualize the local feature importance and which features were positively or negatively contributed to the model loss, respectively.

Based on the diagram above, in order to generate the SHAP bar plots and SHAP loss bar plot needed in the web application, the web service required the features that were submitted by the client when viewing the record that had the truth. First, the web service preprocessed the car specifications and lead information, respectively. Second, the web service used the existing Tree SHAP explainers to calculate the SHAP values for both preprocessed car specifications and lead information, respectively. Third, the web service also used the existing Tree SHAP loss explainers to calculate the SHAP loss values. The SHAP values and SHAP loss values were then used to construct the SHAP bar plots and SHAP loss bar plots,

respectively. Finally, the plots were converted to HTML code and send back to the web application.

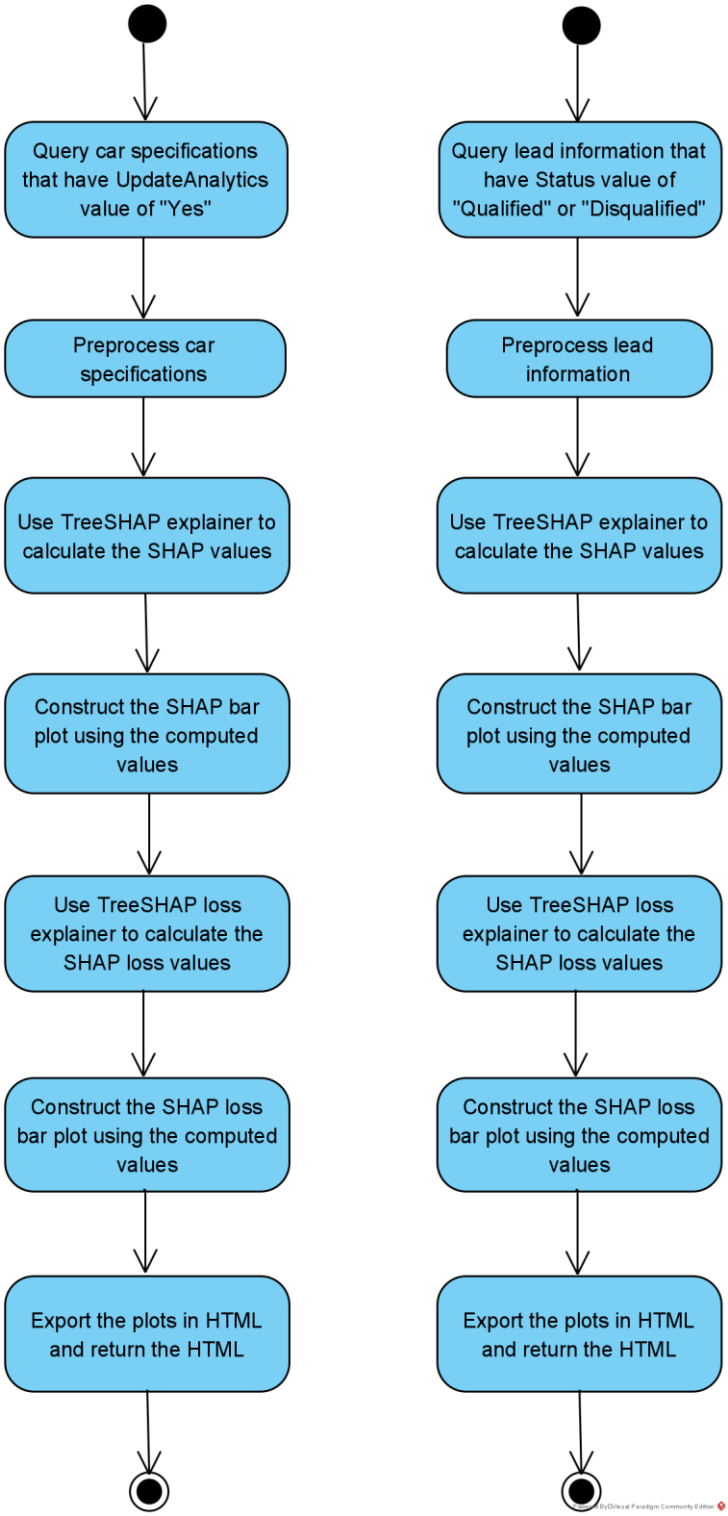


Figure 3.3.4.2: The activity diagram for constructing the SHAP bar plot and SHAP loss bar plot

3.3.5 Evaluating Performance

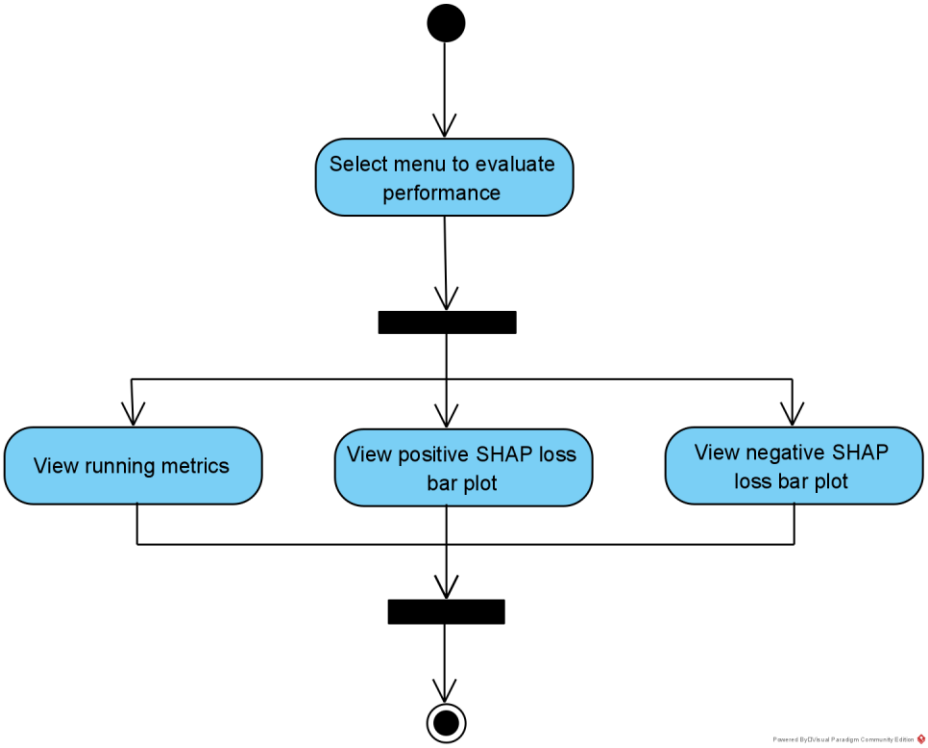


Figure 3.3.5.1: The activity diagram for evaluating model performance

Based on the diagram above, the data scientist could navigate in the menu to review the performance of both car price model and lead scoring model. The web service then displayed the running metrics to visualize the recent model performance on the application data. The web service also displayed the positive and negative SHAP loss bar plot to visualize which features were positively or negatively contributed to the model loss.

Based on the diagram below, in order to generate the plots that were needed in the web application, the web service was required to query all the recent application data that had the truth. First, the web service preprocessed the car specifications and lead information, respectively. Second, the web service used the existing Tree SHAP loss explainers to calculate the SHAP loss values for both preprocessed car specifications and lead information, respectively. Third, the SHAP loss values were then used to construct the positive and negative SHAP loss bar plots. Fourth, the running metrics were retrieved from the adaptive random forest models to plot running R-squared for regression and plot running AUC-ROC for classification. Finally, the plots were converted to HTML code and send back to the web application.

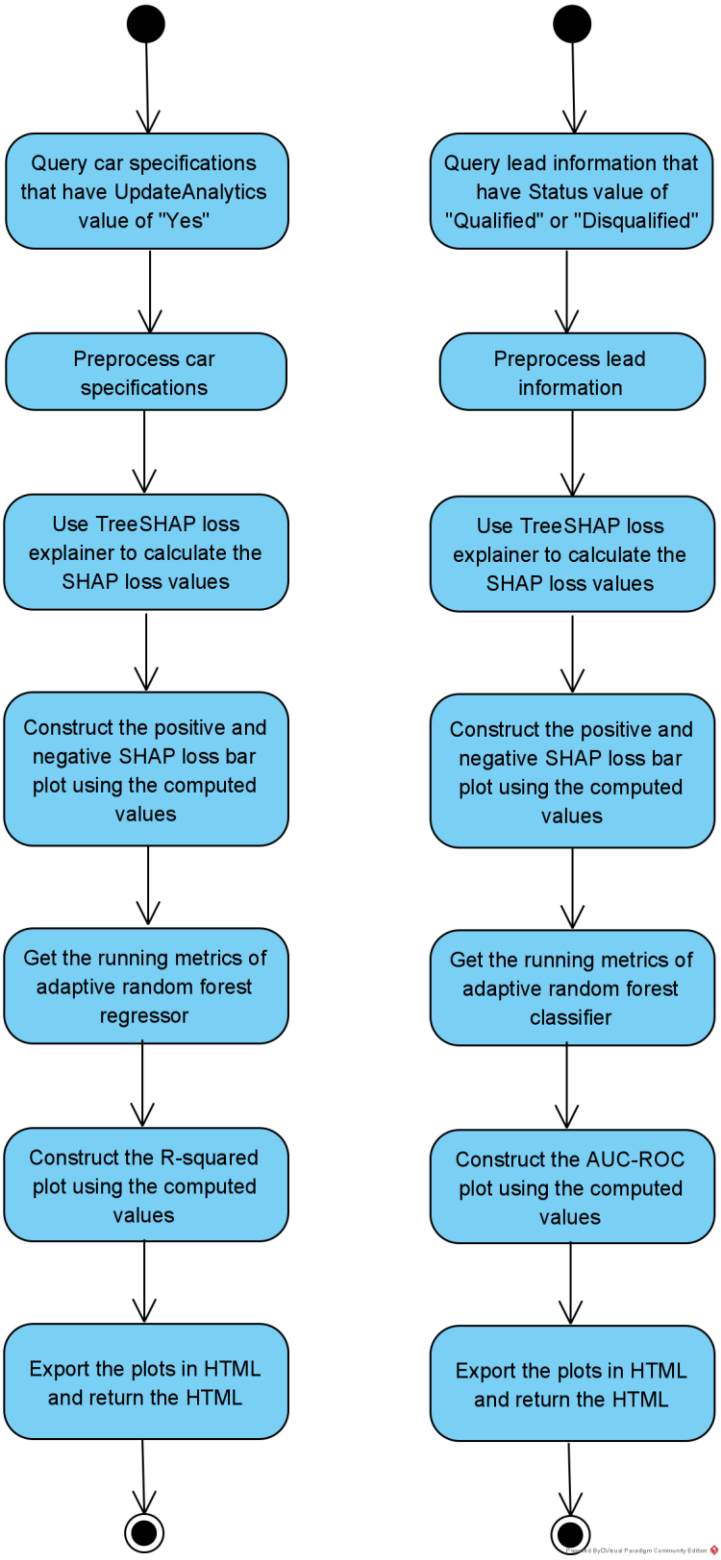


Figure 3.3.5.2: The activity diagram for constructing plots that evaluate model performance

3.3.6 Monitoring Drift

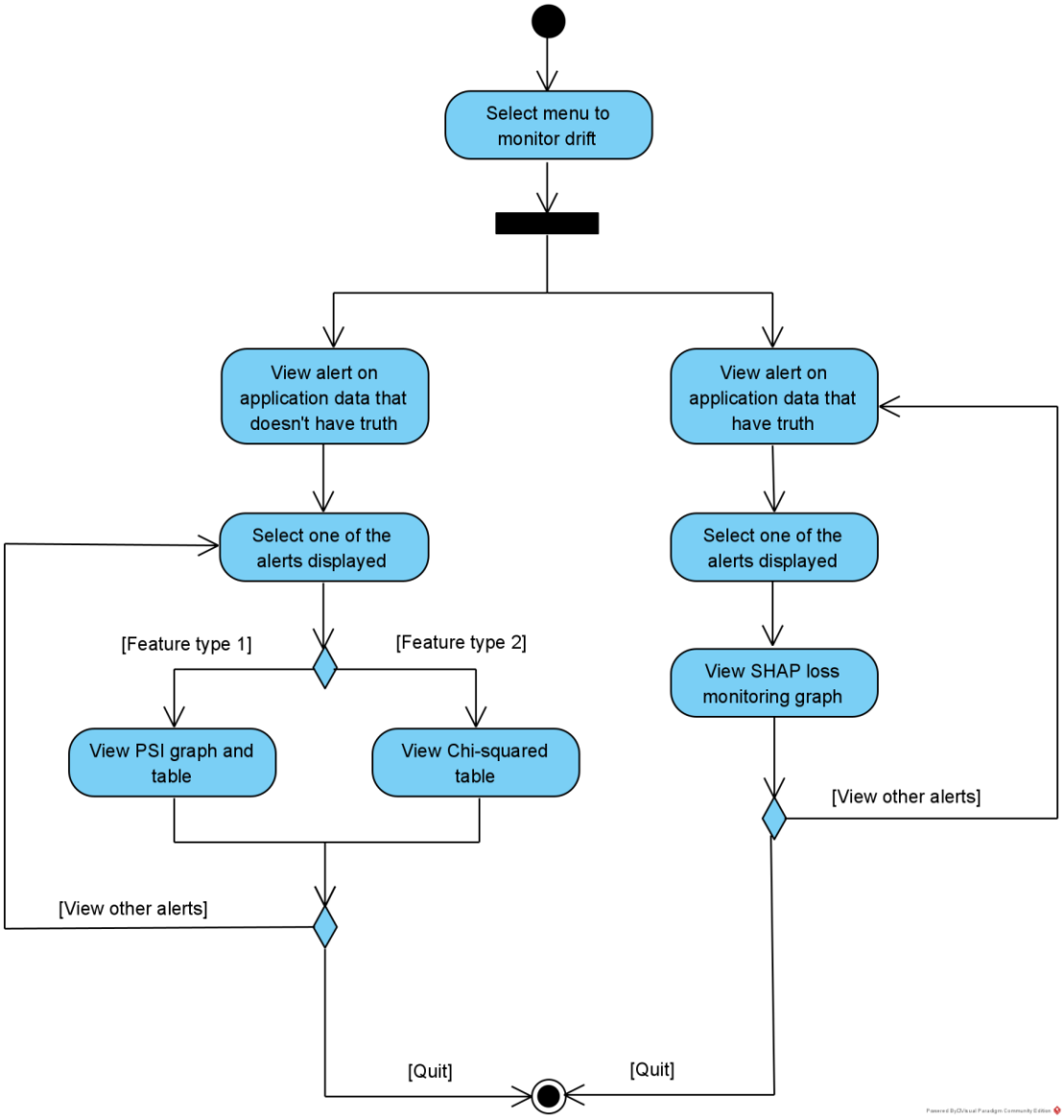


Figure 3.3.6.1: The activity diagram for monitoring drift on both records with truth and without truth

Based on the diagram above, the data scientist could navigate in the menu to monitor the drift for both records with and without truth.

For monitoring car inventory records or lead records that had truth, the webpage would show the SHAP loss monitoring graphs and their corresponding alarm information only for features that had drifted.

For monitoring the records that had not truth, the webpage would show the PSI graphs, PSI tables or chi-squared table for features that had drifted. As shown in the diagram above, the PSI was used to check for drifts if feature was "[Feature type

1]". Else, the chi-squared goodness of fit tests was used to check for drifts if feature was "[Feature type 2]". A feature was considered as "[Feature type 1]" when the feature was numerical, or the feature was categorical with 10 or more categories. On the other hand, a feature was considered as "[Feature type 2]" when the feature was categorical with less than 10 categories.

The two diagrams below showed the activity diagrams for constructing SHAP loss monitoring plots based on car inventory records and lead records, respectively. In order to generate the SHAP loss monitoring plots, the web service was required to query all the recent application data that had the truth. First, the web service preprocessed the car specifications and lead information, respectively. Second, the web service used the existing Tree SHAP loss explainers to calculate the SHAP loss values for both preprocessed car specifications and lead information, respectively. Then, each car specification and lead attribute were iterated to check for the drift. For each iteration, the proposed SHAP loss monitoring function was called to check for any statistically significant difference in SHAP loss means or difference in feature distribution between the validation set and the queried data. If the difference was statistically significant with a p-value of less than 5%, then the alarm information was saved and SHAP monitoring plot were constructed before converting to HTML code. The alarm informed the data scientists on the type of statistical tests that triggered the alarm, and the splitting point between the validation data and recent application data where the mean difference was statistically significant.

In addition, to validate the proposed SHAP loss monitoring function, an experiment was conducted to test the effectiveness of the proposed function as compared to the preliminary function developed by Scott Lundberg. Hence, the author deliberately induced drift and data errors in the car specifications after every query. The experiment was discussed in further detail in Section 5.5.1.

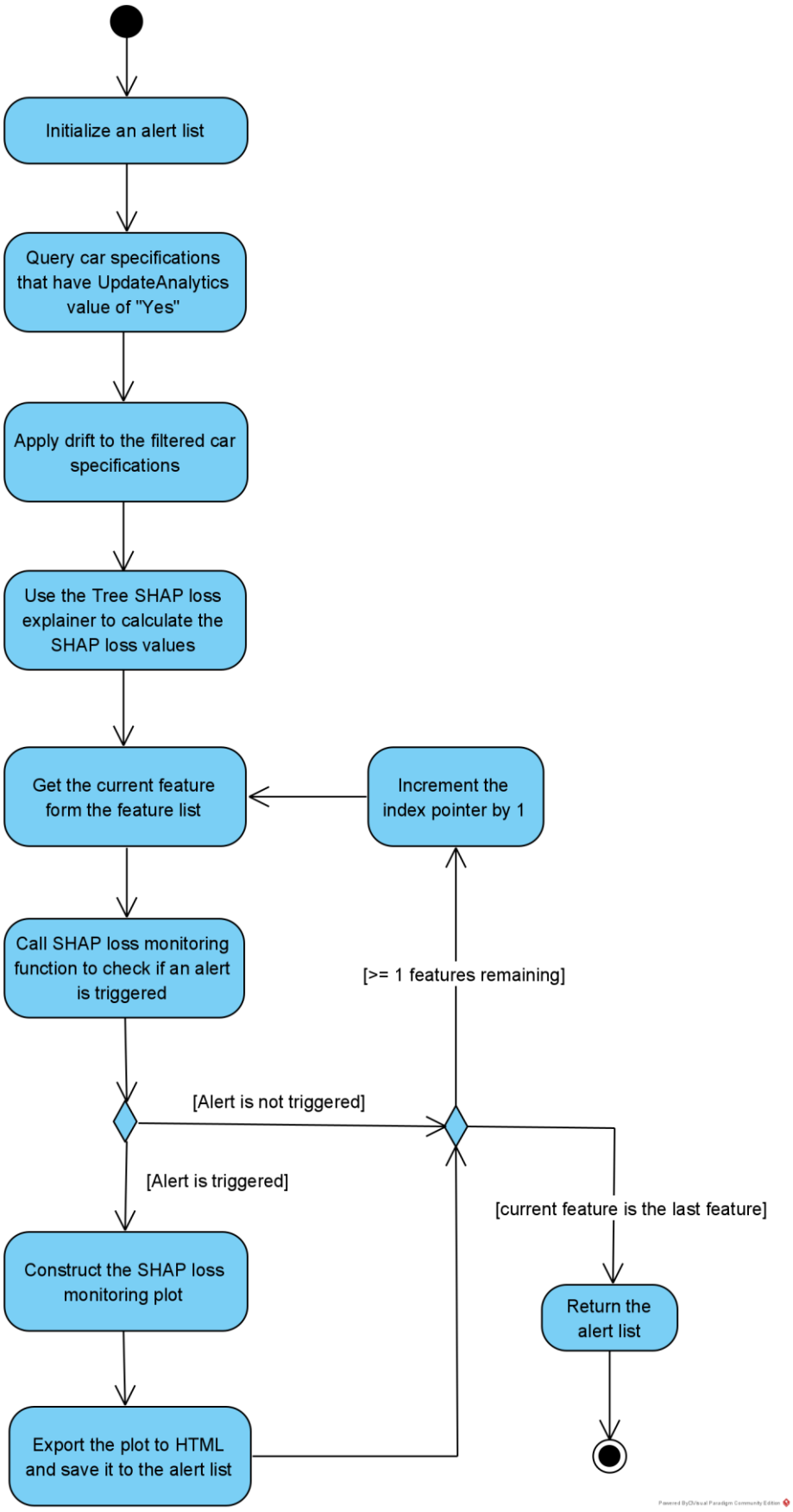


Figure 3.3.6.2: The activity diagram for constructing SHAP loss monitoring plot (car price inventories)

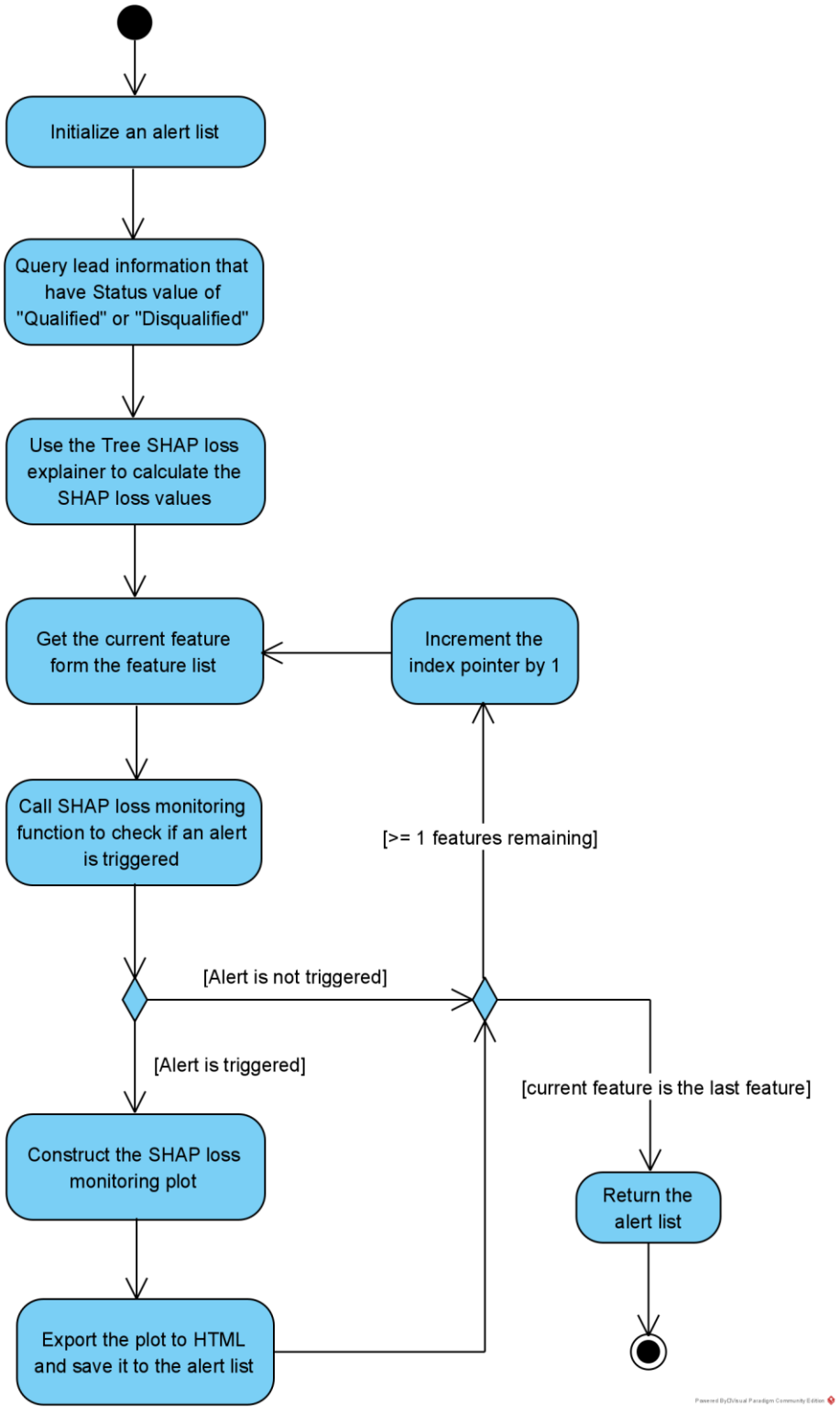


Figure 3.3.6.3: The activity diagram for constructing SHAP loss monitoring plot (lead records)

The two diagrams below showed the activity diagrams for constructing PSI graphs, PSI tables and chi-squared tables based on car inventory records and lead records, respectively. In order to generate these plots, the web service was required to query all the recent application data that had no truth. First, the web service preprocessed the car specifications and lead information, respectively. Second, each car specification and lead attribute were iterated to check for the drift. For each iteration, depending on the feature type, the PSI or the chi-squared goodness of fit test was used to check for significant distribution difference between the validation set and the application data. If the difference was statistically significant with a p-value of less than 5%, then the corresponding tables and plots were constructed before converting to HTML code.

In addition, an experiment was conducted to validate that the PSI and chi-squared goodness of fitness test could detect drifts or data errors. Hence, the author deliberately induced drift and data errors in both car specifications and lead records after every query. The experiments could be found in "jupyter_notebooks/ FYP2_Model_Monitoring_without_Truth.ipynb".

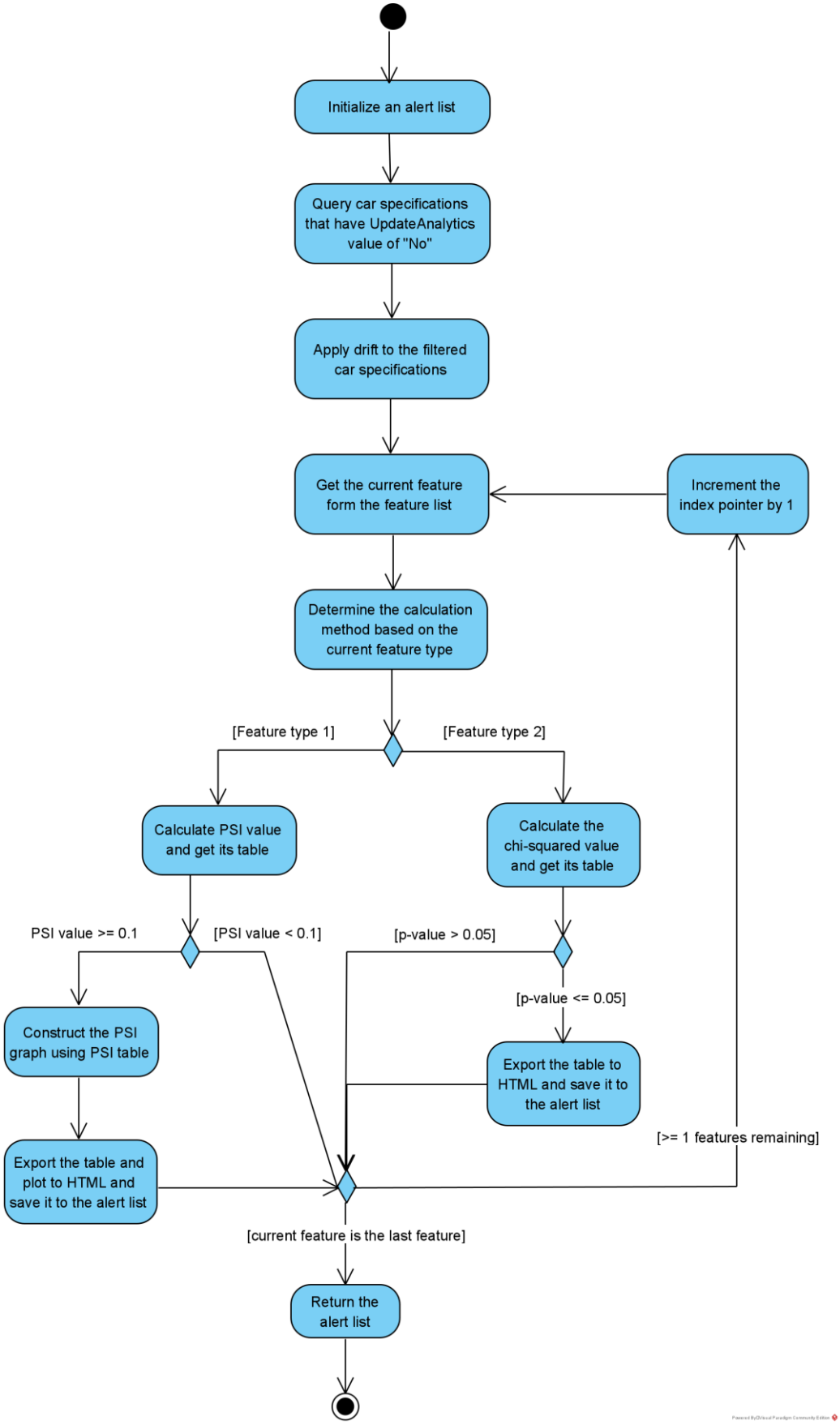

Initialize an alert list
Query car specifications that have UpdateAnalytics value of "No"
Apply drift to the filtered car specifications
Get the current feature form the feature list
Increment the index pointer by 1
Determine the calculation method based on the current feature type
[Feature type 1]
[Feature type 2]
Calculate PSI value and get its table
Calculate the chi-squared value and get its table
PSI value >= 0.1
[PSI value < 0.1]
[p-value > 0.05]
[p-value <= 0.05]
Construct the PSI graph using PSI table
Export the table to HTML and save it to the alert list
Export the table and plot to HTML and save it to the alert list
[>= 1 features remaining]
[current feature is the last feature]
Return the alert list

Figure 3.3.6.4: The activity diagram for constructing PSI graph, PSI table and chi-squared table (car price inventories)

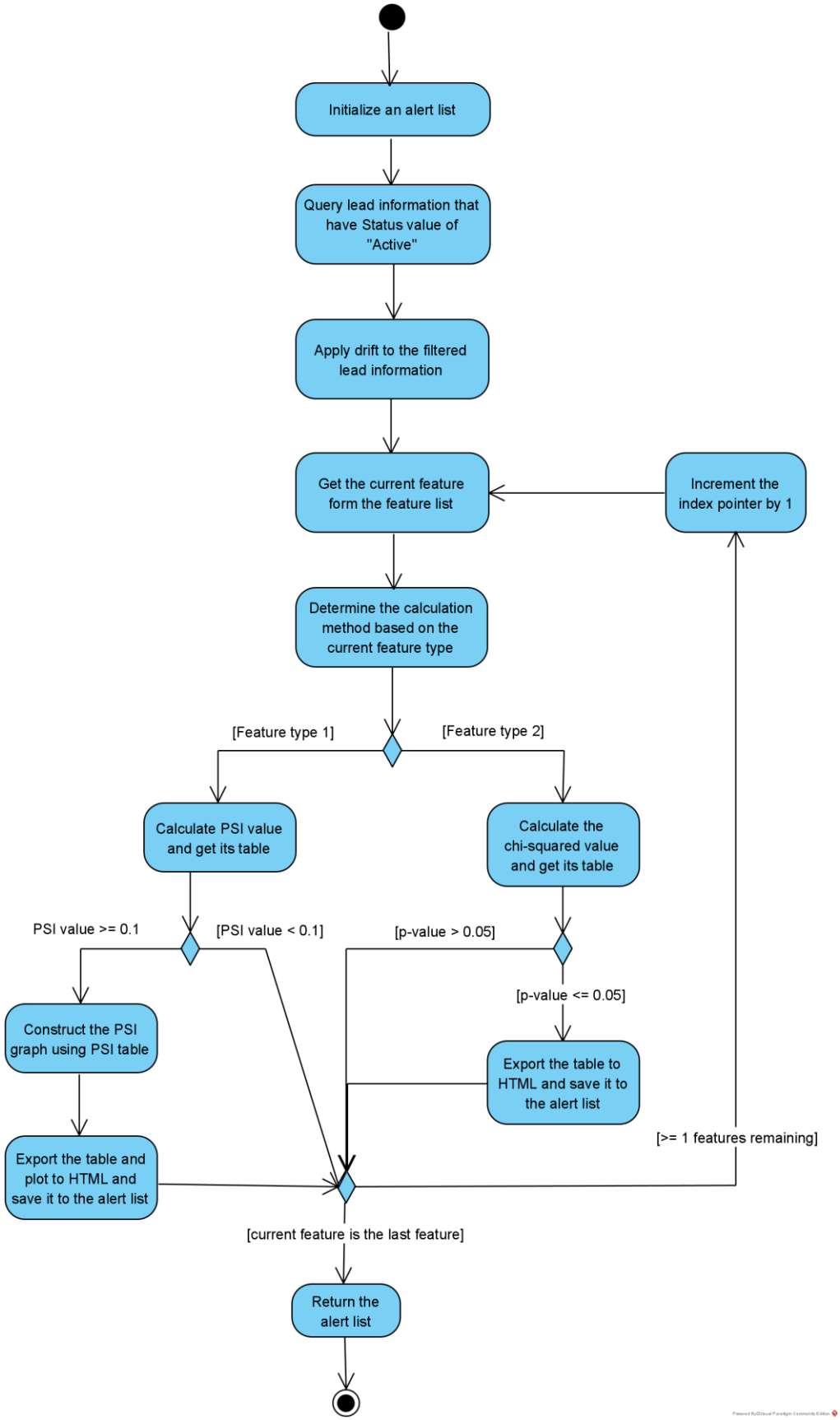


Figure 3.3.6.5: The activity diagram for constructing PSI graph, PSI table and chi-squared table (lead records)

3.4 Database Design

The diagram above showed the design of the database. The car inventory records were normalized and stored into three separate table which were “CarBrands”, “CarModels”, and “Cars”. These “Cars” table also stored “Title”, “CreatedTimestamp”, “PricePerMonth”, and “PredictedPrice” besides storing the car specifications that were used to train the car price model. The extra columns were only used within the web application itself to store relevant information that was useful to the users. In addition, the “Leads” also stores the “Name”, “Email”, “PhoneNo”, “DontCall”, and “PredictedScore” for the same purpose.

In the “Cars” table, the “UpdateAnalytics” column was a flag to indicate whether the application users wanted to train the existing car price model with the current inventory record. In other words, the records with the “UpdateAnalytics” set to “Yes” contained truth and vice versa. In the “Leads” table, the records contained truth if the “Status” was set to “Qualified” or “Disqualified”. The “Status” with the value set to “Active” indicated the opposite case. In both tables, the “CreatedTimestamp” column provided temporal information required to query the most recent records starting from a specified time.

3.5 Transfer Learning

In FYP I, an experiment was conducted, and it was observed that the initial performance of both the adaptive random forest regressor and classifier were poorer than the traditional random forest regressor and classifier. It could be said that the practicability of the proposed web service was greatly reduced since even the model performance had already been an issue at the first place. Hence in FYP II, the author proposed two transfer learning algorithms for adaptive random forest regressor and classifier, respectively.

Basically, in Layman's terms, the algorithms copied the whole tree structure and its trained weights from high-performant traditional random forest regressor and classifier to the adaptive random forest regressor and classifier, in order to boost the initial performance of these incremental tree learners. However, the transfer of the whole tree structure was not as straightforward as it sounded since the source model was a batch learner, and the destination model was an incremental learner.

3.5.1 Design Considerations

First, it was important to ensure that the prediction values represented the same concept and theory for both decision trees and Hoeffding trees. In the case of classifiers, the prediction value of the decision tree classifiers was an array of two numbers. For the leaf nodes, the two numbers were used to predict the class label using majority rule. For binary classification, if the left-side number was larger than right-side number, then the predicted class label was negative and vice versa. Since the prediction logic of Hoeffding tree classifiers was also like that, the prediction value was directly copied to the newly created branch node's object and leaf node's object into the Hoeffding tree classifiers during transfer learning.

In the case of regressors, the prediction value of decision tree regressors was directly used as the predicted output. However, the prediction value could not be directly copied to the newly created branch node's object and leaf node's object in the Hoeffding tree regressors during transfer learning. It was because the Hoeffding tree regressors did not directly store the prediction value. Instead, the Hoeffding tree regressors used "river.stats.Var" class instance to store statistical information on the target values that arrived at the current node. The statistics was then used to calculate the prediction value. Thus, instead of copying prediction values, a "river.stats.Var"

class instance must be initialised and updated with the target value of the samples that arrived at the current node.

Second, it was important to ensure that the split conditions of the decision tree and Hoeffding tree were the same. During the weights transfer, the split value in the Hoeffding trees were overridden with value 0 whenever the corresponding split feature was categorical. It was because split conditions were different for both Scikit-learn implementation of decision tree and River implementation of Hoeffding tree. For decision trees, the split value was fixed at 0.5 and the leaf node's traversal happened when the feature value was less than 0.5. However, for Hoeffding trees, the split value was set at a categorical value and the leaf node's traversal happens when the feature value was exactly equivalent to the split value. Thus, it was assumed all the nominal features were one-hot encoded, such that the Hoeffding trees' split value was always 0.5 for easier implementation of transfer learning.

Last, it was important to ensure that the adaptive random forests that had undergone transfer learning could resume training just like the ones that were trained from scratch. From the official source code, it was observed that the algorithm updated the statistical information and splitters information based on the new samples that arrived at the leaf node in each base learner. The statistical information was important for leaf nodes to output predictions, while the splitters information was important for leaf nodes to make split attempts during incremental learning. Hence, the training must be conducted manually by calling the learning function, in this case, the "learn_many" function to update the information in order for the pre-trained adaptive random forests to work as expected.

3.5.2 Adaptive random forest classifier

Below showed the pseudocode of the transfer learning algorithm for adaptive random forest classifier. The source code for transfer learning classifier algorithm could be found at jupyter_notebooks/arf_cf_transfer_learning.py.

```
FOR each hoeffding_tree
   ENQUEUE root_node to queue

   WHILE queue is not empty
      DEQUEUE node from queue
```

```
    Retrieve left and right child indexes from the corresponding node in
decision_tree
    Retrieve prediction_value from the corresponding node in decision_tree

    IF left child index is -1 (node is a leaf node)
      CALL learn_many to update the node's statistics and splitters
    ELSE
      Retrieve split feature and split value from the corresponding node in
decision_tree

      IF split feature is categorical
        Update the split value to 0

      CREATE branch_node and two child_node
      ATTACH the two child_node to the branch_node
      REATTACH branch_node to its parent to update object reference

      Update the statistics at the branch_node with the prediction_value

      ENQUEUE the two child_node into queue
```

Limitation

The transfer learning classifier algorithm had few limitations. First, the algorithm only supported the transfer of weights between binary classifiers, but not multiclass classifiers or multilabel classifiers. Second, the algorithm also only supported the update of two specific splitters in Hoeffding tree. The two splitters were the GaussianSplitter for the numerical splitter and NominalSplitterClassif for nominal splitter. Third, the algorithm only supported the weights transfer to Hoeffding trees that were configured to use majority class leaf prediction mechanism, but not Naive Bayes mechanism or Naive Bayes Adaptive mechanism. Forth, the pre-trained Hoeffding trees could only perform binary split, but not multi-way split. The Hoeffding tree was enforced as such since Scikit-learn decision tree only performed binary split.

In addition, the hyperparameters must be further limited to ensure the compatibility of the River adaptive random forest classifier with SHAP tree explainer from SHAP library. First, “disable_weighted_vote” must be set to False to disable weighted vote prediction. It was because the model that was ingested into SHAP tree explainers could not perform weighted vote prediction. If the hyperparameter was not

correctly set, then the SHAP-ingested model would output different prediction values, which caused the SHAP value calculation to be inaccurate.

Based on the limitation above, the hyperparameters of adaptive random forest classifier were limited to the values as shown below:

Based on the limitation above, the hyperparameter of TRF and ARF must be set as shown below:

```
river_arf_cf_params = {
    'n_models': len(trf_cf.estimators_), # Number of base learners must be the same with TRF
    'max_depth': None, # Should be equal or larger than max_depth in decision tree
    'binary_split': True, # Enforce binary split and disable multi-way split
    'leaf_prediction': 'mc', # Perform Majority Class prediction at the leaves
    'splitter': None, # Use tree.splitter.GaussianSplitter to monitor the class statistics and pe
rform splits

    # Hyperparameter that is not fully tested
    'merit_preprune': False # Disable merit-based tree pre-pruning.
}
```

Figure 3.5.2.1: The required hyperparameter of the adaptive random forest classifier for the transfer learning to work as expected

3.5.3 Adaptive random forest regressor

Below showed the pseudocode of the transfer learning algorithm for adaptive random forest regressor. The source code for transfer learning regressor algorithm could be found at jupyter_notebooks/arf_rg_transfer_learning.py.

```
FOR each hoeffding_tree
   ENQUEUE root_node to queue

   WHILE queue is not empty
      DEQUEUE node from queue

      Retrieve left and right child indexes from the corresponding node in decision_tree

      IF left child index is -1 (node is a leaf node)
         CALL learn_many to update the node's statistics and splitters
      ELSE
         Retrieve split feature and split value from the corresponding node in decision_tree

         IF split feature is categorical
            Update the split value to 0

         CREATE branch_node and two children_node
         ATTACH the two children_node to the branch_node
```

REATTACH branch_node to its parent to update object reference

CALL update_parent_stats to update the branch_node's statistics

ENQUEUE the two children_node into queue

Similar to the proposed transfer learning classifier algorithm, the proposed transfer learning regressor algorithm had few limitations. First, the algorithm only supported the transfer of weights between single output regressors, but not multi-output regressors. A multi-output regressor was a regressor that predicted more than one numerical variable using the same set of features. Second, the algorithm also only supported the statistics update of two specific splitters in Hoeffding tree. The two splitters were the "EBSTSplitter" for the numerical splitter and "NominalSplitterReg" for nominal splitter. Third, the algorithm only supported the weights transfer to Hoeffding trees that were configured to use target mean leaf prediction mechanism, but not model mechanism or adaptive mechanism. Fourth, the pre-trained Hoeffding trees must only perform binary split, but not multi-way split. The reason that the Hoeffding tree was enforced as such was because the scikit-learn decision tree only performed binary split.

In addition, the hyperparameters must be further limited to ensure the compatibility of the River adaptive random forest regressor with SHAP tree explainer from SHAP library. First, the "disable_weighted_vote" must be set to False to disable weighted vote prediction. Second, the "aggregation_method" must be set to mean. It was because the model that was ingested into SHAP tree explainers outputted prediction values by averaging the equally weighted predictions across all base learners. If these two hyperparameters were not correctly set, the SHAP-ingested model would output different prediction values, which caused the SHAP value calculation to be inaccurate.

Based on the limitation above, the hyperparameters of adaptive random forest regressor were limited to the values as shown below:

Based on the limitation above, the hyperparameter of TRF and ARF must be set as shown below:

```
river_arf_rg_params = {
    'n_models': len(trf_rg.estimators_), # Number of base learners must be the same with TRF
    'max_depth': None, # Should be equal or larger than max_depth in decision tree
    'binary_split': True, # Enforce binary split and disable multi-way split
    'leaf_prediction': 'mean', # Use the target mean as the prediction mechanism at leaves
    'splitter': None, # Use tree.splitter.EBSTSplitter to monitor the class statistics and perfor
m splits

    # Hyperparameter that is not fully tested
    'merit_preprune': False #  Disable merit-based tree pre-pruning.
}
```

Figure 3.5.3.1: The required hyperparameter of the adaptive random forest regressor for the transfer learning to work as expected

3.5.4 Performance Improvement

In River library, the adaptive random forest classifiers and regressors only implemented “learn_one” but did not implement any similar function like “learn_many” to learn a small subset of samples at the same time. Hence, the author vectorized the numerical operations inside the “learn_one” using Numpy to provide some performance boost. Besides, the author also used Joblib to enable process-based parallelism. The tree weight transfers between ensembles were broken down into smaller tasks. For each small task, a process was spawned to transfer weights from a decision tree to a Hoeffding tree.

3.6 Model Tree Weight Extraction

For initializing Tree SHAP explainer, the function internally extracted the tree weights from an API-specific tree models before storing the information in the "TreeEnsemble" class instance. The function directly supported the weights extraction of tree models from commonly used API like Scikit-learn and XGBoost. However, the function did not directly support the weight extraction of adaptive random forest regressor and classifier from River API. Instead, the function accepted a Python dictionary that stored the tree weights which were "children_left", "children_right", "features", "thresholds", "node_sample_weight", and "values". Thus, the tree weights must be manually extracted into a dictionary.

Below showed the pseudocode of extracting the tree weights of adaptive random forest regressor and adaptive random forest classifier into a dictionary. The algorithm was validated as mentioned in Section 5.3.

```
INIT arf_dict
INIT arf_dict["internal_dtype"]
INIT arf_dict["input_dtype"]
INIT arf_dict["objective"]
INIT arf_dict["tree_output"]
INIT arf_dict["base_offset"]

FOR each hoeffding_tree
   ENQUEUE (root_node, 0) to queue

   WHILE queue is not empty
      DEQUEUE (node, node_index) from queue
      RESET flig flag to False
      INIT a dictionary named hoeffding_dict

      IF node is a child node
         Update hoeffding_dict

      ELSE IF node is a parent node

         IF node is a NumericBinaryBranch class instance
            split_threshold = node.threshold

         ELSE IF node is a NominalBinaryBranch class instance
            split_threshold = node.value

            IF split_threshold is 1
               SET flip flag to True
```

```
        split_threshold = 0.5

    node_index = node_index + 1
    left_child_index = node_index
    IF flip
       ENQUEUE (right_child_node, left_child_index) to queue
    ELSE
       ENQUEUE (left_child_node, left_child_index) to queue

    node_index = node_index + 1
    right_child_index = node_index
    IF flip
       ENQUEUE (left_child_node, right_child_index) to queue
    ELSE
       ENQUEUE (right_child_node, right_child_index) to queue

    Update hoeffding_dict
```

3.6.1 Design Considerations

First, it was important to validate how to correctly perform the manual extraction of the split feature and split value from the River models. In River API, the "NumericBinaryBranch" class instance were used if the split feature was numerical and the "NominalBinaryBranch" were used if the split feature was categorical. Based on the official documentation, the split condition of "NumericBinaryBranch" was shown in the image below. The split condition was exactly the same as the split condition used by the Scikit-learn tree models.

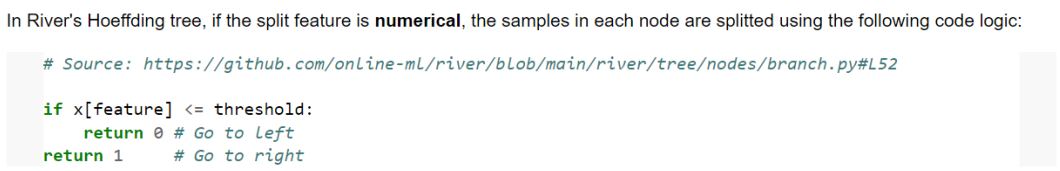


Figure 3.6.1.1: The split condition of River tree models for numerical features

While the split condition of "NominalBinaryBranch" was shown in the image below:

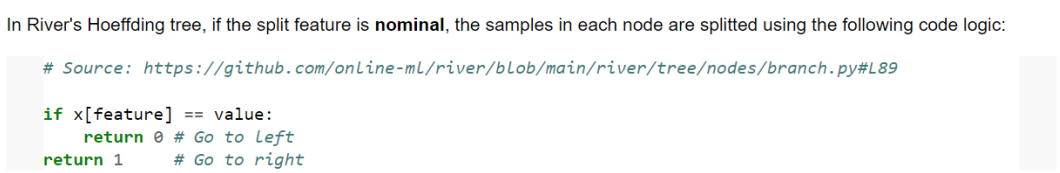


Figure 3.6.1.2: The split condition of River tree models for categorical features

It was discovered that the SHAP library used the split condition that was the same as the one in the “NumericBinaryBranch” regardless of whether the feature was numerical or categorical. Hence, some modifications must be made to ensure that the extraction of the tree weights was done correctly given the difference in the split conditions between SHAP library and River library.

To solve this, the conversion logic was implemented as shown below. It was assumed that all the nominal features were one-hot encoded when training the adaptive random forest regressor and classifier, hence the value could only be either 0 or 1. If the split value was 0, then the split value was converted to 0.5. Else if the split value was 1, then the split value was converted to 0.5. Since the split direction was different, the left and right child of the current node was flipped. In other words, when the split value was 1, flipping the left and right child of the current node would have the same effect as if the split value was 0. Thus, the split value was also set to 0.5.

Model type	Split Value	Split direction if feature value is 0	Conversion
Scikit-learn random tree	0.5	Go to left	-
Adaptive random tree	0	Go to left	Change the split value to 0.5
	1	Go to right	Change the split value to 0.5. Flip the left and right child of the current node.

Figure 3.6.1.3: The split value conversion logic to ensure the correctness of the tree weight extraction process

There were other technical details required to be considered during the weight extraction process. For example, the dictionary must contain the following key value pairs when extracting the tree weights from the adaptive random forest classifier to the dictionary. The author passed in the Scikit-learn random forest models into the tree SHAP explainer to examine how the dictionary should be extracted. Put it simply, it was just an API discovery process, and the details were not discussed here for brevity. The discovery process was well-documented and could be found in “jupyter_notebook/ FYP2_ARF_to_Dict_Validation.ipynb”.

```
Class name that stores tree weights: <class 'shap.explainers._tree.TreeEnsemble'>
internal_dtype : <class 'numpy.float64'>
input_dtype    : <class 'numpy.float32'>
objective      : binary_crossentropy
tree_output    : probability
base_offset    : [0. 0.]
```

Figure 3.6.1.4: The mandatory key value pairs in the dictionary for the tree weight extraction process to be successful

3.7 Tree SHAP

Tree SHAP could be used to review the individual model prediction, review the individual model loss, review the model's average prediction behaviour, evaluate the model performance, and monitor drift. This section would discuss the plots that were used to achieve these purposes. For brevity, the visualization and interpretation plots were only discussed on car price dataset but not lead scoring dataset. It was because both approaches and interpretations were similar. Please refer to "jupyter_notebook/FYP2_Lead_Scoring_Explainer.ipynb" for the discussion of the plots on lead scoring dataset.

3.7.1 Setup

There were two approaches to build a tree SHAP explainer which were tree path dependent approach and interventional approach. The explainer using interventional approach was much slower than tree path dependent approach as the scaled linearly with the size of the provided background dataset. As a result, tree_path_dependent approach was always used except for reviewing individual model loss, evaluating model performance and monitoring drift. However, based on the analysis and validation as mentioned in Section 5.4, only the tree SHAP explainer with the interventional approach was used.

In the SHAP library, to initialize a tree SHAP explainer using interventional approach to review the individual model prediction and review the model's average prediction behaviour, three conditions must be met as follows:

1. interventional approach must be used by setting feature_perturbation parameter to "interventional".
2. Background dataset must be provided when building the explainer. Scott Lundberg, who was the author of the SHAP library, did not recommend providing more than 1000 random background samples since it would be too slow if too many samples were used. The number of background samples could be as little as 100.
3. The truth outputs must be provided when calculating the SHAP values.

In the SHAP library, to initialize a tree SHAP loss explainer using interventional approach to review the individual model loss, evaluate the model performance, and monitor drift, the three aforementioned conditions above must be met and also an additional parameter named “model_output” must be set to “log_loss”.

The images below showed the initialization of the tree SHAP explainer and the tree SHAP loss explainer for explaining car price dataset and lead scoring dataset, respectively. The validation of these explainers was discussed in the Section 5.4.

Build a tree SHAP explainer that uses `interventional` approach.

In [11]:
```
cp_tree_explainer = shap.TreeExplainer(model = cp_arf_dict,
                                       feature_perturbation = 'interventional',
                                       data = cp_X_test_truth_av_subsample)
cp_shap_values = cp_tree_explainer.shap_values(cp_X_test_truth_av)
```

Build a tree SHAP loss explainer that uses `interventional` approach and the `model_output` is set to `log_loss`.

In [12]:
```
cp_tree_loss_explainer = shap.TreeExplainer(model = cp_arf_dict,
                                            feature_perturbation = 'interventional',
                                            model_output = 'log_loss',
                                            data = cp_X_test_truth_av_subsample)

cp_shap_loss_values = cp_tree_loss_explainer.shap_values(cp_X_test_truth_av, cp_y_test_truth_av)
```

Figure 3.7.1.1: The initialization of tree SHAP explainer and tree SHAP loss explainer (Car price dataset)

Build a tree SHAP explainer that uses `interventional` approach.

In [11]:
```
ls_tree_explainer = shap.TreeExplainer(model = ls_arf_dict,
                                       feature_perturbation = 'interventional',
                                       data = ls_X_test_truth_av_subsample)

ls_shap_values = ls_tree_explainer.shap_values(ls_X_test_truth_av)
```

Build a tree SHAP loss explainer that uses `interventional` approach and the `model_output` is set to `log_loss`.

In [12]:
```
ls_tree_loss_explainer = shap.TreeExplainer(model = ls_arf_dict,
                                            feature_perturbation = 'interventional',
                                            model_output = 'log_loss',
                                            data = ls_X_test_truth_av_subsample)

ls_shap_loss_values = ls_tree_loss_explainer.shap_values(ls_X_test_truth_av, ls_y_test_truth_av)
```

Figure 3.7.1.2: The initialization of tree SHAP explainer and tree SHAP loss explainer (Lead scoring dataset)

3.7.2 Review Model's Average Prediction Behaviours

Beeswarm plots

Using tree SHAP explainer, the SHAP values of each sample were calculated and individually plotted as a single dot to form a dense data that is grouped by feature. To prevent data points plotting on top of one another, random jittering effect was applied to move the data points away from one another to prevent overlapping. As a result, the dense data points looked like a beeswarm instead of a straight line. Beeswarm plot could be used to visualize the feature importance and correlation for each individual sample.

The x-axis showed the SHAP values while the y-axis showed the features. The feature with the highest summed absolute SHAP values was displayed at the top, followed by the next highest summed absolute SHAP values in a top-down fashion. A scatterplot was drawn for each feature, where the feature value of each sample was plotted as a point. The feature value was represented by the colour ranging from blue to red. The more reddish the point's colour, the higher the feature value for that sample. The opposite case was true for the bluish colour. The continuous colour scale allowed the viewer to infer whether the feature was positively or negatively correlated with the prediction output.

The image below showed the beeswarm plot for the car price dataset. Below were some examples on how to interpret the beeswarm plot:

1. "Manufacture Year" was the most important feature in determining the price.
2. Features like "Manufacture Year", "Width (mm)", and "Peak Power Hp" were positively correlated with car price. It was because most red points were concentrated on the right side and most blue points were concentrated on the left side.
3. Feature like "Direct Injection_Multi-Point Injected", "Mileage", and "Fuel Type_Diesel" were negatively correlated with car price. It was because most red points were concentrated on the left side and most blue points were concentrated on the right side.
4. Higher "Manufacture Year" correlated with higher car price. The model predicted high car price when the value of the "Manufacture Year" was high.

5. The “Direct Injection_Multi-Point Injected” feature negatively correlated with the car price. The model predicted lower car price when the value of the category value of “Direct Injection” was “Multi-Point Injected”. Note that the categorical feature was formatted such that the categorical feature and its categorical value were separated with an underscore.

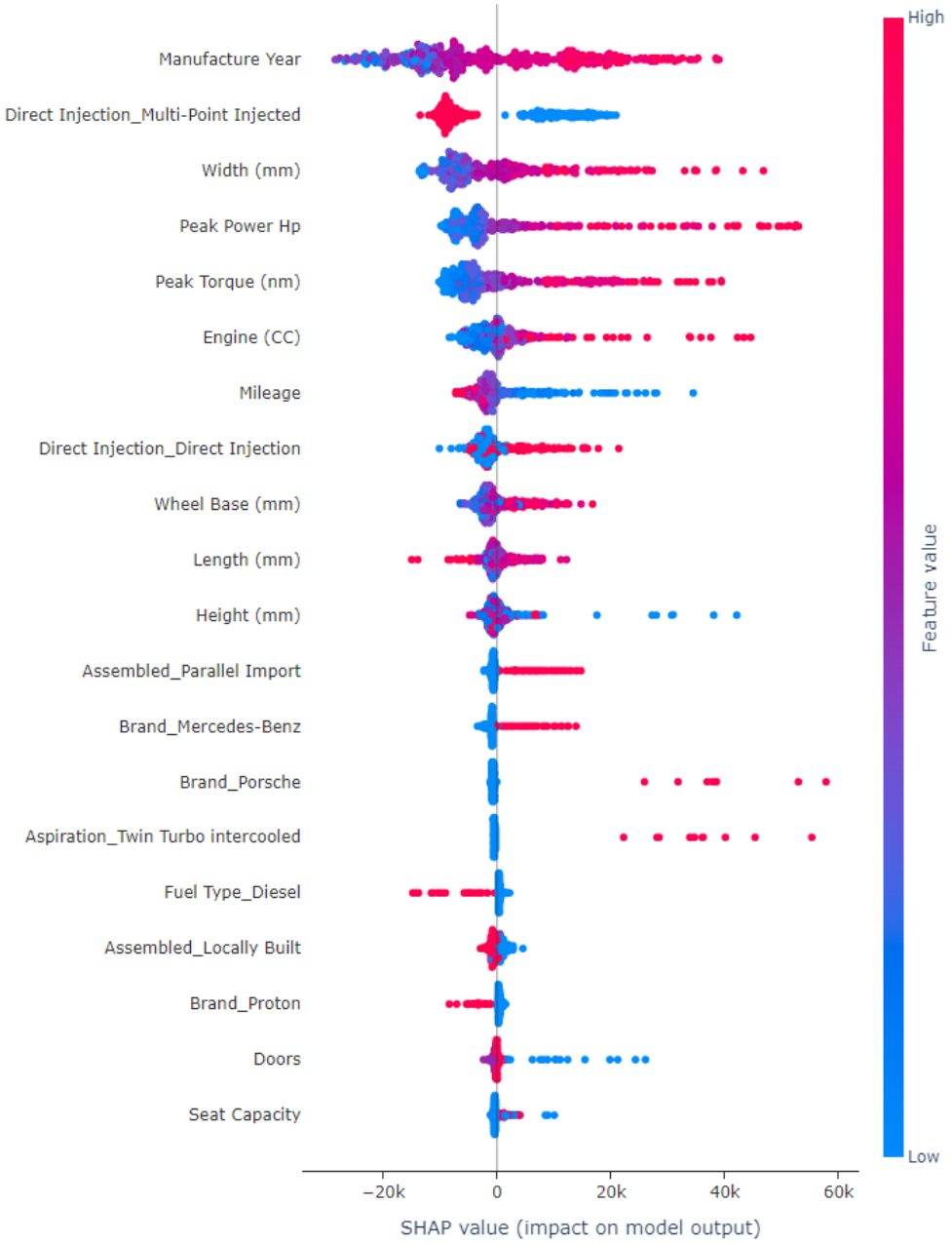


Figure 3.7.2.1: Beeswarm plot (Car price dataset)

Feature importance bar plots

Using tree SHAP explainer, the SHAP values of each sample were calculated and summed up by feature. Bar plot could be used to visualize the feature importance and correlation by calculating the sum of the absolute SHAP values for each feature.

The x-axis showed the SHAP values and the y-axis showed the X features. The feature with the highest summed absolute SHAP values was displayed at the top, followed by the next highest summed absolute SHAP values in a top-down fashion. The red bar represented positive correlation between the feature and the target while the blue bar represented the negative correlation between the feature and the target.

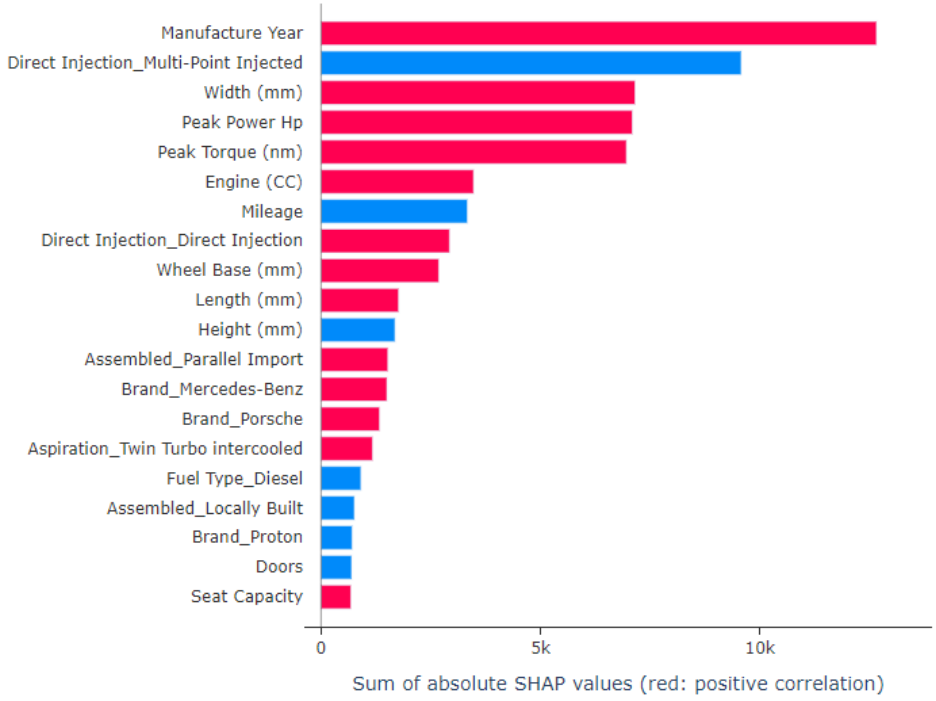


Figure 3.7.2.2: Feature importance bar plots (Car price dataset)

The image above showed the feature importance bar plot for the car price dataset. Below were some examples on how to interpret the plot:

Below were some examples on how to interpret the feature importance bar plot:

1. “Manufacture Year” was the most important feature in determining the price. It was because “Manufacture Year” had the longest bar length.
2. Features like “Manufacture Year”, “Width (mm)”, and “Peak Power Hp” were positively correlated with car price. It was because the bars were coloured in red.

3. Features like “Direct Injection_Multi-Point Injected”, “Mileage”, and “Fuel Type_Diesel” were negatively correlated with car price. It was because the bars were coloured in blue.

3.7.3 Evaluate Model’s Performances

Model loss bar plot

Using tree SHAP loss explainer, the SHAP loss value of each sample were calculated. In every sample, the features with positive SHAP loss values increased the prediction error while features with negative SHAP loss values decreased the prediction error. According to Chuangxin Lin, a bar plot could be constructed to visualize the summed positive/negative SHAP loss value across all features [17]. A positive SHAP loss bar plot could provide information on which features contributed the most model error, while a negative SHAP loss bar plot could provide information on which features helped the most in reducing the model error.

To achieve better model performance, the data scientists should figure out ways to reduce both the positive SHAP loss values and negative SHAP loss values for as many samples as possible.

Positive SHAP loss bar plot

Below was one of the ways to interpret the positive SHAP loss bar plot:

1. “Manufacture Year” contributed the most errors in the car price predictions. Based on the beeswarm plot and the feature importance bar plot, “Manufacture Year” was also contributing the most values in the predicted car prices. This might suggest that the model relied too heavily on this feature to predict the car price. Further model inspection was required by the data scientists to investigate the issue. The investigation was not covered since it was out of scope.

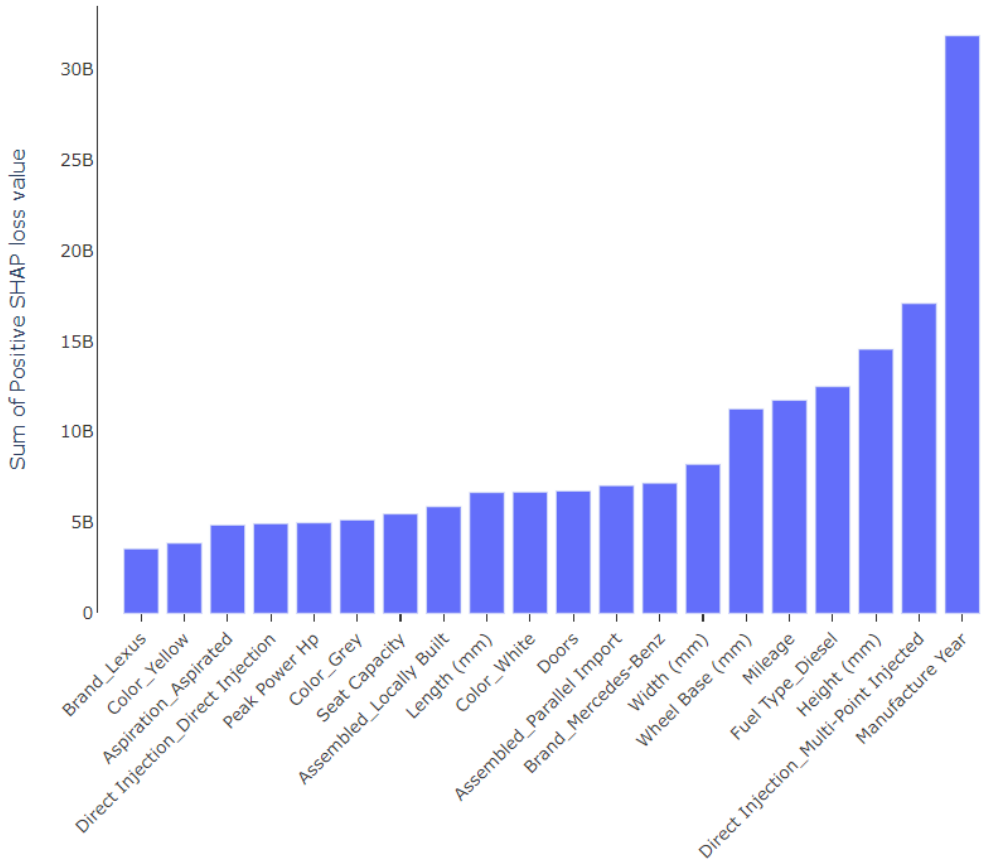


Figure 3.7.3.1: Positive SHAP loss bar plot (Car price dataset)

Negative SHAP loss bar plot

Below was one of the ways to interpret the negative SHAP loss bar plot:

1. "Peak Power Hp" and "Manufacture Year" helped the most in reducing the model error.
2. Combining the insights from the positive SHAP loss bar plot, "Manufacture Year" was relevant in predicting car prices for some samples but not the others. Further model inspection was required by the data scientists to investigate why "Manufacture Year" was irrelevant in predicting those samples. The investigation as not covered since it was out of scope.

Note that the result was not contradictory. It was because the positive SHAP loss bar plot only summed up positive SHAP values for each feature in each sample, while negative SHAP loss bar plot only summed up negative SHAP values.

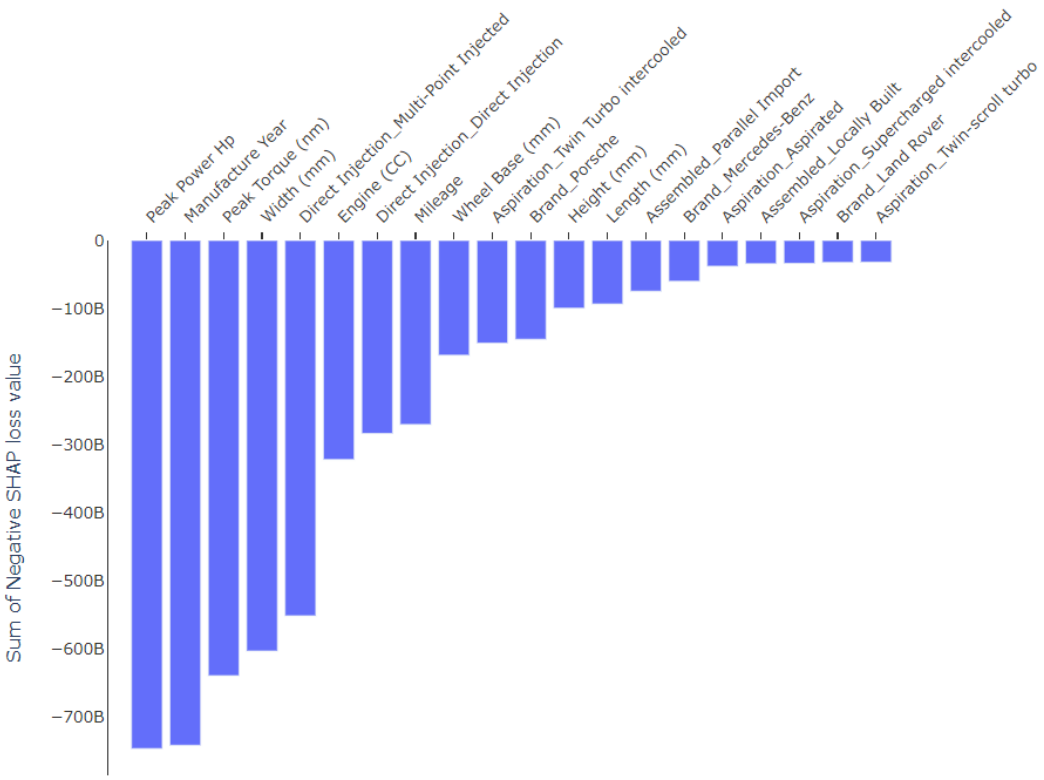


Figure 3.7.3.2: Negative SHAP loss bar plot (Car price dataset)

3.7.4 Review Individual Model Predictions and Individual Model Losses

There were various plot types like waterfall plot, force plot, bar plot, decision plot and more that were useful to provide explanation at a sample level. In comparisons with other plot types, bar plot was chosen since the plot could present the local explanation very clearly to the audience when the number of features displayed was high.

The bar plot could visualize both SHAP values and SHAP loss values. In web application, only SHAP values were visualized when the inventory managers and sales employees requested for explanation on the individual model predictions. Both SHAP values and SHAP loss values were visualized when the data scientists wanted to review individual model losses by analysing which features contributed to the prediction error and which features helped reducing the prediction error.

In the bar plots, the feature with the highest absolute SHAP values was displayed at the top, followed by the next highest absolute SHAP values in a top-down fashion. In both plots, the red bar represented positive SHAP values while the

blue bar represented negative SHAP values. However, the red and blue bar presented different meaning in both plots, respectively. For bar plot visualizing SHAP values, the red bar meant the positive contribution of the feature to the difference between the current predicted price and expected price, while the blue bar meant the negative contribution. For bar plot visualizing SHAP loss values, the red bar meant that the feature contributed to the prediction error while the blue bar meant that the feature helped reducing the prediction error.

The following two plots provided examples on how to interpret the SHAP bar plot and SHAP loss bar plot.

This plot below showed an accurate prediction. The data scientists could derive more useful insights when comparing both bar plots side by side. For this sample, “direct_injection_Multi-Point Injected” was the feature that contributed the most in the car price prediction. Among the top ten features with the highest absolute SHAP loss value, none of these features contributed to the errors in the car price prediction.

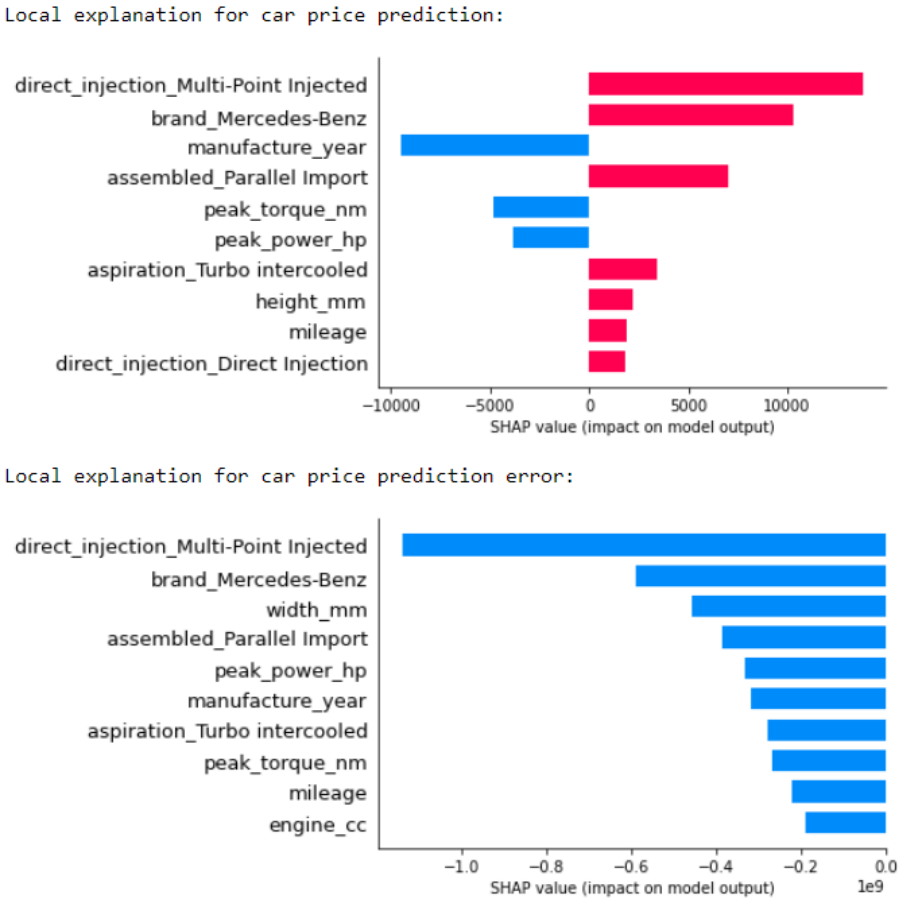


Figure 3.7.4.1: SHAP bar plot and SHAP loss bar plot showing accurate prediction (Car price dataset)

This plot below showed an inaccurate prediction. For this sample, "aspiration_Twin Turbo intercooled" and "peak_power_hp" were the top two features that contributed the most in the car price prediction. Out of the top ten features with the highest absolute SHAP loss value, the "manufacture_year", "mileage", and "height_mm" were the only features that were not contributing to the errors in the car price prediction. Since the current sample was the test sample, it was possible that the model was not sufficiently trained with samples that represented this sample. It was also possible that the sample was a drifted sample, errored sample, or a malicious sample. Further investigation was required but the investigation was not covered since it was out of scope.

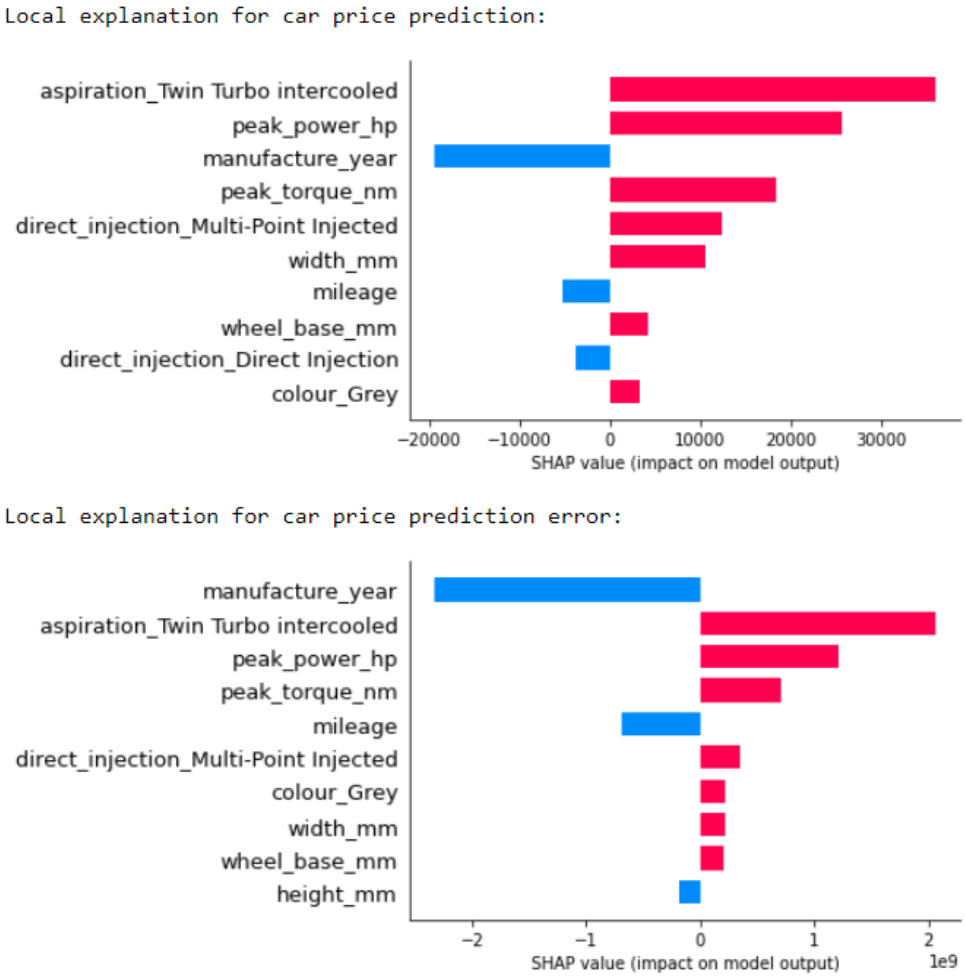


Figure 3.7.4.2: SHAP bar plot and SHAP loss bar plot showing inaccurate prediction (Car price dataset)

3.7.5 Monitor Drifts

This section was discussed in the next section to avoid repetition.

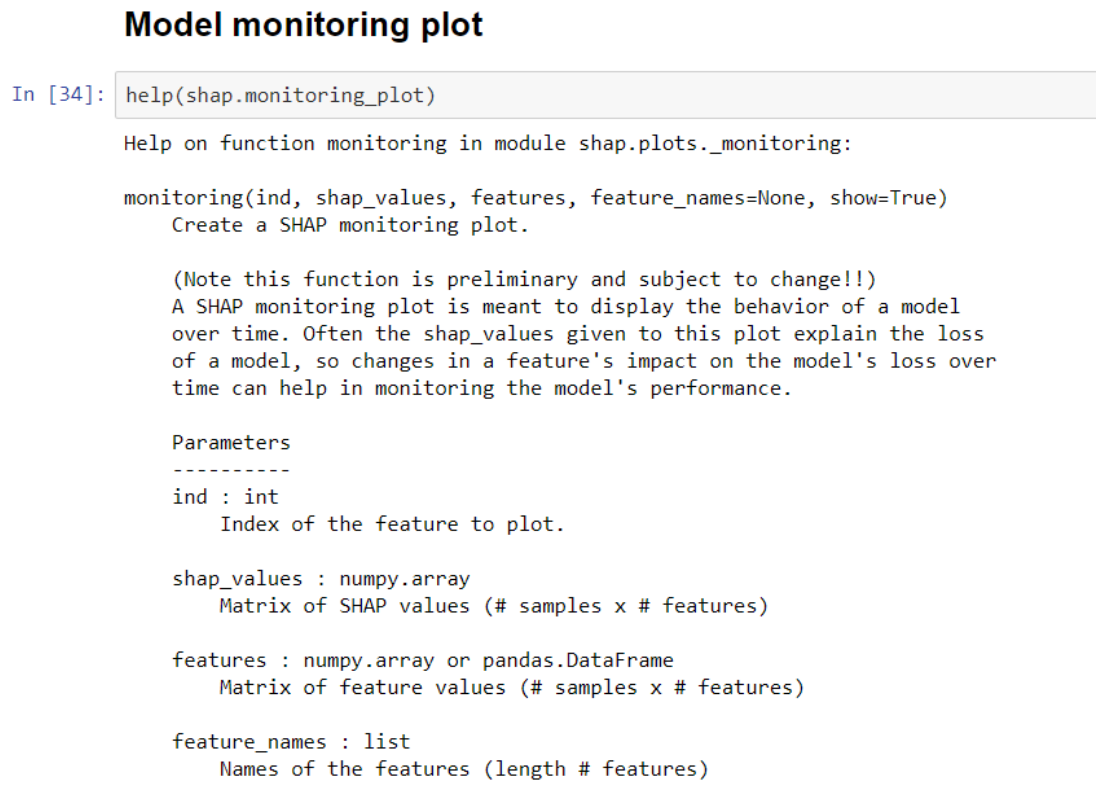
3.8 Monitoring Drift

There were two real-world scenarios when monitoring drifts. The first scenario was that the test set contained the truth while the second scenario was that the test set did not contain the truth. For the first scenario, the drift could be monitored by visualizing the calculation of SHAP loss values for each feature. For the second scenario, the drift could be monitored by performing statistical tests for each feature. Two different samples were required to monitor the drift in both scenarios. The first set of samples were the validation set that represented the expected distribution. The validation set normally consisted of the offline data used in model training or older online data. The second samples were the test set that represented recent online data that might suffer from drift or data error. In this project, the validation set was fixed at using model training dataset due to limited samples.

The following sections discussed the techniques and the ways to visualize the drift using the SHAP loss monitoring and statistical tests. The SHAP loss monitoring function was validated in the Section 5.5.1, while the calculations of the PSI and the chi-squared goodness of fit tests were validated in the Section 5.5.2.

3.8.1 SHAP Loss Monitoring

The image below showed that the SHAP monitoring plot, that was implemented by Scott Lundberg, was still in development and subjected to change as of April 2022. The reviewed SHAP package version is 0.40.0. Hence, the author had conducted an experiment to test the effectiveness of Lundberg's model monitoring plot against drift or data error, as mentioned in Section 5.5.1. The experiment results showed that the preliminary function was not effective in raising alarms on gradual drift and data error. Hence, the author added more statistical tests into the algorithm to reduce the likelihood of having false negative alarms, even at the cost of higher likelihood of having false positive alarms. The proposed function was also validated as mentioned in Section 5.5.1.



Model monitoring plot

```
In [34]: help(shap.monitoring_plot)

Help on function monitoring in module shap.plots._monitoring:

monitoring(ind, shap_values, features, feature_names=None, show=True)
    Create a SHAP monitoring plot.

    (Note this function is preliminary and subject to change!!)
    A SHAP monitoring plot is meant to display the behavior of a model
    over time. Often the shap_values given to this plot explain the loss
    of a model, so changes in a feature's impact on the model's loss over
    time can help in monitoring the model's performance.

    Parameters
    ----------
    ind : int
        Index of the feature to plot.

    shap_values : numpy.array
        Matrix of SHAP values (# samples x # features)

    features : numpy.array or pandas.DataFrame
        Matrix of feature values (# samples x # features)

    feature_names : list
        Names of the features (length # features)
```

Figure 3.8.1.1: The documentation of the Scott Lundberg's SHAP loss monitoring function in SHAP library 0.40.0

The pseudocode of the Lundberg's function and other technical details could be found in jupyter_notebooks/FYP2_Car_Price_Explainer.ipynb. Below showed the pseudocode of the proposed SHAP loss monitoring function.

```
LET shap_values_list[0] be shap_values for validation set
LET shap_values_list[1] be shap_values for evaluation set
LET features_list[0]    be features    for validation set
LET features_list[1]    be features    for evaluation set

LET start_index at the last sample of validation set
Concatenate the shap_values for both sets into a single list
Split the list into two partitions

WHILE number of samples at the next partition >= 1:
   Calculate the independent two-sample test and its p-value on SHAP loss mean
s difference on both partitions
   Update the smallest p-value that has seen so far
   Extend the first partition by 50 samples and shrink the next partition by 50 sa
mples

IF the smallest p-value <= 5%:
   Raise the alarm

ELSE:
   LET start_index at the last sample of validation set
   Concatenate the features for both sets into a single list
   Split the list into two partitions

   WHILE number of samples at the next partition >= 1:
      IF the feature_name is numerical:
         Calculate the Welch's t-test and its p-value on feature means difference o
n both partitions

      ELSE IF the feature_name is categorical:
         Calculate the chi-square goodness of fit test and its p-value on frequency
counts on both partitions

      Update the smallest p-value that has seen so far
      Extend the first partition by 50 samples and shrink the next partition by 50 s
amples

   IF the smallest p-value <= 5%:
      Raise the alarm

Plot the graph
```

There were few differences between the proposed function and the Lundberg's function. First, the Lundberg's function expected a matrix of SHAP values and a matrix of features. As a result, the user was forced to concatenate the values from both (validation and evaluation) sets into one single array. In addition, there was no function's parameter that inform the starting index of the monitoring process. As a

result, the validation set was also partitioned and evaluated. Thus, the proposed function was designed to expect a list of two matrices of SHAP values and also a list of two matrices of features. The first matrix in the array represented the validation set while the second one represented the evaluation set. The partitioning and monitoring would start from the validation set.

Second, the Lundberg's function only used features to extract the feature names and converted the values into an array of colours for plotting the points. On the other hand, the proposed function further utilized the feature values to reduce type II error, even at the cost of having higher type I error. It was because the cost of type II error was higher than type I error in model drift monitoring. For categorical features, the chi-square goodness of fit test was calculated to check if the frequency counts in the evaluation set matched with those in the validation set. For numerical features, the Welch's t-test was calculated to check if the means difference in both sets were statistically significant.

The plot below showed an example of the SHAP loss monitoring plot on "Manufacture Year". The presence of a vertical line that separated the two partitions indicated that there was statistically significant mean difference between those partitions.

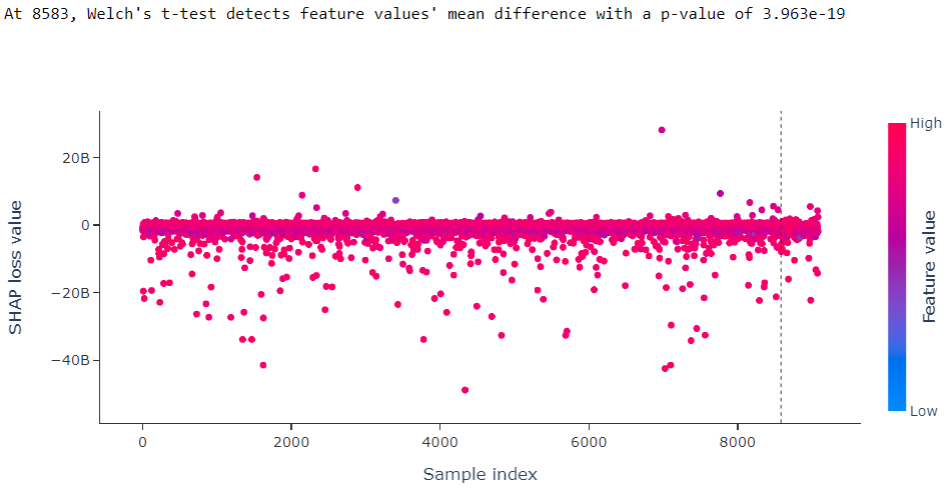


Figure 3.8.1.2: An example of SHAP loss monitoring plot on "Manufacture Year" (Car price dataset)

3.8.2 Statistical test

Statistical tests were conducted to compare the distribution of the features between the validation set and the test set. Population Stability Index (PSI) was used to compare distribution between numerical features and categorical features that had 10 or more categories. Chi-Square Goodness of Fit Test was used to compare distribution between categorical features that have less than 10 categories. This section described the methods used to visualize the drift monitoring result using these statistical tests.

Population Stability Index (PSI)

The table below visualized the calculation of the PSI. For each feature, the data scientists could compare the relative frequencies between the (expected) train set and the (observed) application data in each bin. The data scientists could also observe the PSI value for each bin to identify which bin had drifted the most. For example, based on the table below, the values in the bin [1.56, 2.63] experienced the most drift with a PSI value of 10%. Finally, the formatted PSI table could be exported as a HTML table to be used in the web application using Pandas API.

Bin	Expected freq.	Observed freq.	Expected relative freq. (%)	Observed relative freq. (%)	Diff. in relative freqs. (%)	PSI value (%)
[0.5, 1.56)	579	97	14.23	15.04	-0.81	0.05
[1.56, 2.63)	1477	126	36.29	19.53	16.76	10.38
[2.63, 3.69)	829	130	20.37	20.16	0.21	0.00
[3.69, 4.75)	564	130	13.86	20.16	-6.30	2.36
[4.75, 5.82)	334	75	8.21	11.63	-3.42	1.19
[5.82, 6.88)	156	41	3.83	6.36	-2.52	1.28
[6.88, 7.95)	79	27	1.94	4.19	-2.25	1.73
[7.95, 9.01]	52	19	1.28	2.95	-1.67	1.39
Total	4070	645	100.00	100.00	0.00	18.37

Figure 3.8.2.1: The visualization of PSI of "avg_page_view_per_visit" in tabular format (Lead scoring dataset)

Besides, the calculated PSI values could also be visualized in grouped bar chart. The grouped bar chart enabled side-by-side comparison between expected and observed relative frequency for each bin.

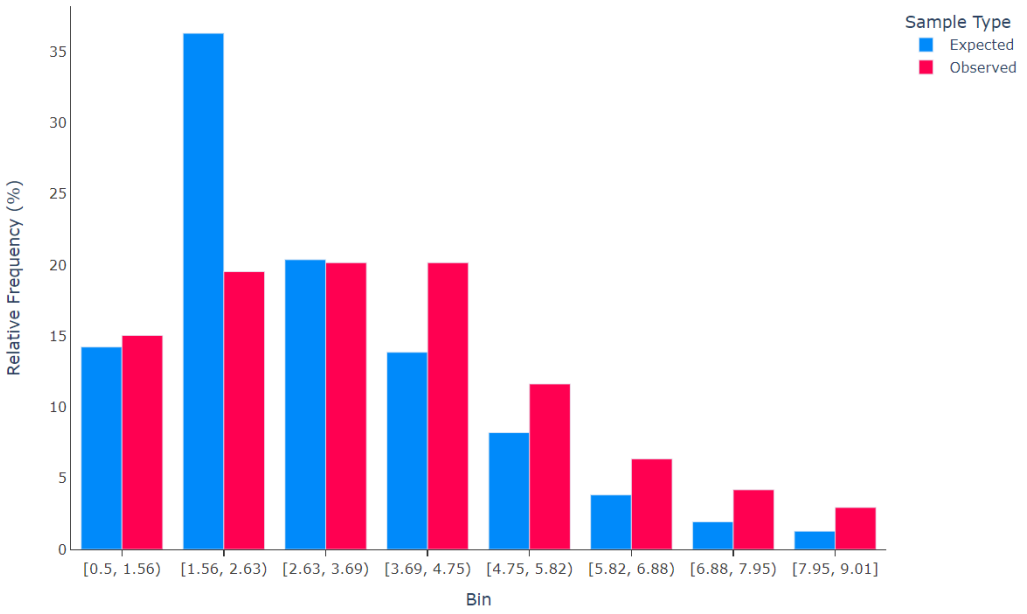


Figure 3.8.2.2: The visualization of PSI of "avg_page_view_per_visit" using grouped bar chart (Lead scoring dataset)

Chi-squared goodness of fit test

The table below visualized the calculation of the chi-squared values. For each feature, the data scientists could compare the expected count between the (expected) train set and the (observed) application data in each category.

```
The chi-square stats for "transmission_Manual" is 9.1312 with the p-value of 0.0025128521.
```

	Expected count	Expected count (Re-adjusted)	Observed count	Diff count	Squared Diff count	Chi2 value
0	7927.0	508.152936	490.0	18.152936	329.529073	0.648484
1	606.0	38.847064	57.0	-18.152936	329.529073	8.482728

Figure 3.8.2.3: The visualization of chi-squared goodness of fit test of "transmission_Manual" in tabular format (Car price dataset)

CHAPTER 4 SYSTEM IMPLEMENTATION

4.1 Dataset

Car price dataset and lead scoring dataset were used to implement the lead management module and the inventory management module, respectively.

4.1.1 Lead scoring dataset

The lead scoring dataset was used both as the application dataset and the dataset to train the lead scoring model. The lead scoring dataset used was obtained from Amrita Chatterjee's published Kaggle dataset [18]. Most features were discarded from the original dataset. It was because the theme of the lead scoring dataset was not car dealership. Instead, the lead scoring dataset, in which the theme was online education, was manually inspected and modified to change the theme to car dealership.

The descriptions of each lead information were shown below. These columns were kept since any lead scoring dataset would have these common lead attributes. In the application dataset, additional columns which were name, email address, and phone number were added to personally identify a lead. In the model training, the target attribute was "Converted" while all the lead attributes below except "Do not call" were used as the features. It was because "Do not call" did not have enough positive samples. For further details on the dataset, please refer to "FYP2_Lead_Scoring_Model_Training" notebook file.

No	Feature Name	Description and Modification	Data type
1	Do Not Email	An indicator variable selected by the customer wherein they select whether or not they want to be emailed about the course or not.	Nominal
2	Do Not Call	An indicator variable selected by the customer wherein they select whether or not they want to be called about the course or not.	Nominal
3	Converted	The target variable. Indicates whether a lead has been successfully converted or not.	Nominal

Table 4.1.1.1: Lead scoring dataset description (Part I)

No	Feature Name	Description and Modification	Data type
4	TotalVisits	The total number of visits made by the customer on the website.	Numerical
5	Total Time Spent on Website	The total time spent by the customer on the website.	Numerical
6	Page Views Per Visit	Average number of pages on the website viewed during the visits.	Numerical
7	Occupation	Indicates whether the customer is a student, unemployed or employed.	Nominal
8	A free copy of “Mastering the Interview”.	Change the name to "A free copy of car specification".	Nominal

Table 4.1.1.2: Lead scoring dataset description (Part II)

4.1.2 Car price dataset

The car price dataset was used both as the application dataset and the dataset to train the car price model. The description of the car specifications was shown below. The used car price data was scraped from https://www.carlist.my in 2021. After data cleaning and pre-processing, the dataset consisted of 8,534 train samples and 1,095 test samples. The dataset was split and stratified in advance to ensure that every category exist in both train and test set. In the model training, the price was the target variable while the rest of the car specifications except "model" were used as the features. The "model" was not used since it consisted of more than 300 categories, which slowed down the transfer learning process, model training process, and the SHAP and the SHAP loss explaining process. For further details on the dataset, please refer to "FYP2_Car_Price_Model_Training" notebook file.

No	Feature Name	Description	Data type
1	manufacture_year	Manufacturing year of the car	Numeric
2	mileage	The distance travelled by the car (in kilometer).	Numeric
3	price	The price of the car (in Ringgit Malaysia).	Numeric
4	length_mm	The length of the car in millimeter.	Numeric
5	engine_cc	The engine capacity of the car.	Numeric
6	aspiration	Type of aspiration in the car engine. Possible values are Aspirated, Turbo intercooled, Twin-scroll turbo, Turbocharged, Turbo supercharged intercooled, Twin Turbo intercooled, Supercharged intercooled, Supercharged.	Categorical
7	wheel_base_mm	The wheel base of the car in millimeter.	Numeric
8	width_mm	The width of the car in millimeter.	Numeric
9	direct_injection	The direct injection in the car engine. Possible values are Multi-Point Injected, Direct Injection, Carburettor Single, Direct/Multi-point injection.	Categorical

Table 4.1.2.1: Car price dataset description (part I)

10	seat_capacity	The number of available seats in the car.	Numeric
11	peak_power_hp	The horsepower/engine power of the car.	Numeric
12	fuel_type	The fuel type of the vehicle. Possible values are Petrol - Unleaded (ULP), Diesel, Hybrid, Petrol - Leaded.	Categorical
13	steering_type	The steering type of the car. Possible values are Rack and Pinion, Electronic Power Steering, Recirculating Ball, Hydraulic Power, Worm, and Roller.	Categorical
14	assembled	The location where the car is assembled. Possible values are Locally Built, Official Import, Parallel Import.	Categorical
15	height_mm	The height of the car in millimeter.	Numeric
16	peak_torque_nm	The peak torque of the car engine in nanometer.	Numeric
17	doors	The number of doors that the car has.	Numeric
18	brand	The car brand. Possible values are Toyota, Honda, Proton, Perodua, Nissan, BMW, Mercedes-Benz, Mazda, Peugeot, Ford, Volkswagen, Mitsubishi, Kia, Volvo, Hyundai, MINI, Porsche, Subaru, Lexus, Land Rover, Renault, Jaguar.	Categorical
19	colour	The main colour of the car. Possible values are White, Black, Silver, Grey, Blue, Red, Gold, Brown, -, Bronze, Purple, Orange, Green, Maroon, Yellow, Beige, Pink, Magenta.	Categorical
20	model	The model of the car. There are 360 possible values such as Alza, Myvi, Triton, Almera, CX-5, Ranger, XV, Iriz, HR-V, CR-Z. Run car_train['model'].value_counts() to see all the values.	Categorical
21	transmission	Either the car is Automatic or Manual.	Categorical

Table 4.1.2.2: Car price dataset description (part II)

4.2 Jupyter Notebook Artifacts

Since the proposed online AI learning and monitoring system were novel, Jupyter notebooks were created to present the technical details that otherwise would be impossible to document inside the web service's source itself. Jupyter notebooks were created to document the experimentation, validation, and testing of the system. Then, the full-proof source code was transferred to the web service and deployed in Docker containers.

4.2.1 Setup

There was an easy way and hard way to open the IPYNB files. For the easy was, the viewers could directly open and view the notebooks that were converted to HTML without any setup. If the viewers would like to execute the IPYNB files, following the steps.

1. First, the machine must be installed with Anaconda or Miniconda. The author would use Miniconda in Windows operating system for the setup process.
2. Open the Anaconda or Miniconda prompt. Execute the two commands below to create two conda environments. One of the conda environment was used to work with River library and another conda environment was used to work with SHAP library. It was because as of April 2021, SHAP library and River library had conflicting dependencies on NumPy library. SHAP library requires NumPy version of 1.21.5 while River library requires NumPy version of 1.22.3. The import of both libraries will fail if the respective requirements are not met. Please use any name that the viewers saw fit.

>> conda env create -n arf_exp_env python=3.9

>> conda env create -n arf_exp_conda_env python=3.9

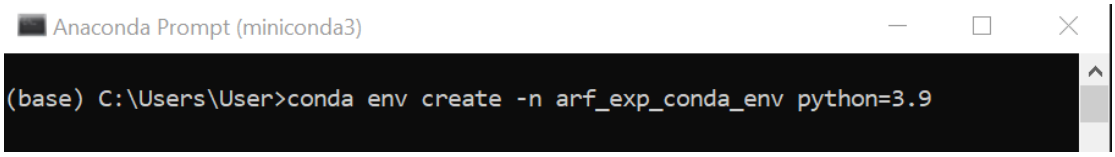


Figure 4.2.1.1: The command that created the conda environment

3. Execute the commands below to activate the specified conda environment. First, activate the arf_exp_env conda environment.

>> conda activate arf_exp_env

>> conda activate arf_exp_conda_env

4. Execute the commands below to install the required Python dependencies. The requirement text file could be found in jupyter_notebook folder. After that, execute the next command to check that there were no conflicting Python dependencies. This command was handy to check future dependencies issues as more third-party libraries were installed.

>> pip install pip install -r river_requirements.txt

>> pipdeptree

5. Perform the same step for the arf_exp_conda_env environment. Use shap_requirements.txt to install the Python dependencies instead. Execute the command below to exit the current conda environment.

>> conda deactivate

6. As documented in the next section, the notebooks required either one of the conda environment to execute. The setup could stop here if the viewers were okay with manually switching from one conda environment to another to execute the notebooks. However, if the views would like to have convenient switching of conda environments within the Jupyter notebook itself, please proceed with the following setup.

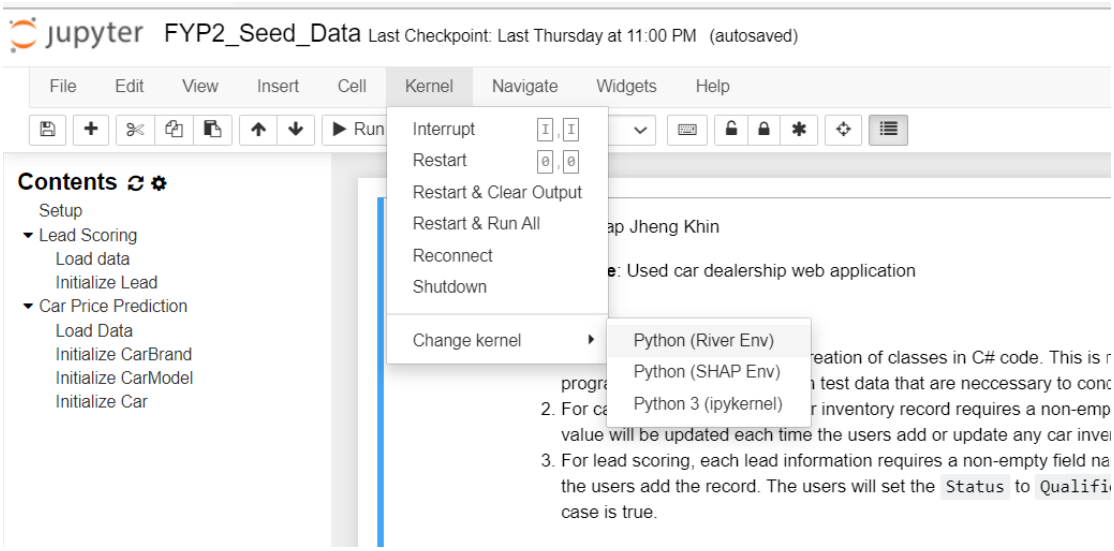


Figure 4.2.1.2: The UI showing the convenient switching of conda environments within the Jupyter notebook

7. First, activate the arf_exp_env conda environment. Execute the command below to create the UI option to switch to River environment within the Jupyter notebook itself.

>> python -m ipykernel install --user --name arf_conda_env --display-name "Python (River Env)"

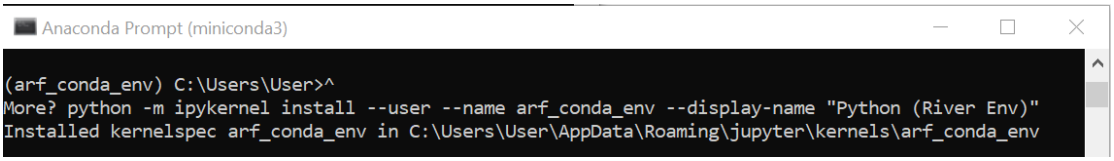


Figure 4.2.1.3: The command that registered the kernel into the Jupyter notebook

8. Deactivate the current environment and activate the arf_conda_exp_env conda environment. Execute the command below to create the UI option to switch to SHAP environment within the Jupyter notebook itself.

>> python -m ipykernel install --user --name arf_conda_exp_env --display-name "Python (SHAP Env)"

9. Run the command below to ensure that the command output looked like the one shown below. The setup process had been completed.

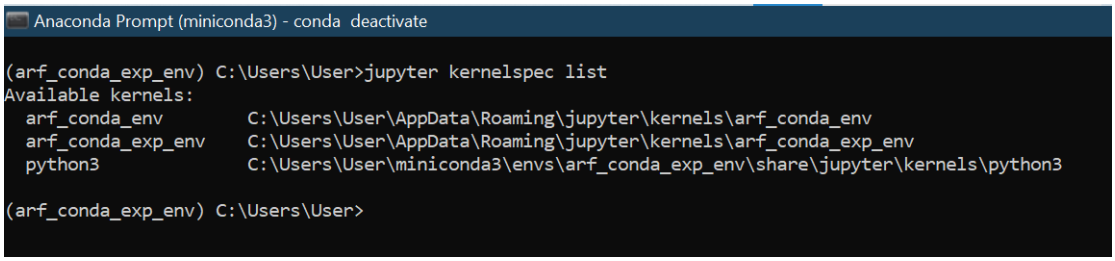


Figure 4.2.1.4: The command that validated the setup of environment switching

4.2.2 Execution Sequence

The table below showed the execution sequence of the Jupyter notebook files. Please follow the execution sequence if the viewers would like to execute all the notebook files from scratch.

The notebook should be executed in the sequence shown below.

1. Execute the files in the order specified below to run the experiment that validated the algorithm that transferred training weights from Scikit-learn random forest classifier to River adaptive random forest classifier.
   a) FYP2_ARF_CF_Transfer_Learning.ipynb
   b) FYP2_ARF_CF_Performance_Analysis.ipynb

2. Execute the files in the order specified below to run the experiment that validated the algorithm that transferred training weights from Scikit-learn random forest regressor to River adaptive random forest regressor.
   a) FYP2_ARF_RG_Transfer_Learning.ipynb
   b) FYP2_ARF_RG_Performance_Analysis.ipynb

3. The table below summarized the execution order for the rest of the IPYNB files.

IPYNB file	Prerequisite of IPYNB file
FYP2_ARF_to_Dict_Validation	FYP2_ARF_to_Dict_Conversion.ipynb
FYP2_Car_Price_Explainer	FYP2_Lead_Scoring_Model_Training FYP2_ARF_to_Dict_Conversion FYP2_Seed_Data
FYP2_Lead_Scoring_Explainer	FYP2_Car_Price_Model_Training FYP2_ARF_to_Dict_Conversion FYP2_Seed_Data
FYP2_Explainer_Validation	Execute all the files as specified in the second and third rows.
FYP2_Model_Monitoring_without_Truth	

Table 4.2.2.1: The execution sequence of the remaining IPYNB files

4. To avoid complications, just execute all the IPYNB files as shown below:
    a) FYP2_ARF_CF_Transfer_Learning.ipynb
    b) FYP2_ARF_CF_Performance_Analysis.ipynb
    c) FYP2_ARF_RG_Transfer_Learning.ipynb
    d) FYP2_ARF_RG_Performance_Analysis.ipynb
    e) FYP2_Lead_Scoring_Model_Training.ipynb
    f) FYP2_Car_Price_Model_Training.ipynb
    g) FYP2_ARF_to_Dict_Conversion.ipynb
    h) FYP2_ARF_to_Dict_Validation.ipynb
    i) FYP2_Seed_Data.ipynb
    j) FYP2_Explainer_Validation.ipynb
    k) FYP2_Lead_Scoring_Explainer.ipynb
    l) FYP2_Car_Price_Explainer.ipynb
    m) FYP2_Model_Monitoring_without_Truth.ipynb

4.2.3 Description of IPYNB files

Below was the description of each IPYNB files. Only the important documentation and the source code were extracted and discussed in the relevant report's sections.

1. FYP2_ARF_CF_Transfer_Learning.ipynb
    a) Purpose: A transfer learning algorithm was proposed to improve the offline training performance of the adaptive random forest classifier by transferring tree weights from fitted Scikit-learn traditional random forest classifier to River adaptive random forest classifier. The notebook described the algorithm pseudocode, improvements, limitations, and potential improvements. The transfer learning algorithm was also validated to ensure its correctness.
    b) Conda Environment: arf_conda_env (River environment)
    c) Execution Time: 20 minutes

2. FYP2_ARF_CF_Performance_Analyswas.ipynb
    a) Purpose: An experiment was conducted to verify that:
        - The pre-trained River adaptive random forest classifier's classification performance was at least good or better than River adaptive random

forest classifier that was trained from scratch in offline settings and online settings.

- The performance of the adaptive random forest classifier was at least good or better than traditional random forest classifier in offline settings.
- The performance of the adaptive random forest classifier was at least good or better than traditional random forest classifier under the influence of data drift or concept drift.
- Offline settings were scenarios where the dataset was full available for training and inference. The online settings were scenarios where the dataset was incrementally available over time and the drift occurrence was possible.

b) Conda Environment: arf_conda_env (River environment)

c) Execution Time: 25 minutes

3. FYP2_ARF_RG_Transfer_Learning.ipynb
   a) Purpose: A transfer learning algorithm was proposed to improve the offline training performance of the adaptive random forest regressor by transferring tree weights from fitted Scikit-learn traditional random forest regressor to River adaptive random forest regressor. The notebook described the algorithm pseudocode, improvements, limitations, and potential improvements. The transfer learning algorithm was also validated to ensure its correctness.
   b) Conda Environment: arf_conda_env (River environment)
   c) Execution Time: 3 minutes

4. FYP2_ARF_RG_Performance_Analyswas.ipynb
   a) Purpose: An experiment was conducted to verify that:
      - The pre-trained River adaptive random forest regressor's performance was at least good or better than River adaptive random forest regressor that was trained from scratch in offline settings and online settings.
      - The performance of the adaptive random forest regressor was at least good or better than traditional random forest regressor in offline settings.

- The performance of the adaptive random forest regressor was at least good or better than traditional random forest regressor under the influence of data drift or concept drift.
- Offline settings were scenarios where the dataset was full available for training and inference. The online settings were scenarios where the dataset was incrementally available over time and the drift occurrence was possible.

b) Conda Environment: arf_conda_env (River environment)

c) Execution Time: 10 minutes

5. FYP2_Lead_Scoring_Model_Training.ipynb
   a) Purpose: The web application required a machine learning model to assign a lead score. The first part described the data cleaning and processing process on the lead scoring dataset. The second part trained the models and evaluated their performance on the dataset. Even though all the models' performance were highly similar, the pre-trained adaptive random forest classifier was chosen instead of the traditional random forest classifier or the adaptive random forest classifier that was trained from scratch. It was because pre-trained adaptive random forest classifier yielded more stable and accurate predictions as compared to the adaptive random forest classifier that was trained from scratch. Traditional random classifier was not chosen since it did not support incremental learning.
   b) Conda Environment: arf_conda_env (River environment)
   c) Execution Time: 5 minutes

6. FYP2_Car_Price_Model_Training.ipynb
   a) Purpose: The web application required a machine learning model to predict car price. The first part described the data cleaning and processing process on the car price dataset that was scrapped from Carlist.my website. The second part trained the models and evaluated their performance on the dataset. Even though all the models' performance were highly similar, the pre-trained adaptive random forest regressor was chosen instead of the traditional random forest regressor or the adaptive random forest regressor that was trained from

scratch. It was because pre-trained adaptive random forest regressor required less memory as compared to the adaptive random forest regressor that was trained from scratch. Traditional random regressor was not chosen since it did not support incremental learning.

b) Conda Environment: arf_conda_env (River environment)

c) Execution Time: 5 minutes

7. FYP2_Seed_Data.ipynb

a) Purpose: The lead scoring dataset and car price dataset that were used in model evaluation were converted to records. The records were inserted to the application database through Entity Framework Core migration in the C# web application.

- To test the lead scoring prediction, the lead status of half of the test samples were set to "Active" and the lead status of another half samples were set to "Qualified" or "Disqualified". The value of the lead status was "Qualified" if the samples' target class was positive, while the value of the lead status was "Disqualified" if the samples' target class was negative. In the web application, the records with the active lead status were considered to have no truth outputs while the records with the qualified or disqualified lead status were considered to have truth outputs. In other words, the deployed lead scoring model would not train on lead records with the active lead status.
- To test the car price prediction, half of the test samples were set to "enable car price analytics" while another half samples were set to "disable car price analytics". In the web application, the deployed car price model would not train on the car inventory records with value of update analytics that was set to 0.

b) Conda Environment: arf_conda_env (River environment)

c) Execution Time: 1 minute

8. FYP2_ARF_to_Dict_Conversion.ipynb

a) Purpose: The River adaptive random forest regressor and classifier could not be directly ingested into SHAP tree explainer since it was not supported. Instead, SHAP tree explainer supported the ingestion of Python dictionary

containing common tree weights which were split feature, split threshold, node's prediction value and more. Thus, the tree weights must be manually extracted from both the regressor and classifier to a Python dictionary, respectively. The notebook showed the proposed River tree weights extraction process and its pseudocode.

b) Conda Environment: arf_conda_env (River environment)
c) Execution Time: 2 minutes

9. FYP2_ARF_to_Dict_Validation.ipynb
    a) Purpose: The notebook checked the correctness of the tree weights extraction from the River adaptive random forest regressor and classifier. To do so, the SHAP tree explainer objects were initialized with the Scikit-learn random forest regressor and classifier. During the initialization, the SHAP tree explainer would internally extract the tree weights into a class instance called TreeEnsemble from the ingested Scikit-learn model object. After the initialization, all the attributes of TreeEnsemble instance were accessed and reviewed to inform the proposed River tree weight extraction process on how to correctly format the weights.
    b) Conda Environment: arf_conda_exp_env (SHAP environment)
    c) Execution Time: 10 minutes

10. FYP2_Explainer_Validation.ipynb
    a) Purpose: The Tree SHAP library implemented by SHAP library was not expected to explain the predictions and model losses for incremental tree learners like adaptive random forest regressor and adaptive random forest classifier, and vice versa. An experiment was conducted to investigate why the SHAP values calculated by Tree SHAP explainer using the weights extracted from River tree models were faulty. The experimental result was able to prove that the implementation logics of River and SHAP library crashed with one other, causing the SHAP values calculation to be inaccurate. To resolve this, the tree SHAP explainer using interventional approach was used instead of the tree path dependent approach.
    b) Conda Environment: arf_conda_exp_env (SHAP environment)
    c) Execution time: 5 minutes

11. FYP2_Car_Price_Explainer.ipynb
    a) Purpose: The SHAP tree explainer and SHAP tree loss explainer were initialized by using the dictionary with the tree weights extracted from adaptive random forest regressor. First, the explainers were validated to ensure the correctness of the River tree weights extraction process. Then, the notebook proposed the most efficient way to visualize global model explanation and local model explanation. The beeswarm plot and feature importance bar plot were used to visualize the SHAP values globally; the model loss bar plot and the model monitoring plot were used to visualize the SHAP loss values globally for model monitoring and debugging; the bar plot was used to visualized SHAP values locally. The examples on the interpretation of each plot were documented to inform the users and scientists. In addition, the improved version of model monitoring function was proposed to reduce the likelihood of false negative alarms. It is because false negative alarms were expensive in model monitoring and debugging. An experiment was conducted to compare the effectiveness of the proposed model monitoring function and the original model monitoring function in detecting feature drifts or data errors.
    b) Conda Environment: arf_conda_exp_env (SHAP environment)
    c) Execution Time: 10 minutes

12. FYP2_Lead_Scoring_Explainer.ipynb
    a) Purpose: The purpose of this notebook was the same with FYP2_Car_Price_Explainer.ipynb, but in the context of lead scoring dataset. The only difference was that no experiment was conducted to compare the effectiveness of the model monitoring functions.
    b) Conda Environment: arf_conda_exp_env (SHAP environment)
    c) Execution Time: 5 minutes

13. FYP2_Model_Monitoring_without_Truth.ipynb
    a) Purpose: The purpose of this notebook was to document the drift monitoring process on used car application data that did not contain truth. Either

Population Stability Index (PSI) or Chi-squared goodness of fit test was used to detect the drift for each feature. The notebook documented how the algorithm decided to use PSI or Chi-squared goodness of fit test in checking feature drift based on factors like feature types and number of categories. Then, the notebook also documented the calculation, validation, and visualization of both drift calculation method. Testing was also conducted to check that both the calculation methods were working as expected.

b) Conda Environment: arf_conda_exp_env (SHAP environment)

c) Execution Time: 3 minutes

4.2.4 Description of Python libraries

The IPYNB notebooks also required the Python libraries to execute. The source code that was going to use in the web service was transferred to a standalone file while its documentation was in the IPYNB notebook. Note that the exact function name and its parameter values would not be discussed since it was well-documented as shown in diagram below. The pseudocode of some important implementations like transfer learning were discussed in Chapter 3.

```
def transfer_tree_weights(task_no, hoeffding_tree, decision_tree, train_data, label_classes,
                          feature_names, nominal_attrs):
    """Transfer the trained weights from the decision tree to the hoeffding tree.

    Parameters
    ----------
    task_no: int
        Task number to trace the original order of trees in the ensemble.
    hoeffding_tree: river.tree.HoeffdingTree
        The object representing the Hoeffding tree.
    decision_tree: sklearn.tree.DecisionTreeClassifier
        The object representing the decision tree.
    train_data: DataFrame
        Training data.
    label_classes: list of int or list of str
        The list containing the class label names.
    feature_names: list of str
        The list containing the X feature names.
    nominal_attrs: list of str
        The list containing the nominal feature names.

    Returns
    -------
    task_no: int
        Task number to trace the original order of trees in the ensemble.
    hoeffding_tree: river.tree.HoeffdingTree
        The Hoeffding tree's object with the transferred weights.
    """
```

Figure 4.2.4.1: The example showing a well-documented Python function.

Besides, all the Python files were implemented by the author. Credits were explicitly stated if the source code was directly copied from somewhere else.

```
def get_shap_color_scale(cmap):
    """
    The color scale construction in Matplotlib library and Plotly library is different. Thus,
    this function is used to convert the default Matplotlib color scale used SHAP's author into
    the format that the Plotly can understand.

    Parameters
    ----------
    cmap: matplotlib.colors.LinearSegmentedColormap
        The default colour scale used by SHAP's visualization is in shap.plots.colors.red_blue

    Returns
    -------
    color_scale: pandas.core.series.Series
        The list containing the 200 hex colors codes. The whole list represent a continuous color scal
    """
    data_to_color_mapper = cm.ScalarMappable(norm=cm.colors.Normalize(0, 1),
                                             cmap=cmap)

    # Get the pre-computed color RGBA values
    c_vals = np.linspace(0, 1, num=200, dtype=np.float64)
    shap_color_vals = data_to_color_mapper.to_rgba(c_vals, bytes=True)
    shap_color_vals = pd.DataFrame(shap_color_vals, columns=['red', 'green', 'blue', 'alpha'])

    # Convert RGBA values to Hex values
    # Source: https://www.educative.io/edpresso/how-to-convert-hex-to-rgb-and-rgb-to-hex-in-python
    def rgb_to_hex_vec(r, g, b):
        return ('#{:02x}{:02x}{:02x}').format(r, g, b)
```

Figure 4.2.4.2: The example showing the source code that is credited to educative.io website author.

The list below showed the brief descriptions of each Python file:

1. arf_cf_transfer_learning.py: This file consisted of the functions to perform transfer learning from Scikit-learn random forest classifier to River adaptive random forest classifier.
2. arf_rg_transfer_learning.py: This file consisted of the functions to perform transfer learning from Scikit-learn random forest regressor to River adaptive random forest regressor.
3. arf_to_dict_conversion.py: This file consisted of the two functions that extracted the tree weights from the River adaptive random forest regressor and classifier, respectively.
4. arf_training.py: This file consisted of the functions that incrementally trained the River adaptive random forest regressor and classifier with new samples.

5. data_preprocessing.py: This file consisted of a class that preprocessed the datasets.
6. explainer_visualization.py: This file consisted of the functions to construct the visualizations of SHAP values and SHAP loss values using diagrams like beeswarm plots, feature importance bar plot, positive and negative model loss bar plot and more.
7. feature_dist_monitoring.py: This file consisted of the functions to check the feature drift using PSI and chi-squared goodness of fit test. The file also contained functions that formatted and constructed the PSI table, PSI plots and chi-squared goodness of fit test.
8. general_utils.py: This file consisted of the functions to serialize and deserialize the dictionaries containing the extracted tree weights in order for the dictionaries to be JSON-convertible.
9. rf_cf_performance_eval.py: This file consisted of the functions to compare and visualize the performance of the tree regressors using table and graphs.
10. rf_rg_performance_eval.py: This file consisted of the functions to compare and visualize the performance of the tree classifiers using table and graphs.

4.3 Cloud Database

An Azure SQL database instance was created to serve as the database for both web applications and web services. The database setup was separated into two stages. The first setup was done before setting up both web application and web service. The second setup was done after the web application had been published to the Azure App Service.

4.3.1 Setup Part I

1. Navigate to the https://portal.azure.com/#create/Microsoft.SQLDatabase to create a new SQL database in Azure portal.
2. For brevity, tables were constructed to document the options selected for each field.

Field	Value
Subscription	Use any subscription that viewers preferred.
Resource group	Use any group that viewers preferred.
Database name	UsedCarDealserhipDatabase
Server	Click "Create new" and refer to the next table for the UI options.
Want to use SQL elastic pool?	No
Compute + storage	General Purpose. Gen5, vCores, 32 GB storage, zone redundant disabled. Click "Configure database" and refer to the next two tables for the UI options.
Backup storage redundancy	Geo-redundant backup storage

Table 4.3.1.1: UI options in "Create SQL Database"

Field	Value
Server name	used-car-dealership-utar-fyp-1800224
Location	(US) East US
Authentication method	Use SQL authentication. Enter the username and password as well.

Table 4.3.1.2: UI options in "Create SQL Database Server"

Field	Value
Service tier	General Purpose (Scalable compute and storage options)
Compute tier	Serverless
Hardware Configuration	Gen5 up to 40 vCores, up to 120 GB memory
Max vCores	2
Min vCores	0.5
Auto-pause delay	Enable auto-pause. Set to 1 hour.
Data max size (GB)	1 GB
Would you like to make this database zone redundant?	No

Table 4.3.1.3: UI options in the "Compute + storage" section in "Create SQL Database"

3. Click "Next: Networking". Configure the options as shown below. The fields that were not mentioned should be left as defaults.

Field	Value
Network connectivity	Public endpoint
Allow Azure services and resources to access this server	No
Add current client IP address	Yes

Table 4.3.1.4: UI options in the "Networking" tab in "Create SQL Database"

4. Click "Next: Security". Click "Not now" in "Enable Microsoft Defender for SQL". Leave the remaining options as default.
5. Click "Next: Additional settings". Select "None" in "Use existing data". Leave the remaining options as default. Finally, click "Review + create" before clicking "Create".

4.3.2 Setup Part II

1. Proceed with the following setup only after the web application had been published to the Azure App Service.
2. Navigate to the homepage of the Azure SQL server instance. In the navigation menu on the left, click “Query editor (Preview)”.

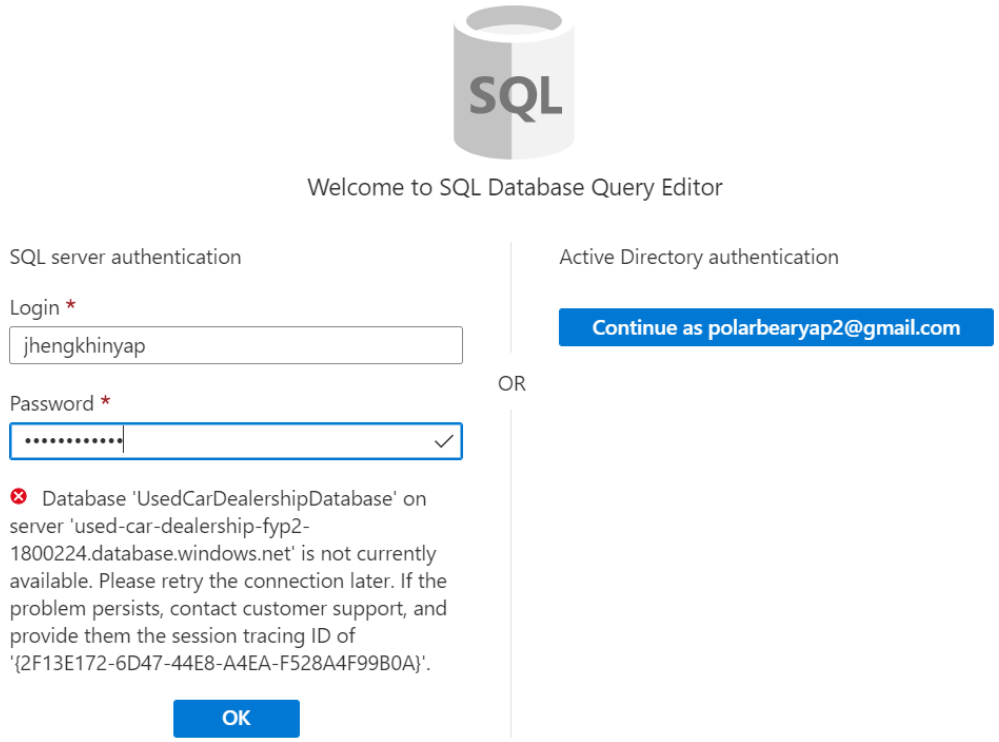


Figure 4.3.2.1: The UI of the Azure SQL database query editor’s login page

3. Enter the username and password under the SQL server authentication. In this project, the auto-pause delay was enabled to save the author’s resources since private Azure account was used. It might take 1 or 2 minutes for the dataset to resume. Then, re-enter the credentials again.
4. Copy the content from the FYP2_Database_Procedure.txt and paste into the editor as show in the image below. Then, click “Run” to execute the query. Refresh the database and four procedures should be created by opening the “Stored Procedures” folder as shown in the left side of the image.
5. Execute the “EXEC <procedure name>” command to test the procedures. The query should return a JSON string.

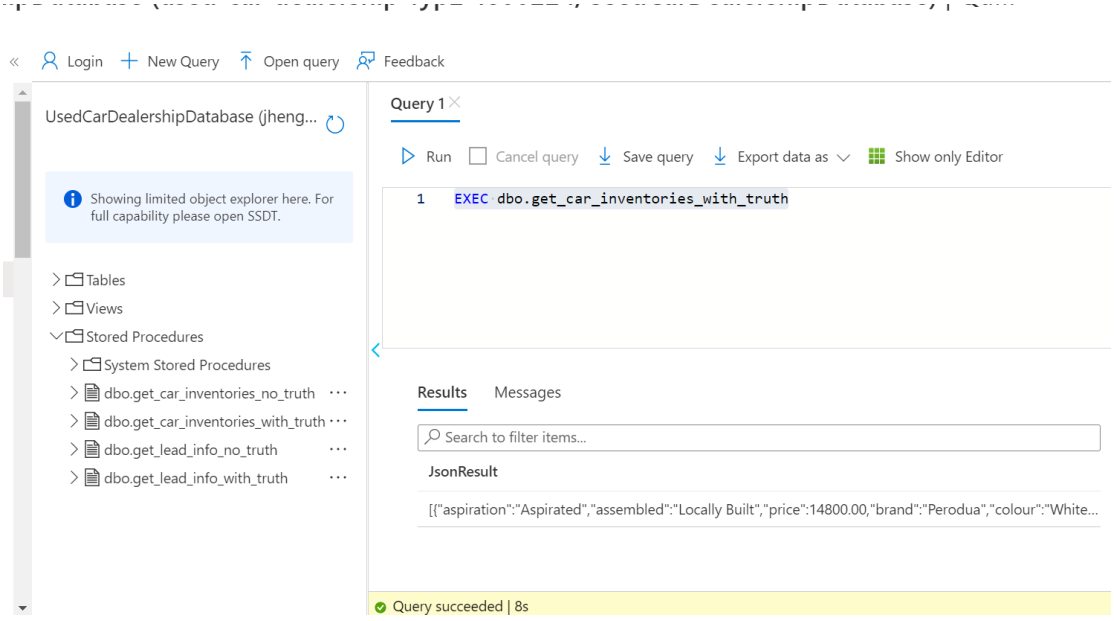


Figure 4.3.2.2: The UI of the Azure SQL database instance homepage.

4.3.3 Procedures

Procedures were created to allow easy retrieval of specific subset of data by just calling the EXEC syntax. A total of four procedures were created to retrieve the car inventory records with truth, car inventory records without truth, lead records with truth, and lead records without truth. Only the records with truth were queried to review the models' overall prediction behaviour and the models' performance. Both records with and without truth were queried to detect and monitor drifts or data errors.

In "Cars" table along with its parent tables, the unused columns in both procedures were row unique identifiers, "Cars.Title", "Cars.CreatedTimestamp", "Cars.UpdateAnalytics", "Cars.PricePerMonth", "Cars.PredictedPrice", and "CarModels.Name". The "AssignedPrice" column, which was the truth of the car inventory records, was only retrieved in the "get_car_inventories_with_truth" procedure. If the "Cars.UpdateAnalytics" was "Yes", then the truth was available. The opposite case was true when the Cars.UpdateAnalytics" was "No".

In "Leads" table, the unused columns in both procedures were "ID", "Name", "Email", "PhoneNo", "DontCall", "PredictedScore", and "CreatedTimestamp". The "Status" column, which was the truth of the lead records, was retrieved in the "get_lead_info_with_truth" procedure. If the "Status" was "Qualified" or "Disqualified", then the truth was available. The opposite case was true when the "Status" was "Active".

The image below showed an example of the four procedures. It was worth noticing that only the columns that were used as features and truth column were retrieved. The query result was converted into JSON when the “FOR JSON PATH” syntax was used. One important thing to note was that the procedure should be retrieving most recent data by setting a configurable starting date when filtering the “CreatedTimestamp” column. In this project, the starting date was fixed to '2022-04-01' since the data was not available in real-time.

```
CREATE OR ALTER PROCEDURE dbo.get_lead_info_with_truth
AS
SET NOCOUNT ON;
SELECT CAST(
    (SELECT
        [DontEmail] AS 'dont_email',
        [DontCall] AS 'dont_call',
        [Occupation] AS 'occupation',
        [ReceivedFreeCopy] AS 'received_free_copy',
        [AvgPageViewPerVisit] AS 'avg_page_view_per_visit',
        [TotalSiteVisit] AS 'total_site_visit',
        [TotalTimeSpendOnSite] AS 'total_time_spend_on_site',
        CASE [Status]
            WHEN 'Qualified' THEN 1
            ELSE 0
        END AS 'converted'
    FROM
        [dbo].[Leads]
    WHERE
        Status != 'Active' AND
        CreatedTimestamp >= Convert(datetime2, '2022-04-01')
    FOR JSON PATH)
    AS NVARCHAR(MAX)
) AS JsonResult
GO
```

Figure 4.3.3.1: The procedure that retrieved the lead information that had truth

4.4 Web Service Artifacts

As mentioned earlier, the SHAP library was incompatible with River library. Thus, two Docker containers were created for the services that only used SHAP library and the services that only used River library. Docker Compose was used to initialize and run these Docker containers so that these containers appear to work as a single web service. The setup and implementation were described in the next two sections.

4.4.1 Setup

1. The setup must be done after the web application had been published to the Azure App Service and the second stage of setup had done for the Azure SQL database.
2. First, the machine must be installed with Docker Desktop to build, run, pull, and push the Docker images and containers. The author used Docker Desktop in Windows operating system in this setup.
3. Open the Docker Desktop Application to start the Docker engine.
4. Copy the web service source code to a desired location.
5. Before building the image, the "docker-compose.yml" file required the connection string of the Azure SQL database in order to query the application data needed for explaining the models and their losses.
6. To do this, navigate to the Azure SQL database instance homepage, then click on "Show database connection strings" as shown below.

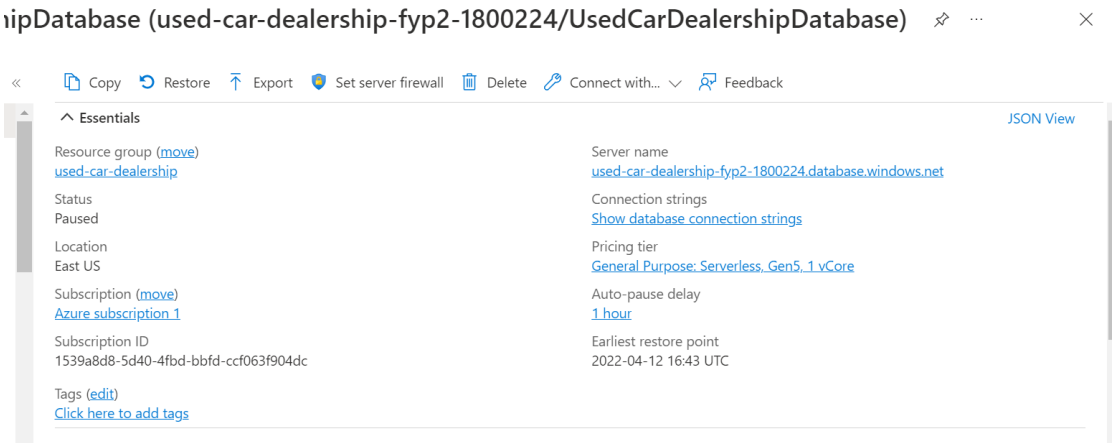


Figure 4.4.1.1: The UI of the Azure SQL database instance homepage.

7. Then, click on the "ODBC" tab. Only copy the connection string as highlighted below. Add the password at the end of the connection string.

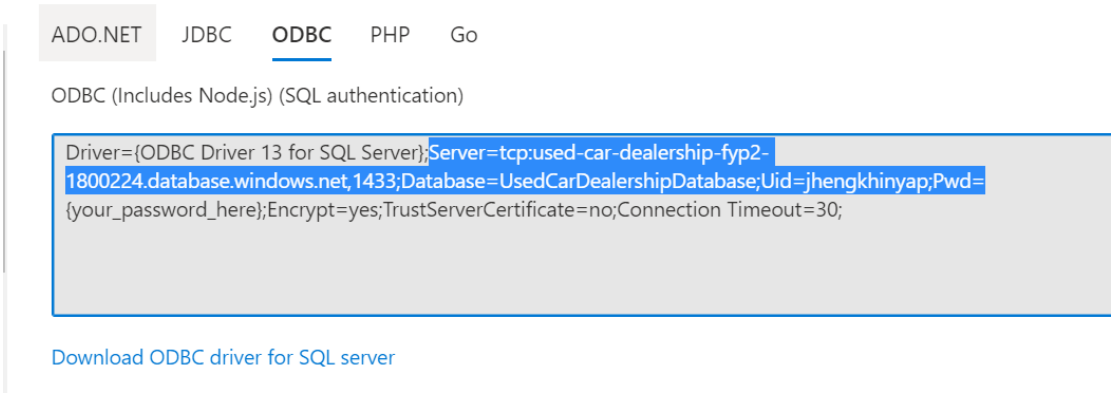


Figure 4.4.1.2: The UI that showed the ODBC database connection string.

8. Paste the database connection string to the location as shown below.

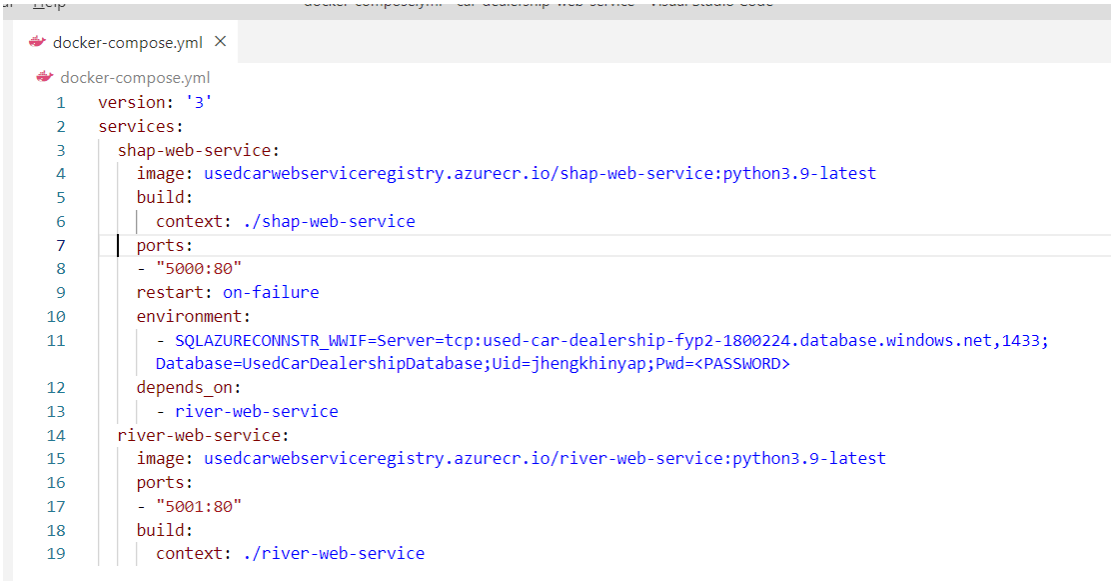


Figure 4.4.1.3: The paste location of the database connection string.

9. The building of the Docker contains began now. Type “cmd” in the search bar to open the command prompt.

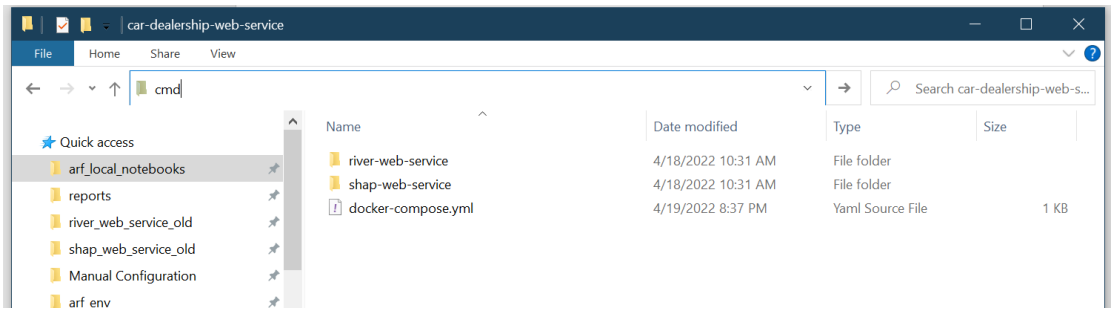


Figure 4.4.1.4: Screenshot that showed on how to open the command prompt right from the file explorer

10. In the command prompt, execute the command below. The initial building time would be at most 20 minutes depending on the internet connection.

>> docker compose build

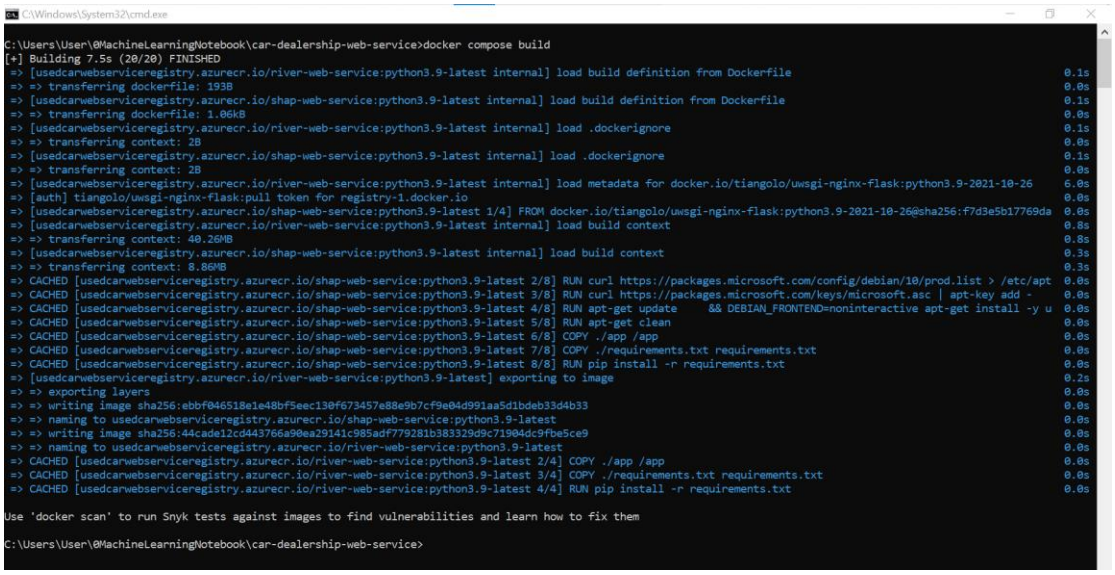


Figure 4.4.1.5: The command output of the "docker compose build"

11. Next, execute the command "docker compose up" to run the multi-container Docker application in the local machine. Add the "-d" flag to omit the display of application's running logs.

>> docker compose up

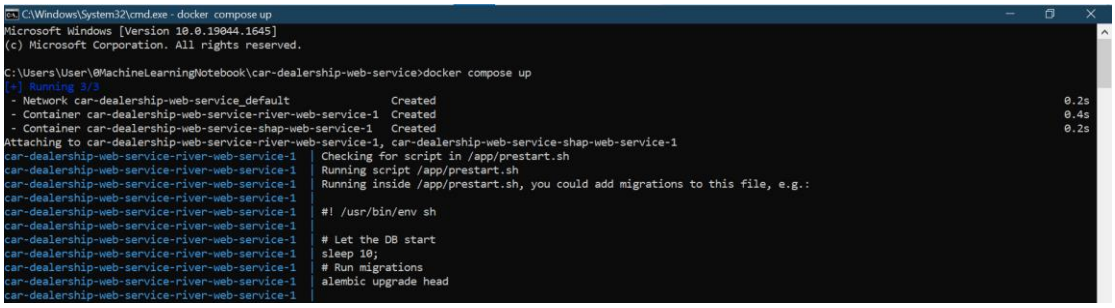


Figure 4.4.1.6: The command output of the "docker compose up"

12. To check the web service was running as expected, use tools like Postman to send a POST request to http://localhost:5000/lead/update/model. The POST response

must be returned as shown below. The remaining tests of web service were discussed in Section 6.1. The request body were also shown below.

```
{
    "dont_email": "No",
    "dont_call": "No",
    "occupation": "Currently Not Employed",
    "received_free_copy": "Yes",
    "avg_page_view_per_visit": 9.0,
    "total_site_visit": 9.0,
    "total_time_spend_on_site": 208.0,
    "converted": 0
}
```

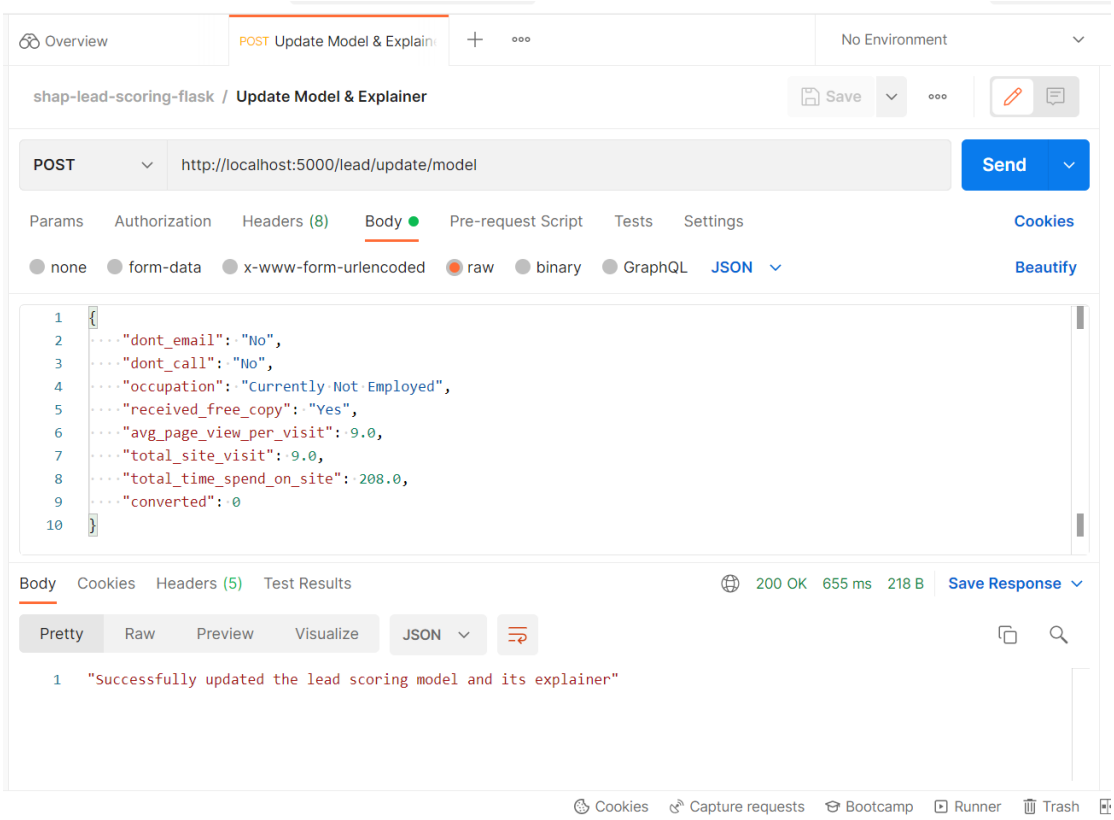


Figure 4.4.1.7: The screenshot showing Postman making a POST request

13. The setup process was completed. To shut down the application, execute the command “docker compose down” at the root directory of the source code.

Unfortunately, the author was ashamed to declare that he was incapable of deploying to the Azure App service the due to inexperience with Docker and Azure cloud services. The deployment of multi-container Docker containers was not as

straightforward as deploying a single Docker image. Please deduct the grade accordingly.

4.4.2 Description of artifacts

docker-compose.yml

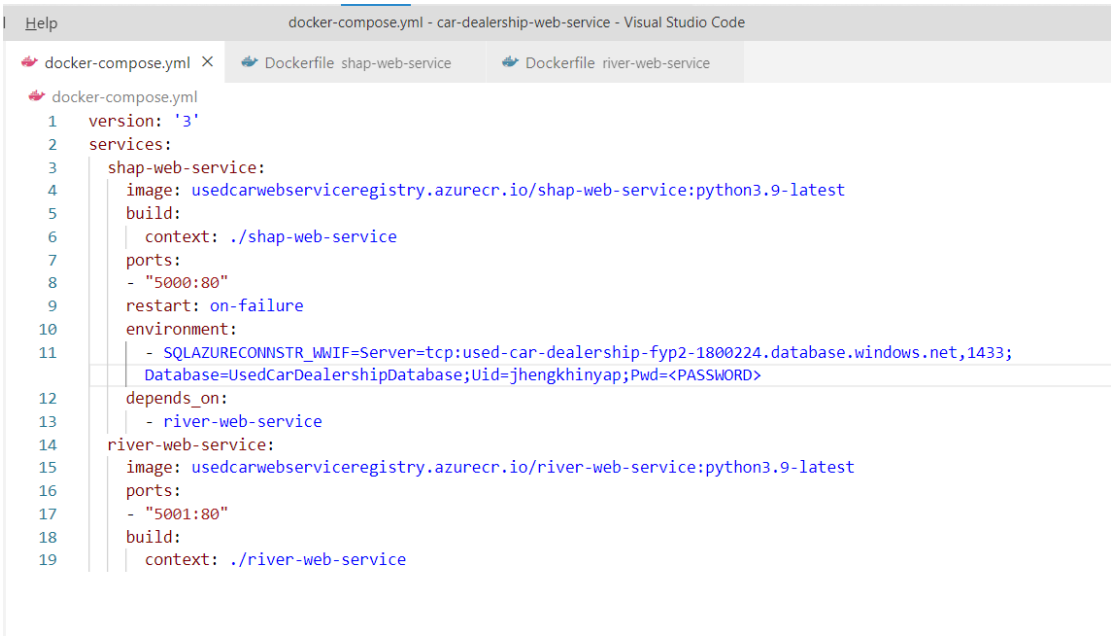


Figure 4.4.2.1: The screenshot showing docker-compose.yml

Below showed some important configuration in docker compose.

1. ports: Nginx HTTP server was used as the production server for running Flask RESTful API. By default, Nginx HTTP server listened for incoming connection on port 80 in the container. Since port 80 was used in the local machine, port 5000 and port 5001 were used to map the incoming connections to the port 80 in the container, respectively.
2. restart: Define the service to restart if encountered an error. In this project, the error could happen when there was connection timeout when querying the application data. This was because the cloud database instance was paused to save cost since individual account was used.
3. depends_on: Configure the shap-web-service to start after the river-web-service had started. It was because shap-web-service was the public facing API and required the incremental training functionality in the backend facing API.
4. Please refer to the Docker official documentations for other configurations.

Dockerfile for SHAP web service

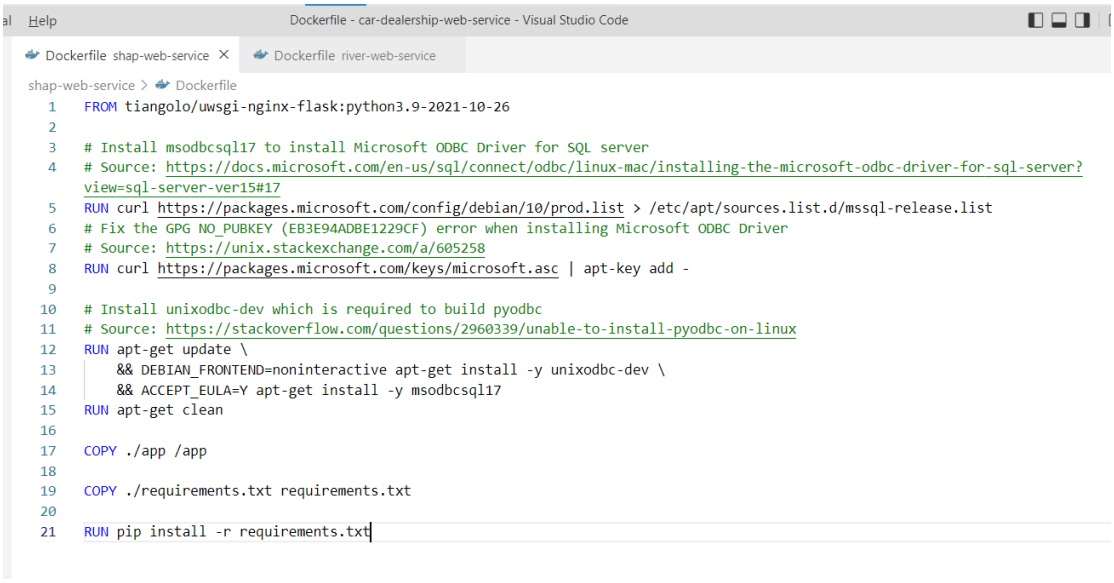


Figure 4.4.2.2: The Dockerfile for SHAP web service

Based on the image above:

1. The Docker image used was from an online community developer named tiangolo. The image was a Debian system that had pre-configured with uWSGI and Nginx for running production Flask applications. The image abstracted away the technical difficulties to configure uWSGI and Nginx from scratch.
2. The SHAP web service required a Microsoft ODBC driver to query the application data from the cloud instance. First, the installation information and Microsoft's public keys were installed to the image. Then, msodbcsql17 was installed into the image which was the Microsoft ODBC Driver 17 for SQL Server.
3. Finally, the source code was copied and the Python dependencies were installed accordingly.

Dockerfile for River web service.

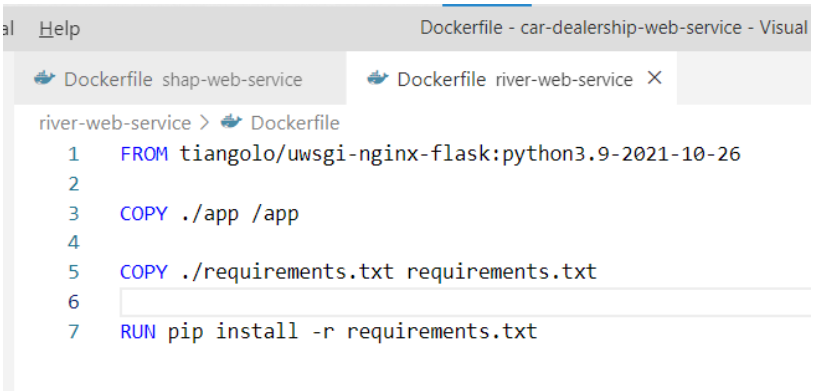


Figure 4.4.2.3: The Dockerfile for River web service

The Dockerfile for River web service was similar to the SHAP web service. No driver was installed since River web service did not need the data.

Web Service's Artifacts

This section explained the important artifacts found in the SHAP web service. Not all details were discussed but the code was already well-documented.

A) Python third-party libraries

The three main Python libraries were used in the SHAP web service, which were SHAP, Flask-RESTful, and Plotly. First, SHAP library was used since the program required tree SHAP explainers and tree SHAP loss explainers. Second, a Flask extension named Flask-RESTful was used to speed up the development of RESTful API. Third, Plotly library was used to construct plots. Plotly was chosen since it was built on top of Plotly JavaScript library (plotly.js). This meant that the plot could be constructed and configured in Python (plotly.py) before converting to JavaScript code embedded in HTML standalone file or div element. The JavaScript code enhanced user experience with a rich set of functionalities such as image downloading, zooming, and data hovering with little or no configuration. Open the notebook HTMLs to see the plots in action.

B) Car Price Model and Lead Scoring Model

The car price model used was the pre-trained adaptive random forest regressor implemented by the River library. The lead scoring model used was the pre-trained adaptive random forest classifier implemented by the River library. {Do Ref on how to find other details}

C) Initialization

During the initialization of the web service:

a) The car price model training set and the lead scoring model training set were loaded. Both training set were used to (1) detect feature drift when there were no truth values and (2) to initialize the tree SHAP loss explainers to explain model loss, respectively.
b) The data pre-processors were loaded to pre-process incoming data and queried data.
c) The dictionaries containing the extracted weights from pre-trained car price model and pre-trained lead scoring model were loaded to serve as the initial model in the web service. The dictionaries would get updated from time to time as POST requests were made to train the River models in the River Web Service. It was important to note that the River model objects were not in SHAP web service since the River library could not be loaded with SHAP library. Instead, only the dictionaries representing the latest models were stored.
d) With the loaded data, pre-processors and model dictionaries loaded, the program could initialize the tree SHAP explainers, tree SHAP loss explainers, and the feature drift detector.

D) Database Connection

To establish the database connection to the Azure SQL server:

a) “ConnectionManager” class and “Queryable” class was created to manage the connection. The class was directly copied from the Microsoft official GitHub repository [19]. Please refer to the repository for better explanations.

E) SHAP API Design

The SHAP web service’s API was designed as shown below. The API design was briefly mentioned since it had been explained in detail in the system design. SHAP web service was the public facing API, while the River web service was the backend facing API. The River web service only provided one functionality, which was to incrementally trained the car price and the lead scoring model.

1) URL: '/car/global/review/model', '/lead/global/review/model'

Method: GET

Description: HTTP GET requests could be made to review the models' overall prediction behaviour. As mentioned in the system design, the web service queried the data that had truth values, calculated the SHAP values based on the data, and finally constructed the beeswarm plot and the feature importance bar plot. The response was a JSON object that contained the two plots that were exported in HTML, respectively.

2) URL: '/car/global/review/performance', '/lead/global/review/performance'

Method: GET

Description: HTTP GET requests could be made to review the models' performance. The web service queried the data that had truth values, calculated the SHAP loss values based on the data, and finally constructed the positive and negative SHAP loss bar plot. Unfortunately, the author was ashamed to declare that the running metrics functionality was not implemented. The response was a JSON object that contained the two plots that were exported in HTML, respectively.

3) URL: '/car/local/review/prediction', '/lead/local/review/prediction'

Method: POST

Description: HTTP POST requests could be made to review the individual car price prediction and lead scoring prediction. The request body consisted of a single car inventory record or a single lead scoring record. The web service then calculated the SHAP value on the record, and finally constructed the SHAP bar plot. The response was a JSON object that contained the SHAP bar plots that was exported in HTML.

4) URL: '/car/local/review/model_loss', '/lead/local/review/model_loss'

Method: POST

Description: HTTP POST requests could be made to review the model losses in both car price model and lead scoring model. The request body consisted of a single car inventory record or a single lead scoring record. The web service then calculated the SHAP value and SHAP loss value on the record, and finally constructed the SHAP bar plot and the SHAP loss bar plot. The response was a JSON object that contained the two plots that were exported in HTML, respectively.

URL: '/car/update/model', '/lead/update/model'

Method: POST

Description: HTTP POST requests could be made to incrementally train the car price regressor and the lead scoring classifier with new samples, respectively. The request body consisted of a single car inventory record or a single lead scoring record. The SHAP web service the forwarded the same record to the River web service for incremental training via POST requests. The River web service would completed the training and the weights of the models would be extracted to the dictionaries. Finally, the dictionaries were returned as the POST responses and the SHAP web service would update the tree SHAP explainer and tree SHAP loss explainer using these dictionaries.

URL: '/car/global/review/drift/truth', '/lead/global/review/drift/truth'

Method: GET

Description: HTTP GET requests could be made to detect and monitor the drift on the application data that had truth values. The web service queried the data that had truth values, calculated the SHAP loss values on the data, used the pre-computed SHAP loss values from validation dataset, checked if there was any significant difference in in SHAP loss mean values or in feature mean values for each feature, before finally constructed the SHAP loss monitoring plot if an alarm was triggered for any feature. The response was a JSON object that contained a list of alarms and a list of SHAP loss monitoring plots constructed for each feature that had drifted or was problematic.

Note that, in this project, the validation set was fixed to use the model training dataset due to low availability of samples (< 10,000 samples for both datasets).

Endpoint: '/car/global/review/drift/no_truth', '/lead/global/review/drift/no_truth'

Method: GET

Description: HTTP GET requests could be made to detect and monitor the drift on the application data that had no truth values. The web service queried the data that had no truth values, used PSI or chi-squared goodness of fit test to check for distribution difference between the validation data and the application data, before finally constructed the PSI plot, PSI table, or chi-squared table if an alarm was triggered for any feature. The response was a JSON object that contained a list of PSI plots, a list of PSI tables, and list of chi-squared constructed for features that had drifted or was problematic. Note that, in this project, the validation set was fixed to use the model training dataset due to low availability of samples (< 10,000 samples for both datasets).

Summarization of API endpoints in SHAP and River web services

For clarification, the table below categorized all the API endpoints into the system submodules.

System submodules	API endpoints
Online AI learning system	**Inventory management:** '/car/update/model' **Lead management:** '/lead/update/model'
AI monitoring system	**Inventory management:** '/car/global/review/drift/truth' '/car/global/review/drift/no_truth' '/car/global/review/performance' '/car/local/review/model_loss' **Lead management:** '/lead/global/review/drift/truth'

	'/lead/global/review/drift/no_truth' '/lead/global/review/performance' '/lead/local/review/model_loss'
Explainable AI in predictive analytics	**Inventory management:** '/car/global/review/model' '/car/local/review/prediction' **Lead management:** '/lead/global/review/model' '/lead/local/review/prediction'

Table 4.4.2.1: The summarization of API endpoints in SHAP and River web services

4.5 Web application artifacts

The author used ASP.NET Core MVC to develop the inventory management module and the lead management module in the web application. The author also used Entity Framework Core migration to seed the application data to the Azure SQL database instance. Below was the setup of the web application from importing the project to deploying to Azure App Service. In this setup, the author used Visual Studio 2019 in Windows operating system.

4.5.1 Setup

Importing Project

1. Open Visual Studio. Click “Open a project or solution”.

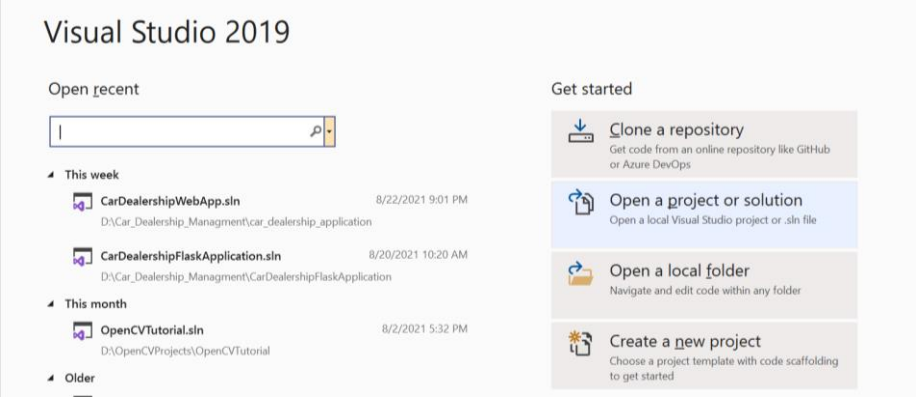


Figure 4.5.1.1: The starting menu of Visual Studio 2019

2. Import the sln file as shown in the image below.

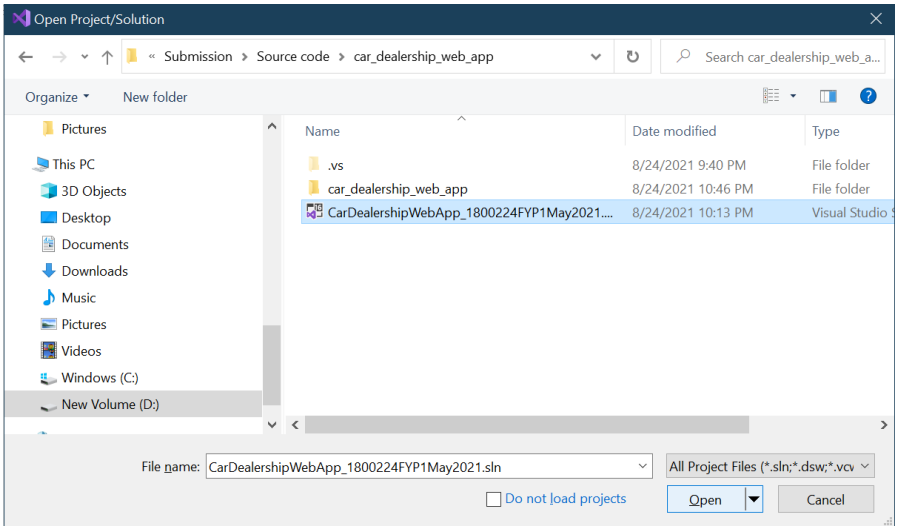


Figure 4.5.1.2: Screenshot that demonstrated the import of sln file

3. In Visual Studio, click “Extensions > Manage Extensions”. Ensure that Microsoft Library Manager was installed.

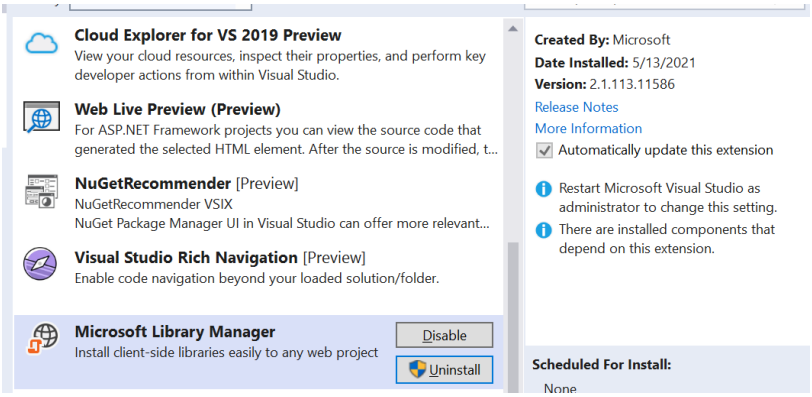


Figure 4.5.1.3: Screenshot showing that the Microsoft Library Manager was installed

4. In the Solution Explorer, right-click on “libman.json” and then click on “Restore Client-Side Libraries”.
5. Open a command prompt inside the **“**car_dealership_web_app”. The diagram below showed one way of doing so. In the file explorer, type “cmd” in the search bar and click enter.

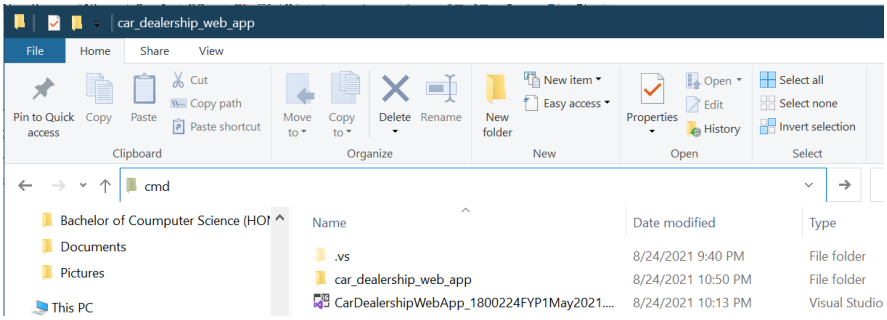


Figure 4.5.1.4: Screenshot showing on how to launch command prompt straight from the file explorer

6. Ensure that the ending file path was “car_dealership_web_app/car_dealership_web_app” not “car_dealership_web_app”.

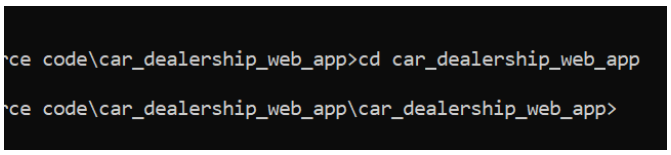


Figure 4.5.1.5: Screenshot showing the right file path to execute the “npm install” command

7. Execute the command “npm install”.

```
D:\Car_Dealership_Managment\0DataDrift\FYP\Submission\Source code\car_dealership_web_app\car_dealership_web_app>npm install
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependencies.

added 529 packages, and audited 530 packages in 19s

27 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (4 low, 5 moderate)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

Figure 4.5.1.6: The command output of the "npm install"

8. Then, go back to Visual Studio. Right-click on "gulpfile.js" and then click on "Task Runner Explorer".

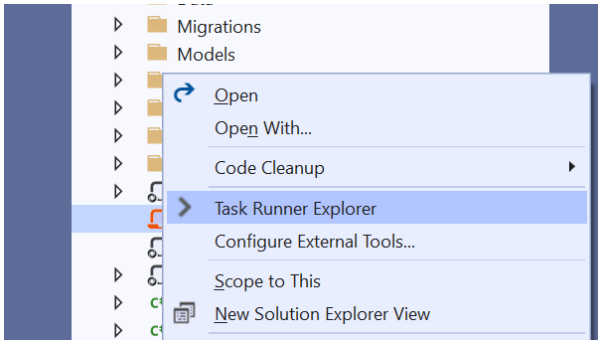


Figure 4.5.1.7: The UI location of "Task Runner Explorer"

9. Click "Refresh".

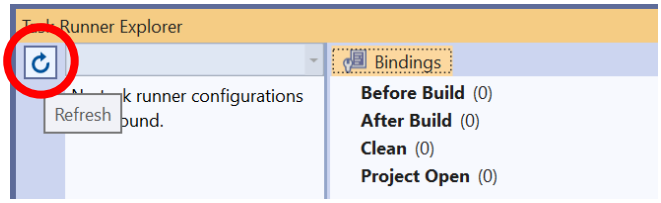


Figure 4.5.1.8: The UI location of Refresh

10. Ensure that webpack is bind to "Before Build". If not, right click on "webpack", click "Bindings"> "Before Build".

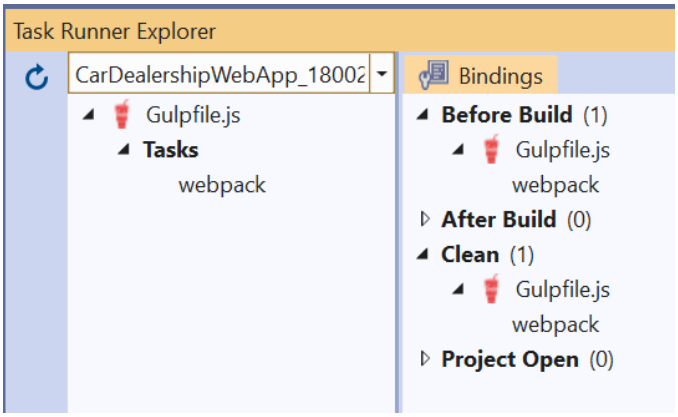


Figure 4.5.1.9: The screenshot showing the webpack was bind to "Before Build"

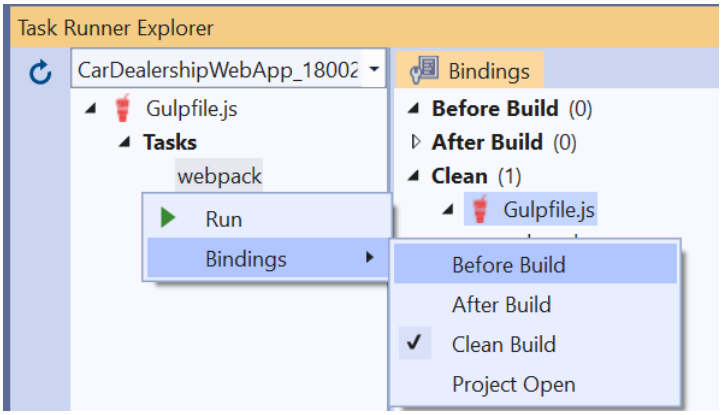


Figure 4.5.1.10: The screenshot showing on how to bind the webpack to "Before Build"

11. Then, right click "Tools" and click "NuGet Package Manager" > "Package Manager Console"**.**
12. In the Package Manager Console, execute the two commands below.

>> add-migration InitialCreate

>> update-database

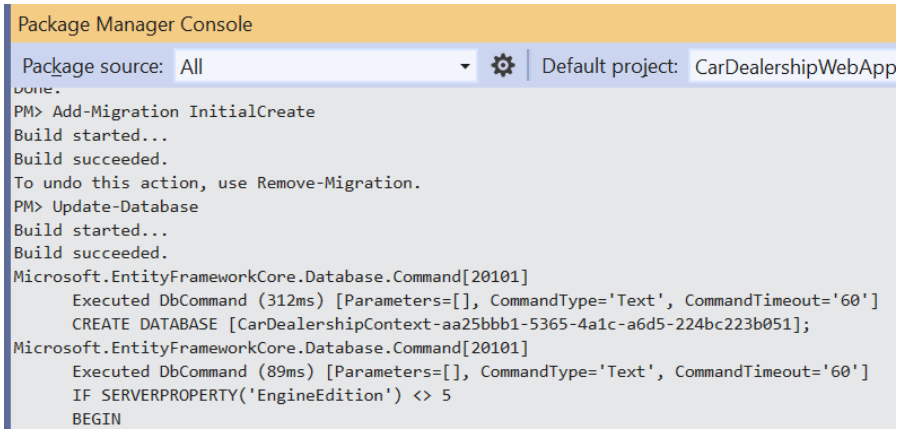


Figure 4.5.1.11: The command output of adding and applying the migration to the local database

13. Run the web application as shown in the diagram below.

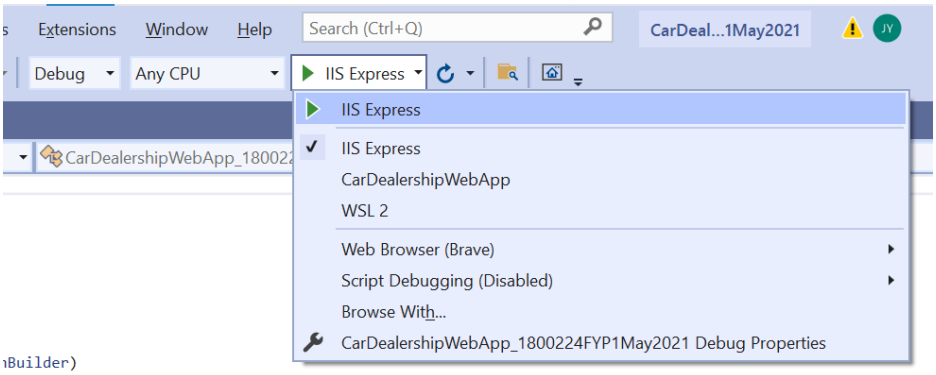


Figure 4.5.1.12: Screenshot that demonstrated on how to run the web application

Cloud Deployment

1. Navigate to https://portal.azure.com/#create/Microsoft.WebSite to create an Azure Web App instance in Azure portal, if the build was successful. For brevity, tables were constructed to document the options selected for each field.

Field	Value
Subscription	Use any subscription that viewers preferred.
Resource group	Use any group that viewers preferred.
Database name	UsedCarDealserhipDatabase
Name	used-car-dealership-fyp2-1800224
Publish	Code
Runtime stack	.NET 5
Operating System	Windows
Region	East US
Sku and size	Free F1 Shared infrastructure, 1 GB memory
Zone redundancy	Disabled

Table 4.5.1.1: UI options in "Create Web App"

2. Click "Next > Deployment". Disable the continuous deployment. Click "Next > Network (preview)" and disable the network injection. Click "Next > Monitoring" and disable the application insights. Click "Review + create" before clicking "Create".
3. If the build was successful, right click on the "CarDealershipWebApp". Then, click "Publish…".

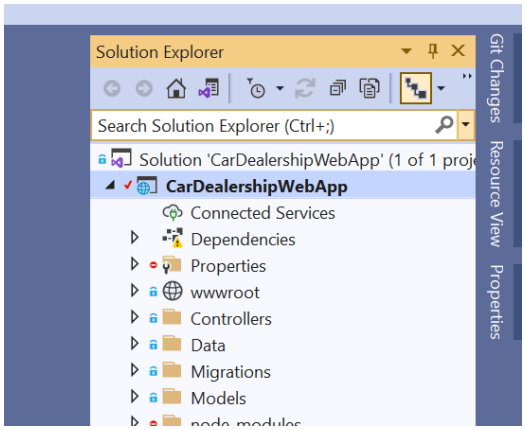


Figure 4.5.1.13: UI location of the "CarDealershipWebApp" as highlighted in bold text

4. Click the "Azure App Service (Windows)" in the specific target navigation menu.

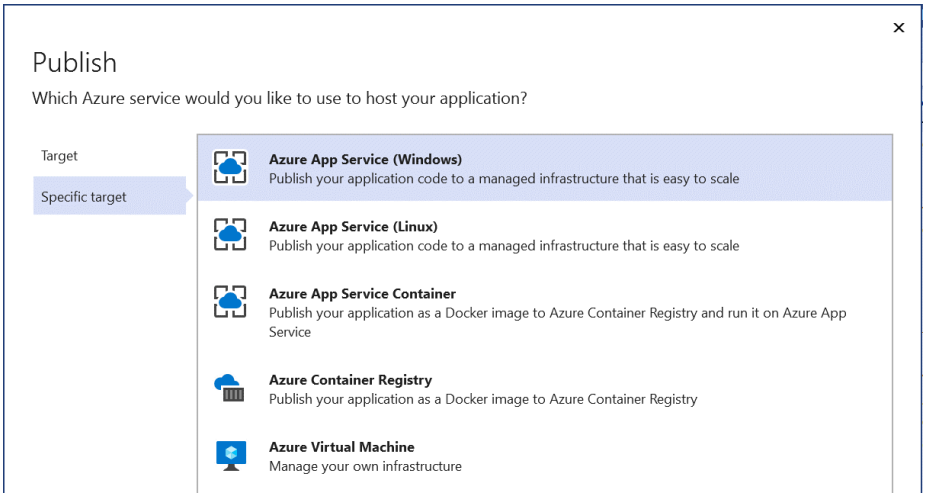


Figure 4.5.1.14: UI location of the "Azure App Service (Windows)"

5. Click on the Azure App service instance created just now. Then, click "Next".

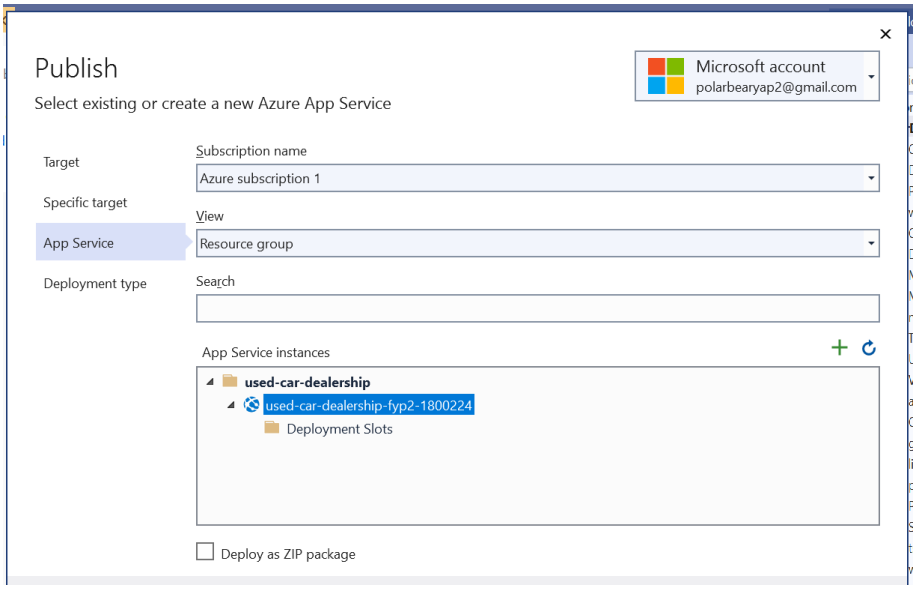


Figure 4.5.1.15: Screenshot showing the name of the Azure App Service instance

6. Click “Publish (generates pubxml file)” before clicking “Finish”.

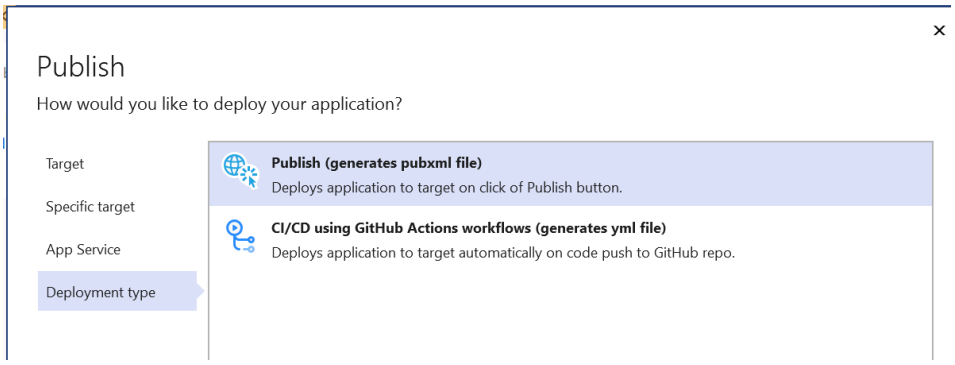


Figure 4.5.1.16: UI location of the “Publish (generates pubxml file)”

7. In the newly generated pubxml file, click “More actions” and then click “Edit”.

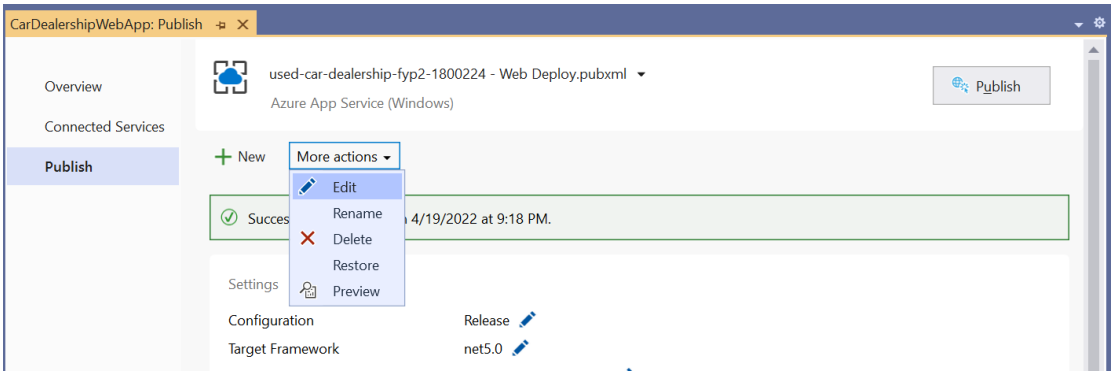


Figure 4.5.1.17: Screenshot showing the UI location of the Edit Button.

8. Ensure that the publish settings matched with the image shown below.

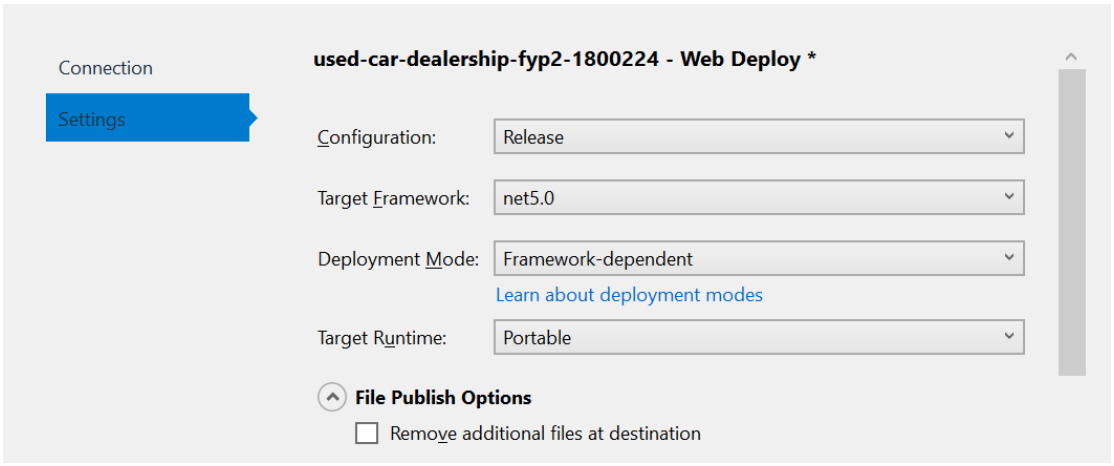


Figure 4.5.1.18: Screenshot used to validate the web app’s publish settings

9. Click on the “CarDealershipContext” and replace the local database connection string with the connection string of the Azure SQL database created just now.

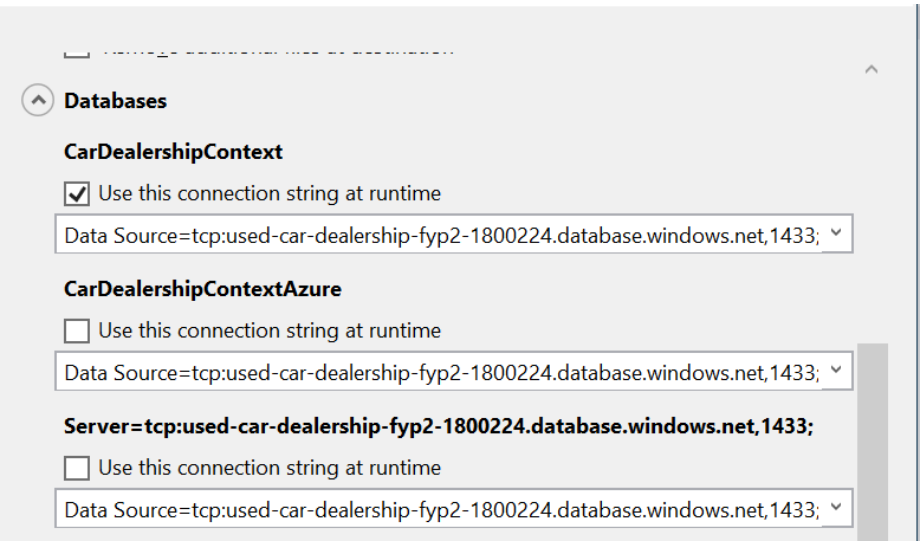


Figure 4.5.1.19: Screenshot showing the configuration of the database

10. To retrieve the connection string, navigate to the Azure SQL database instance homepage, then click on "Show database connection strings" as shown below.

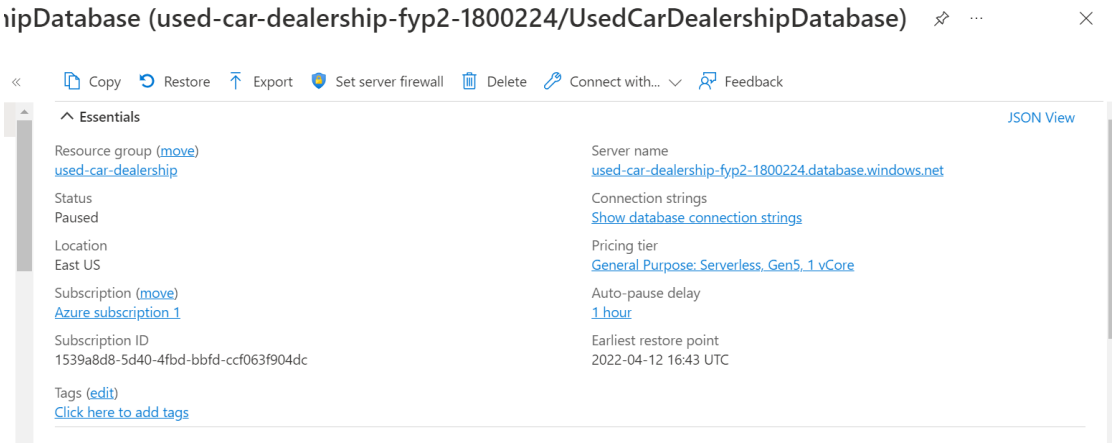


Figure 4.5.1.20: The UI of the Azure SQL database instance homepage.

11. Then, click on the "ODBC" tab. Only copy the connection string as highlighted below. Add the password at the end of the connection string.

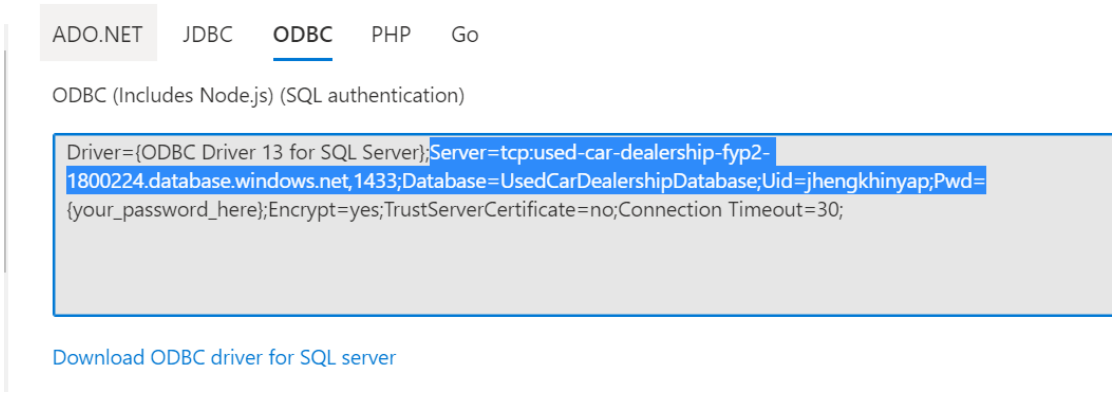


Figure 4.5.1.21: The UI that showed the ODBC database connection string.

12. Copy the same connection string to the "CarDealershipContext" under the "Entity Framework Migration" section to seed the application data to the cloud database.

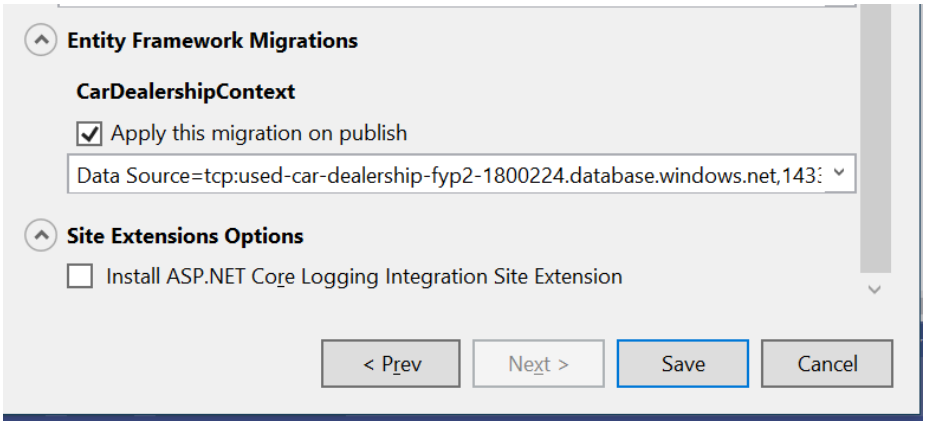


Figure 4.5.1.22: Screenshot showing the configuration of the entity framework migrations

13. Finally, click "Publish" and the web application would be deployed to the Azure App Service.

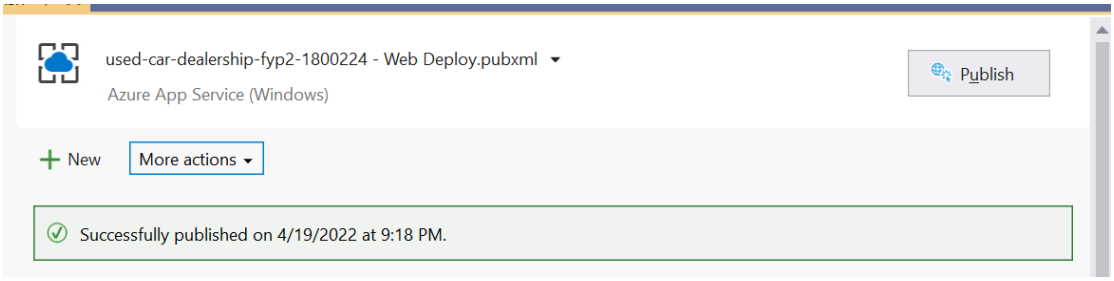


Figure 4.5.1.23: Screenshot showing that the deployment was successful

14. Navigate to the website to check that the application worked as expected. The functionality testing was discussed in Section 6.2.

4.5.2 Description of artifacts

In the web application, only the important parts of the artifacts were discussed since the implementation was quite common and could be found in online tutorials from the internet.

Models

CS File Name	File location relative to	Description

	Models	
CarBluePrint LeadBluePrint	./Interfaces	CarBluePrint and LeadBluePrint were the blueprint classes that converted the HTML-encoded values to the column values that were saved in the database. For example, if the users selected value 5 in the HTML drop-down menu in the "Aspiration" field. Then, the application server would map the value 5 to "Twin Turbo intercooled", which was the value stored in the database.
CarViewModel LeadViewModel	./ViewModels	CarViewModel and LeadViewModel were the classes that used the "ViewModel" pattern to only extract the necessary attributes that used to display to the user interface. Furthermore, these classes also enforce the constraint and format of the field values to ensure that the values persisted to the database were not faulty so that the models would not get corrupted from training with erroneous data.
Car	.	Car, CarBrand, and CarModel were the classes that contained the car specifications that were needed by the car price model to perform inference. Additionally, Car also contained some extra attributes like "Title" and "PricePerMonth" that the inventory manager would like to manage.
Lead	.	Lead was the class that contained the lead attributes that were needed by the lead scoring model to assign lead score. Additionally, the class also contained some extra attributes like "Name" and "PhoneNo" that the sales employees would like to manage.
SeedData	.	SeedData was the class that seeded the car inventory records and lead records in order to populate both the local database and the cloud database.

Table 4.5.2.1: The summary of Model classes

Controllers

Both the CarsController and the LeadsController classes defined the following controller actions:

Controller actions	Description
/Cars /Leads	The controller actions retrieved the up to 50 records from the database to display in a tabular format per request. Additionally, the GET parameters like "sortOrder" and "searchString" were added to allow the application users to search records, sort columns in order, and control the number of records displayed.
/Cars/Details /Leads/Details	The controller actions returned the record where its unique identifier was equivalent to the GET parameter named "id".
/Cars/Create /Leads/Create	For HTTP GET requests, the controller actions returned the record with empty values. For HTTP POST requests, the controller actions inserted the record to the database.
/Cars/Edit/<ID> /Leads/Edit/<ID>	For HTTP GET requests, the controller actions returned the record where its unique identifier was equivalent to the GET parameter named "id". For HTTP POST requests, the controller actions updated the record in the database.
/Leads/Delete/<ID> /Leads/Delete/<ID>	For HTTP GET requests, the controller actions returned the record where its unique identifier was equivalent to the GET parameter named "id". For HTTP POST requests, the controller actions deleted the record from the database.

Table 4.5.2.2: The summary of Controller classes

The basic functionalities of both lead management module and inventory management module were implemented. Unfortunately, the author was extremely ashamed that he did not implement the controller actions to connect the web service functionalities to the web application. Please deduct the grade accordingly.

TypeScript and Webpack

Various third-party libraries like Bootstrap, jQuery and TypeScript were required and loading the libraries individually on the client's browser would slow down network performance. Instead, Webpack was used to compile the client-side TypeScript codes and their dependencies into one or more single optimised, minified, and portable versions of JavaScript codes. It could be understood that Webpack played an important role as a linker in compiling the TypeScript codes with dependencies into one single executable JavaScript code.

In the artifacts, tsconfig.json and webpack.config.js were created to configure the compiler to compile the TypeScript files into a bundled JavaScript file. First, the TypeScript modules were exported in the main.ts in the TypeScripts folder.

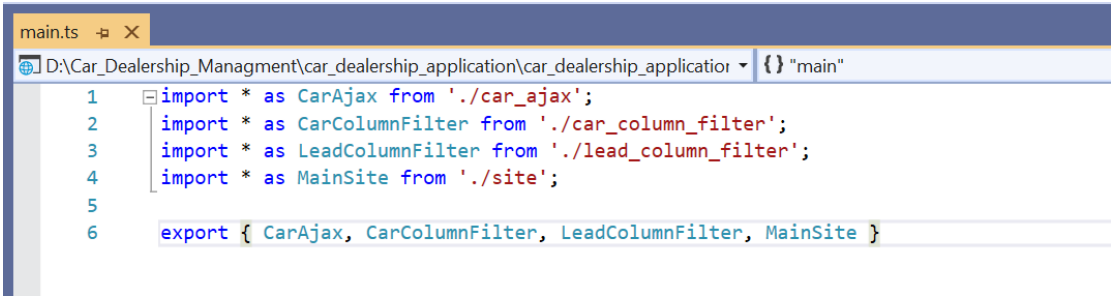


Figure 4.5.2.2: Screenshot showing the syntax to export the TypeScript modules

After the compilation, the compiled JavaScript file would be stored in the folder wwwroot/js. In the shared template named Views/Shared/_layout.cshtml, the JavaScript modules were then loaded depending on the page type. For example, the CarAjax JavaScript module was loaded when creating or editing car records to automatically populate the car models input field as soon as the application users selected a corresponding value in the “car brand” input field using AJAX.

```
    <!-- Load custom script last -->
    <script src="~/js/app.bundle.js" asp-append-version="true"></script>
    @{
        string title = (string)ViewData["Title"];
        string subject = (string)ViewData["Subject"];
    }
    <script>
        App.MainSite.load();
    </script>
    @* Load custom scripts *@
    @if (title == "Cars")
    {
        <script>
            App.CarColumnFilter.load();
        </script>
    }
    @if (title == "Leads")
    {
        <script>
            App.LeadColumnFilter.load();
        </script>
    }
    @if ((title == "Edit" || title == "Create") && subject == "Car")
    {
        <script>
            App.CarAjax.load();
        </script>
    }
</body>
```

Figure 4.5.2.3: Screenshot showing the syntax to export the JavaScript modules based on the page type

CHAPTER 5 EXPERIMENT AND VALIDATION

5.1 Transfer Learning

A total of three tests were conducted to validate the transfer learning algorithms. The implementation of the test codes for the adaptive random forest classifier and regressor could be found at FYP2_ARF_CF_Transfer_Learning.ipynb and FYP2_ARF_RG_Transfer_Learning.ipynb, respectively.

The first test was conducted to compare the total number of nodes of each base learners between the adaptive random forests (ARF) and the traditional random forests (TRF). The transfer learning algorithm was successful if the total number nodes in both models was the same. The number of nodes can be compared by getting the attributes for the respective base learners. The table below on the left proved that the transfer learning classifier algorithm was successful while the table below on the right proved that the transfer learning regressor algorithm was successful.

	TRF	ARF	match
1	1205	1205	True
2	965	965	True
3	1033	1033	True
4	979	979	True
5	1117	1117	True
6	1241	1241	True
7	1197	1197	True
8	1217	1217	True
9	1041	1041	True
10	1185	1185	True
11	1211	1211	True
12	1093	1093	True
13	995	995	True
14	1081	1081	True
15	1065	1065	True

	TRF	ARF	match
1	829	829	True
2	873	873	True
3	839	839	True
4	777	777	True
5	739	739	True
6	873	873	True
7	791	791	True
8	863	863	True
9	819	819	True
10	809	809	True
11	829	829	True
12	867	867	True
13	841	841	True
14	787	787	True
15	843	843	True

Figure 5.1.1: The tables that were used to compare the number of nodes between TRF and ARF

The second test was conducted to compare the tree structures of each base learners between the adaptive random forests (ARF) and the traditional random forests (TRF). The test was conducted to ensure that the nodes were placed at the right place in the ARF. The test code outputted the placements of each node for ARF into a log file named arf_debug.txt, while the placements of each node for TRF was found at trf_debug.txt. As shown in the image below, both file contents must look the same to prove the right placements of all nodes. Each record in the file represented the location of the current internal node, its children nodes' indexes, and the number of samples arrived at the current internal node. For example, if the node location was “001”, then the node was the right child of the left child of the left child of the root node.

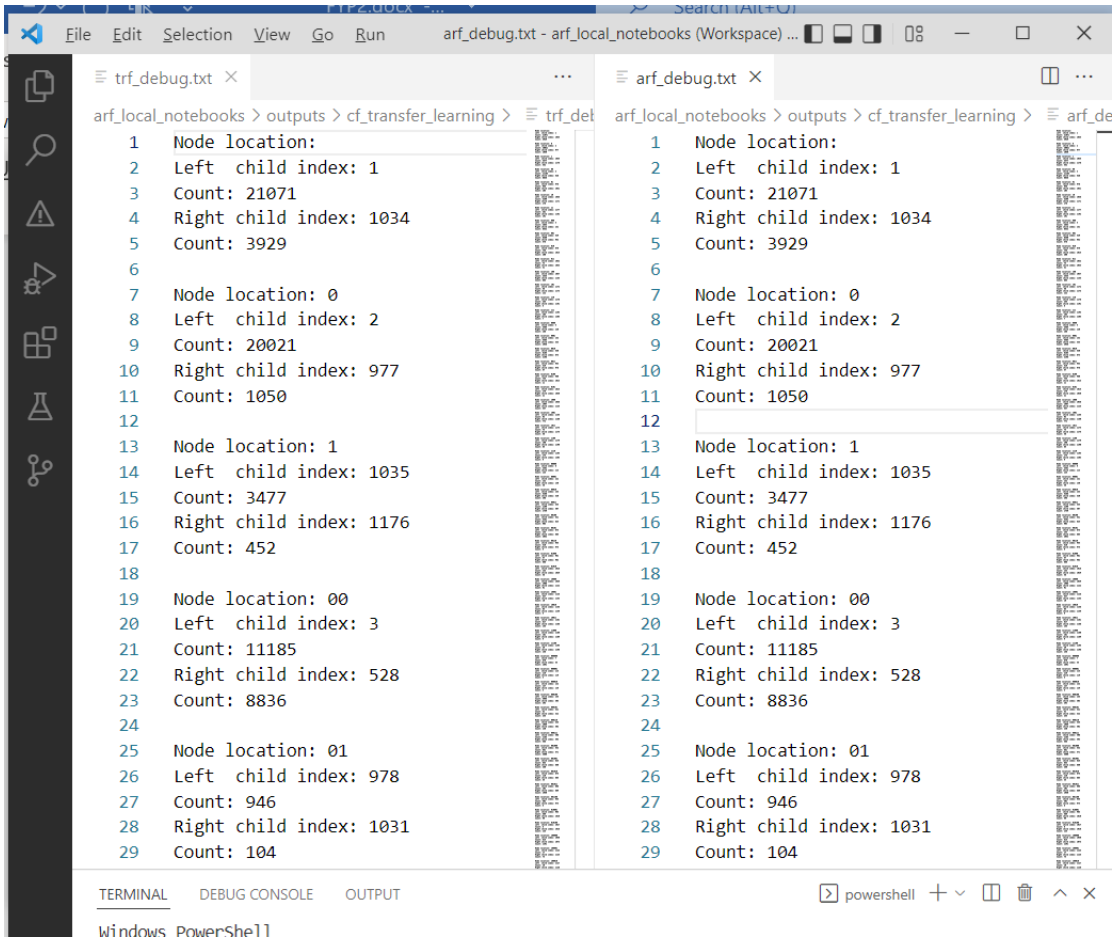


Figure 5.1.2: The comparison between trf_debug.txt and arf_debug.txt for validating the transfer learning classifier algorithm

The third test was conducted to ensure that the adaptive random forests that had undergone transfer learning could resume training just like the ones that were trained from scratch. The source code for training the adaptive random forest classifier and regressor could be found at jupyter_notebooks/arf_training.py.

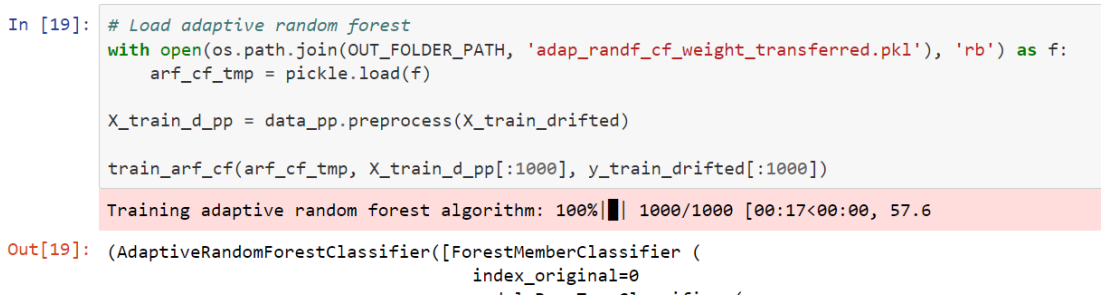


Figure 5.1.3: Screenshot showing that the incremental training of pre-trained adaptive random forest classifier was successful

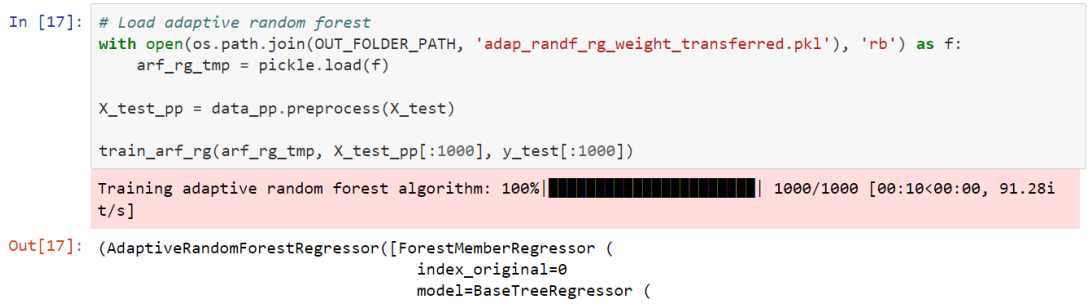


Figure 5.1.4: Screenshot showing that the incremental training of pre-trained adaptive random forest regressor was successful

5.2 Performance Evaluation

In the following sections, experiments were conducted to measure the performance of models in offline and online settings. In offline settings, the offline data was fully available and could be used to train both adaptive random forest models and traditional random forest models. While in online settings, the online data was incrementally available and thus could only be used to train adaptive random forest models.

For classification, an experiment was conducted to evaluate the overall performance of the adaptive random forest classifier that had undergone transfer learning. In the experiment, the offline and online performance between the pre-trained adaptive random forest (ARF) classifier and the traditional random forest (TRF) classifier were compared. Since the tree weights of the TRF classifier was transferred to the pre-trained ARF classifier, the performance similarity was analysed to validate the transfer learning classifier algorithm. Besides, a new adaptive random forest classifier was created and trained from scratch to serve as the baseline performance for the pre-trained adaptive random forest classifier. The purpose of the experiment was to prove that:

1. The pre-trained ARF classifier's classification performance was at least good or better than ARF classifier that was trained from scratch in both offline settings and online settings.
2. The performance of the pre-trained ARF classifier was at least good or better than TRF classifier during initial training.
3. The performance of the ARF classifier was better than TRF classifier under the influence of data drift or concept drift.

Then, the lead scoring model selection was conducted among these three models by analysing the model performance and the tree structure.

The exact same setup was also applied when conducting the experiment for tree ensemble regression models. After conducting the experiment, the car price model selection was conducted among the three models by analysing the model performance and the tree structure.

Setup required to visualize model performance over time

Both experiments computed the running metrics to visualize the performance of ARF classifier and TRF classifier over time. The running metrics were calculated in four different stages, which were (1) offline train settings, (2) offline test settings, (3) online train settings, and (4) online test settings. By separating the calculation of running metrics into stages, the graph could show more useful information by comparing performance across stages.

Below showed the calculation of the running metrics in detail:

1. Traditional random forest model
    - For every increment of 100 samples, get the predictions from the beginning up to the current increment and update the running metrics.
2. Pre-trained adaptive random forest model
    - Offline train set: Running metrics were not available since the weights were directly transferred. Thus, the calculation method was the same as traditional random forest.
    - Online train set: The running metrics were directly retrieved from the model training process. The running metrics were updated for every 100 training samples.
    - Offline test and online test set: The calculation method was the same as traditional random forest.
3. Adaptive random forest model that was trained from scratch
    - Offline train and online train set: The running metrics were directly retrieved from the model training process. The running metrics were updated for every 100 training samples.
    - Offline test and online test set: The calculation method was the same as traditional random forest.

5.2.1 Experimental dataset: AGRAWAL dataset

An experiment was conducted to evaluate the performance of adaptive random forest classifier and traditional random forest classifier with and without the influence of drift. A robust concept drift generator from scikit-multiflow API was used to simulate drift in the experiment. The AGRAWAL stream generator from the same API was used to supply the data to the concept drift generator for generating the drifted stream. In addition, the perturbation parameter was set to 0.10 to introduce some noise into the data.

According to the official documentation, the image below showed the features generated by the AGRAWAL stream generator.

feature name	feature description	values
salary	the salary	uniformly distributed from 20k to 150k
commission	the commission	if (salary < 75k) then 0 else uniformly distributed from 10k to 75k
age	the age	uniformly distributed from 20 to 80
elevel	the education level	uniformly chosen from 0 to 4
car	car maker	uniformly chosen from 1 to 20
zipcode	zip code of the town	uniformly chosen from 0 to 8
hvalue	value of the house	uniformly distributed from 50k x zipcode to 100k x zipcode
hyears	years house owned	uniformly distributed from 1 to 30
loan	total loan amount	uniformly distributed from 0 to 500k

Figure 5.2.1.1: Screenshot showing the features that were generated by the AGRAWAL stream generator

To test the models' performance without drift, the 30,000 samples were generated using the API to represent the offline data. To test the models' performance with drift, the 30,000 samples were generated using the API to represent the online data. 5,000 out of 30,000 samples were chosen as the test set to evaluate the generalization performance of the models in online and offline settings.

Offline Training Performance

The graph below compared the training performance between TRF classifier and ARF classifier in the offline setting. It could be inferred that the transfer learning process was successful as the ROC AUC score of both TRF classifier and pretrained ARF classifier differed no more than 3%. There was a slight difference in the performance since the leaf prediction mechanisms in the TRF classifier and the ARF classifier were different.

On the other hand, the training performance of ARF classifier that was trained from scratch was the lowest with the ROC AUC score difference of 5% as compared to pretrained ARF classifier. However, the result could not guarantee the superiority of the transfer learning classifier algorithm for other datasets, as shown in the next section.

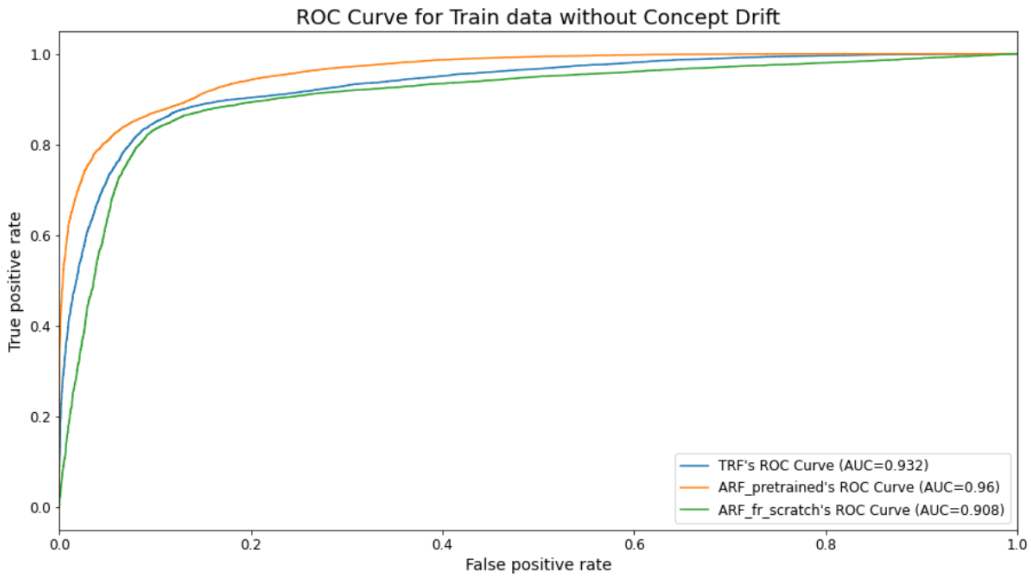


Figure 5.2.1.2: The offline training performance of tree classifiers (AGRAWAL dataset)

Offline Generalization Performance

The graph below compared the generalization performance between the TRF classifier and the ARF classifier in the offline setting. The graph again proved that transfer learning process was successful as the ROC AUC score of both TRF classifier and pretrained ARF classifier differed no more than 1%. On the other hand, the generalization performance of ARF classifier that was trained from scratch was

the lowest with a ROC AUC score difference of 2% as compared to pretrained ARF classifier.

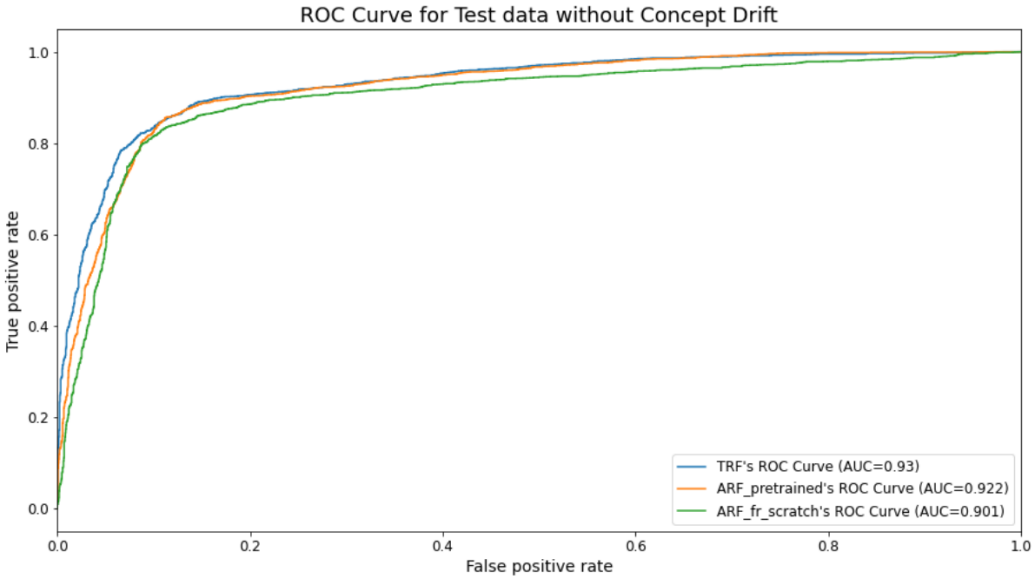


Figure 5.2.1.3: The offline generalization performance of tree classifiers (AGRAWAL dataset)

Online Training Performance

Model type	Offline Train ROC AUC	Online Train ROC AUC	Difference
TRF	0.932	0.680	-0.252
Pre-trained ARF	0.960	0.833	-0.127
ARF trained from scratch	0.908	0.755	-0.153

Table 5.2.1.1: The comparison between offline and online training performance for tree classifiers (AGRAWAL dataset)

The graph below compared the training performance between TRF classifier and ARF classifier in the online setting. Due to drift, all the models' performance were affected. The train performance of TRF classifier had the most impact with a drop of 25% as compared to offline one. The pre-trained ARF classifier's online train performance dropped about 13% while the performance for ARF that was trained from scratch dropped about 15%. It could be deduced that ARF algorithm was more robust against drift as compared to TRF algorithm.

Among the ARF models, the training performance of pre-trained ARF classifier was better than ARF that was trained from scratch with a ROC AUC score difference of 8%. However, the result again could not guarantee the superiority of the transfer learning algorithm for other datasets. It is because the transfer learning algorithm might not give any significant performance improvement for some datasets, which was discussed in the next section.

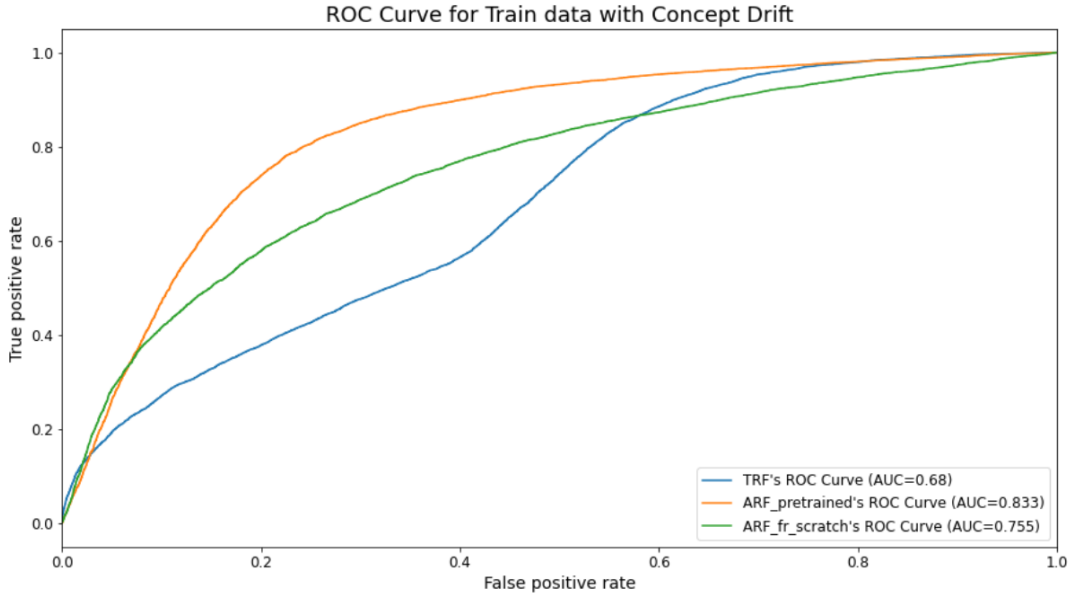


Figure 5.2.1.4: The online training performance of tree classifiers (AGRAWAL dataset)

Online Generalization Performance

Model type	Offline Test ROC AUC	Online Test ROC AUC	Difference
TRF	0.930	0.641	-0.289
Pre-trained ARF	0.922	0.899	-0.023
ARF trained from scratch	0.901	0.800	-0.101

Table 5.2.1.2: The comparison between offline and online generalization performance for tree classifiers (AGRAWAL dataset)

The graph below compared the online generalization performance between TRF classifier and ARF classifier. The table above displayed the difference between offline and online generalization performance of each model. Due to drift, the online generalization performance of TRF classifier dropped about 29% as compared to offline one. The pre-trained ARF classifier's online generalization performance

dropped about 2% while the performance for ARF classifier that was trained from scratch dropped about 10%. It could be deduced that ARF algorithm was robust against drift but not TRF algorithm.

On the other hand, the online generalization performance of the pre-trained ARF classifier was better than ARF classifier that was trained from scratch with a difference of 10%. Again, the result could not guarantee the superiority of the transfer learning algorithm for other datasets, which was discussed in the next section.

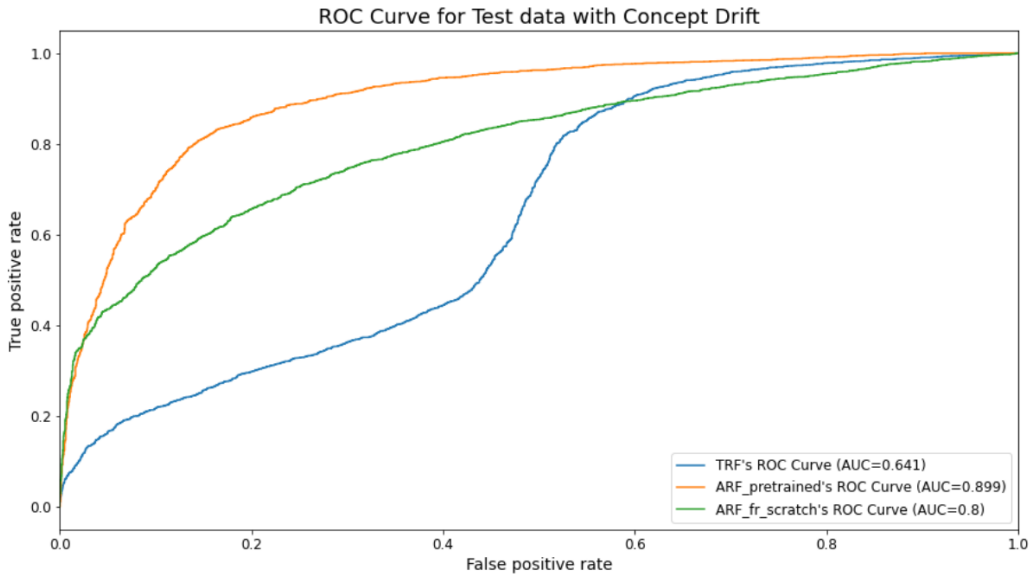


Figure 5.2.1.5: The online generalization performance of tree classifiers (AGRAWAL dataset)

Overfitting Issues

Based on the table below, the difference of ROC AUC in the offline and online set was calculated by subtracting the training ROC AUC score with the testing ROC AUC score for each set. The result below showed that the ARF classifiers were free from overfitting issue when training with online samples.

Model type	Diff ROC AUC in Offline set	Diff ROC AUC in Online set
TRF	0.932 - 0.930 = 0.002	0.680 - 0.641 = 0.039
Pre-trained ARF	0.960 - 0.922 = 0.038	0.833 - 0.899 = -0.066
ARF trained from scratch	0.908 - 0.901 = 0.007	0.755 - 0.800 = -0.045

Table 5.2.1.3: The table that checked the overfitting issues for tree classifiers (AGRAWAL dataset)

Visualizing Model Performance over Time

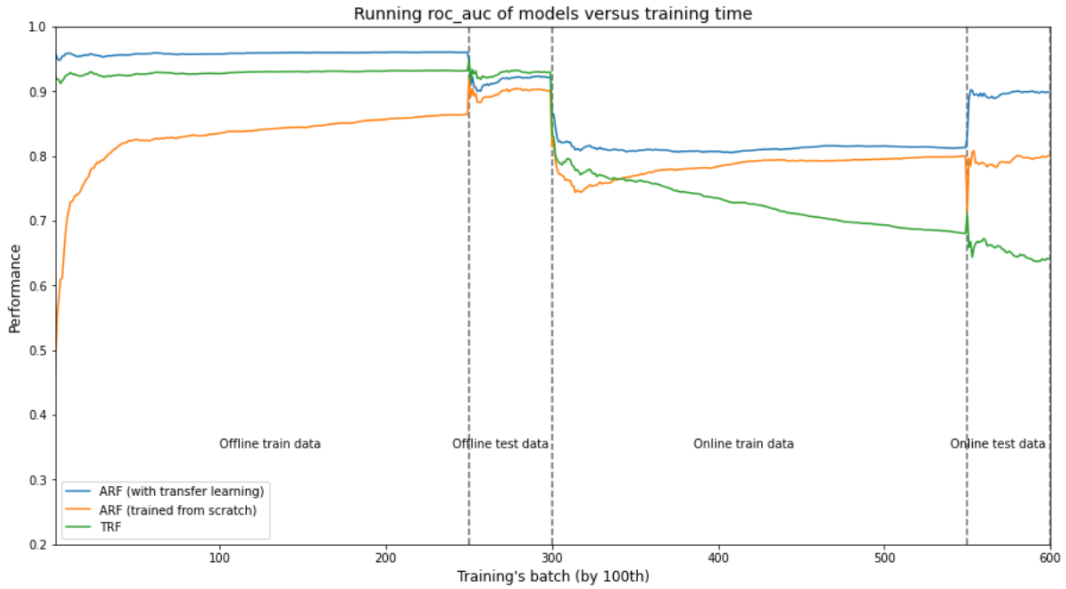


Figure 5.2.1.6: The performance of tree classifiers over time (AGRAWAL dataset)

Based on the graph above:

In the offline setting:

1. The performance of pre-trained ARF classifier was very close to TRF classifier since the training weights was obtained from the same model.
2. The TRF classifier was better than ARF classifier that was trained from scratch in terms of the train performance and generalized performance. This was because:
   a) TRF algorithm had the complete statistics for every attempt to split a leaf node, including the root node. This allowed the TRF algorithm to choose the best split every single time.
   b) ARF algorithm did not have the full statistics to decide on a split, nor the foreknowledge on how big the train data was. Thus, the algorithm was designed to split on limited number of samples using hyperparameters like grace period and Hoeffding bound. Upon splitting the root node, for each subsequent split attempt, the split decision was suboptimal since the first splitting was also suboptimal.

In the online setting:

1. TRF model's performance degraded over time when encountering drifted data. Manual model retraining was required to replace with a new TRF model. However, the performance for ARF models remained stable since model retraining was automatically taking place in the background. The algorithm would replace a base learner with the correspond background model if the base learner's performance degraded for some time.
2. The performance of ARF classifier that was trained from scratch slowly recovered when train on drifted sample, up to a point where the performance of both ARF models were similar at the end. This showed that ARF classifier that was trained from scratch required more training samples to get the same performance as pre-trained ARF classifier.
3. The ARF algorithm could generalize well to new data. It was because the online test performance of both ARF models was equivalent or higher than online train performance.

In short, the pre-trained ARF classifier generally had the best performance in both offline and online settings. Transfer learning from TRF classifier to ARF classifier combined the advantages of both TRF and ARF algorithms. The TRF algorithm provided batch learning that provided best split information to the pre-trained ARF classifier. While the ARF algorithm provided incremental learning that was robust to drift.

5.2.2 Application dataset: Lead scoring dataset

The performance of traditional random forest (TRF) classifier, pre-trained adaptive random forest (ARF) classifier, and adaptive random forest (ARF) classifier that was trained from scratch were evaluated on the lead scoring dataset. The model with the best overall performance would be deployed to the web service. 5219 training samples were used as the offline training set while 1305 samples were used as the offline test set. The information of the lead scoring dataset could be found in Section 4.1.1.

Offline Training Performance

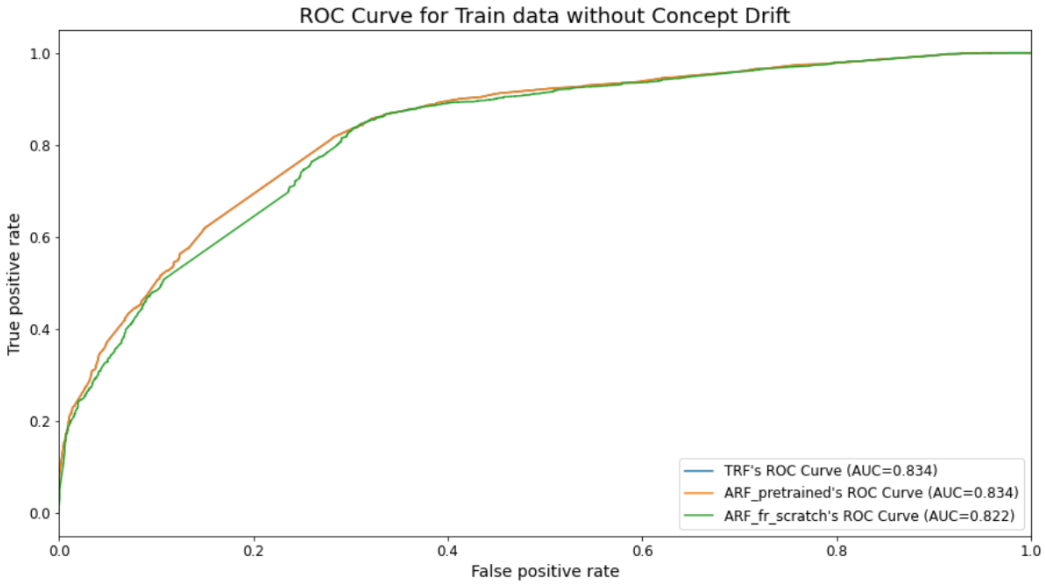


Figure 5.2.2.1: The offline training performance of tree classifier (Lead scoring dataset)

The graph above compared the train performance between TRF classifier and ARF classifier in the offline setting. The result exactly matched with the one conducted in the experiment. It could be inferred that the transfer learning process was successful as the ROC AUC scores of both TRF classifier and pretrained ARF classifier were the same. The performance was the same since the prediction value of every node was directly copied from the TRF classifier to the pretrained ARF classifier. On the other hand, the training performance of ARF classifier that was trained from scratch was similar to the performance pretrained ARF classifier with a negligible difference of 1%.

Combining the results that were observed from the last section, it could be inferred that the initial performance boost provided by transfer learning varied from one dataset to another. Regardless, transfer learning classifier algorithm was useful in situations where the performance was absolutely critical (e.g., data science competitions, banking system).

Offline Generalization Performance

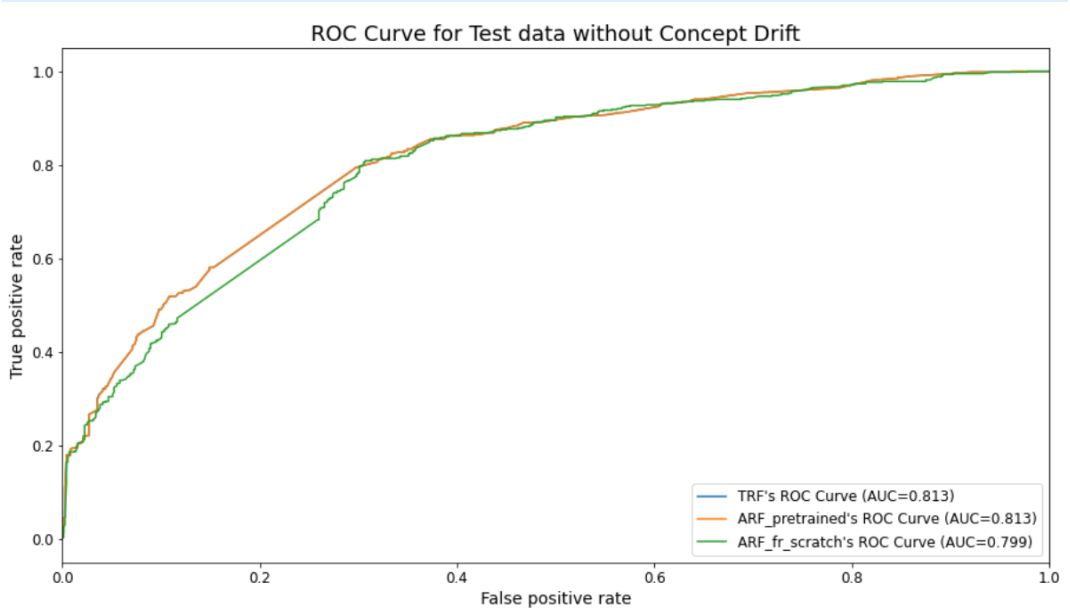


Figure 5.2.2.2: The offline generalization performance of tree classifier (Lead scoring dataset)

The graph above compared the generalization performance between TRF and ARF in the offline setting. The output again proved that transfer learning process was successful as the ROC AUC score of both TRF and pretrained ARF was the same. The generalization performance of ARF that was trained from scratch was as good as the pretrained ARF with a negligible ROC AUC score difference of 1%.

Model type	Diff ROC AUC in Offline set
TRF	0.834 - 0.813 = 0.021 (2.1%)
Pre-trained ARF	0.834 - 0.813 = 0.021 (2.1%)
ARF trained from scratch	0.822 - 0.799 = 0.023 (2.3%)

Table 5.2.2.1: The table that checked the overfitting issues for tree classifier (Lead scoring dataset)

The difference of R-squared in the offline set was calculated by subtracting the training R-squared score with the testing R-squared score for each set. Based on table above, the models did not suffer from serious overfitting issues even if the hyperparameters used were the same as the ones in the experiment.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Model	**Properties**																				
TRF	**total_nodes**	39	45	41	45	43	43	39	41	39	41	39	33	41	41	39	47	39	39	43	39
	max_height	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
ARF_fr_scratch	**total_nodes**	117	113	1	89	71	73	113	49	39	47	9	105	89	23	19	55	45	11	99	87
	max_height	17	13	0	12	10	10	18	10	7	12	4	14	13	5	6	8	10	5	13	11
ARF_pretrained	**total_nodes**	39	45	41	45	43	43	39	41	39	41	39	33	41	41	39	47	39	39	43	39
	max_height	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Figure 5.2.2.3: The table that showed the tree structure of the base learners of each tree ensemble (Lead scoring dataset)

The model selection could not be finalized since the test performance of pre-trained ARF classifier was similar to the ARF classifier that was trained from scratch. Hence, the tree structure of the base learners in both ARF models were analysed. Based on the table above, it could be observed that the pre-trained ARF's base learners' tree structures were more consistent. For ARF model that was trained from scratch, the tree height of its base learners varied from 0 to 17. It meant that some base learners only consisted of shallow leaf nodes or one root leaf node. Hence, ARF classifier that was trained from scratch yielded less stable predictions as compared to pre-trained ARF classifier. A consistent tree structure was important since it ensured that every base learner could contribute accurate and reliable prediction probabilities before averaged and summed into ensemble prediction probabilities. This suggested one of the reasons why the ARF classifier that was trained from scratch performed slightly worse as compared to other classifiers. Thus, the pre-trained ARF classifier was chosen.

5.2.3 Experimental dataset: California housing dataset

An experiment was conducted to evaluate the performance of adaptive random forest regressor and traditional random forest regressor with and without the influence of drift. No concept drift stream generator was used since none of the generated stream could train well with the traditional random forest regressor. Instead, the famous California housing dataset was used since the traditional random forest could be trained well with it. The target was the median house price, and the drift was manually induced by increasing the price by 25% due to economic factors like inflation or war.

To test the models' performance without drift, the first 20,640 samples were used to represent the offline data. To test the models' performance with drift, the last 20,640 samples to represent the online data. 4,128 samples out of 20,640 samples would be used to check model's generalisation error in both offline and online settings.

Offline Training Performance

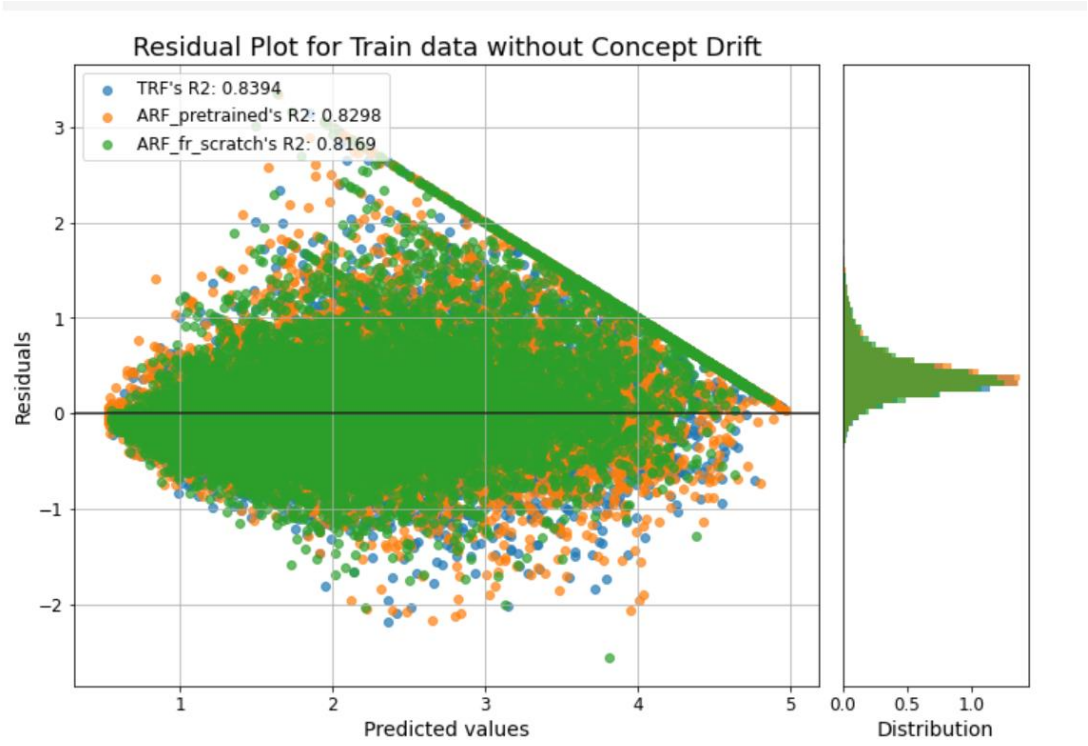


Figure 5.2.3.1: The offline training performance of tree regressors (California housing dataset)

The graph above compared the training performance between TRF regressor and ARF regressor in the offline setting. It could be inferred that the transfer learning process was successful as the R-squared value of both TRF regressor and pretrained ARF regressor differed no more than 1%. There was a slight difference in the performance since the leaf prediction mechanisms in the TRF regressor and ARF regressor were different. On the other hand, the training performance of ARF that was trained from scratch was slightly lower than pretrained ARF with the R-squared value difference of 1%.

Offline Generalization Performance

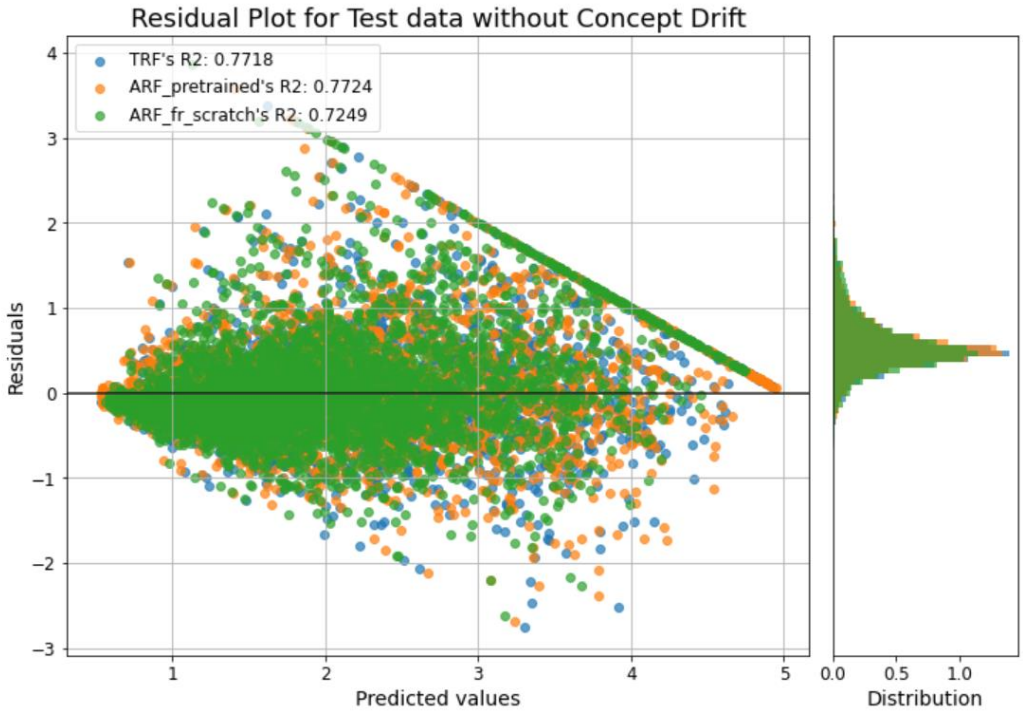


Figure 5.2.3.2: The offline generalization performance of tree regressors (California housing dataset)

The graph above compared the generalization performance between TRF regressor and ARF regressor in the offline setting. The graph again proved that transfer learning process was successful as the R-squared score of both TRF regressor and pretrained ARF regressor differed no more than 1%.

On the other hand, the generalization performance of ARF regressor that was trained from scratch was the lowest with a ROC AUC score difference of 5% as compared to pretrained ARF regressor. However, the result could not guarantee the superiority of the transfer learning algorithm for other datasets, which was discussed in the next section. In addition, it could be observed that all the models suffered from overfitting. In business scenarios, TRF and ARF should undergo thorough hyperparameter tunning to further reduce overfitting.

Online Training Performance

Model type	Offline Train R2	Online Train R2	Difference
TRF	0.8394	0.6493	-0.1901
Pre-trained ARF	0.8298	0.7950	-0.0348
ARF trained from scratch	0.8169	0.8042	-0.0127

Table 5.2.3.1: The comparison between offline and online training performance for tree regressors (California housing dataset)

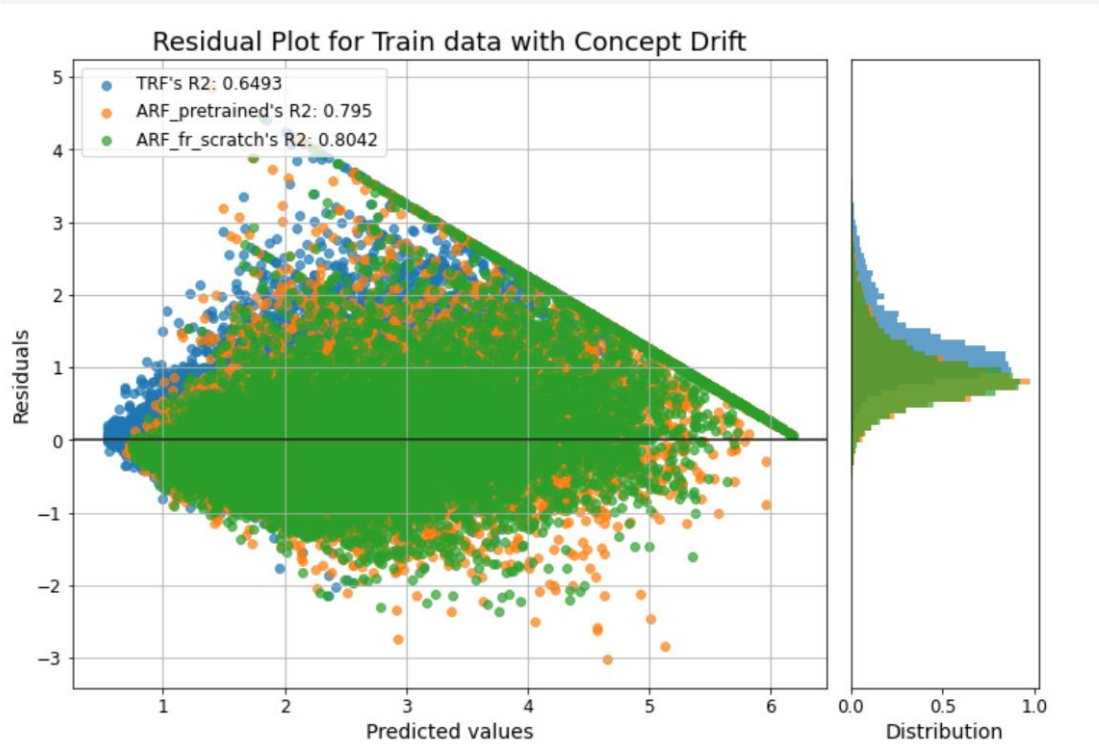


Figure 5.2.3.3: The online training performance of tree regressors (California housing dataset)

The graph above compared the training performance between TRF regressor and ARF regressor in the online setting. Due to drift, all the models' performance were affected. The train performance of TRF regressor had the most impact with a drop of 19% followed by both performance of ARF regressor with a drop of at most 4%. It could be deduced that ARF algorithm was more robust against drift as compared to TRF algorithm. On the other hand, the training performance of pre-trained ARF regressor was similar to ARF regressor that was trained from scratch with a slight R-squared value difference of 1%.

Online Testing Performance

Model type	Offline Test R2	Online Test R2	Difference
TRF	0.7718	0.5992	-0.1726
Pre-trained ARF	0.7724	0.7263	-0.0461
ARF trained from scratch	0.7249	0.7157	-0.0092

Table 5.2.3.2: The comparison between offline and online generalization performance for tree regressors (California housing dataset)

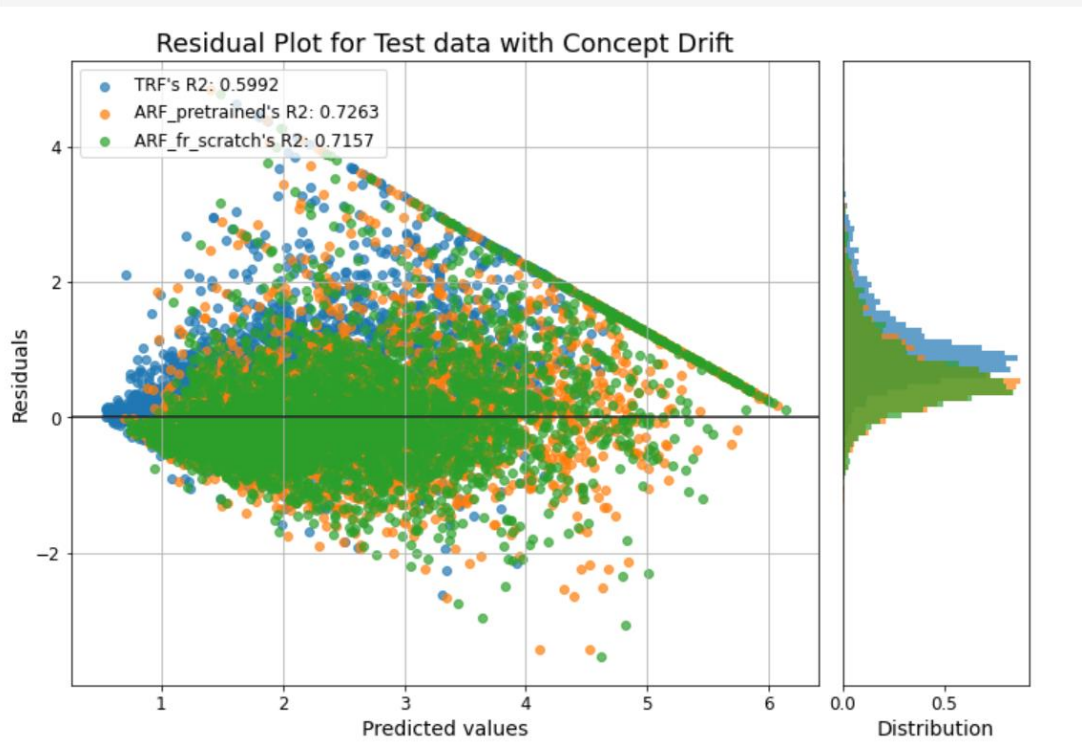


Figure 5.2.3.4: The online generalization performance of tree regressors (California housing dataset)

The table above compared the online generalization performance between TRF regressor and ARF regressor. The table above displayed the difference between offline and online generalization performance of each model. Due to drift, the online generalization performance of TRF dropped about 17% as compared to offline one. The pre-trained ARF's online generalization performance dropped about 5% while the performance for ARF that was trained from scratch dropped about 1%. It could be deduced that ARF algorithm was robust against drift but not TRF algorithm. On the other hand, the online generalization performance of the pre-trained ARF was similar to ARF that was trained from scratch with a difference of 1%.

Model type	Diff R2 in Offline set	Diff R2 in Online set
TRF	0.8394 - 0.7718 = 0.0676	0.6493 - 0.5992 = 0.0501
Pre-trained ARF	0.8298 - 0.7724 = 0.0574	0.7950 - 0.7263 = 0.0687
ARF trained from scratch	0.8169 - 0.7249 = 0.0920	0.8042 - 0.7157 = 0.0885

Table 5.2.3.3: The table that checked the overfitting issues for tree regressors (California housing dataset)

Visualizing Model Performance over Time

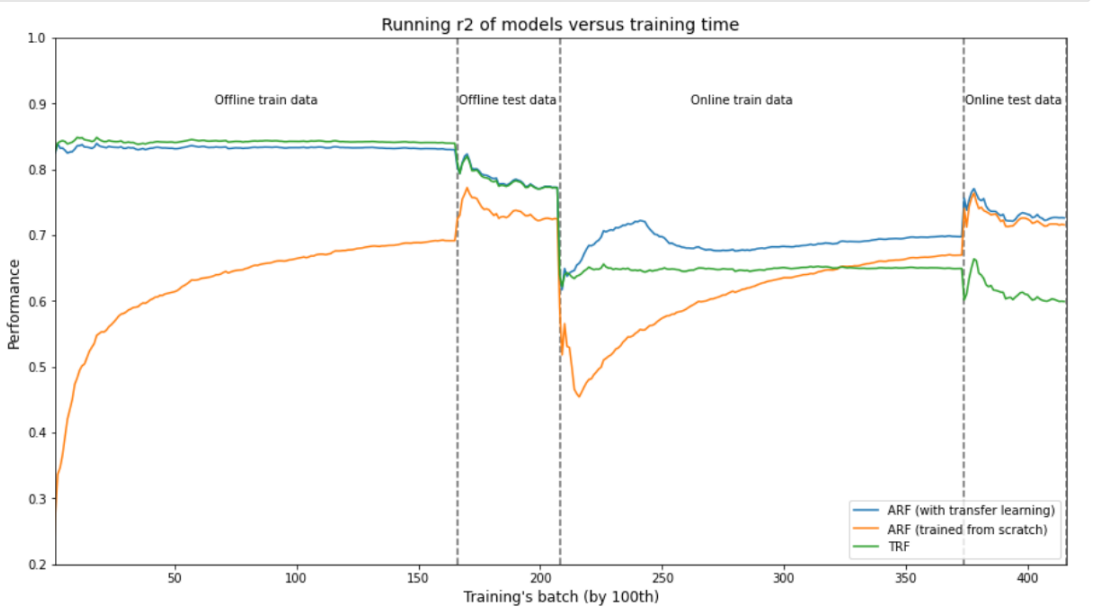


Figure 5.2.3.5: The performance of tree regressors over time (California housing dataset)

In the offline setting:

1. The performance of pre-trained ARF regressor was very close to TRF since the training weights was obtained from the same model.

In the online setting:

1. TRF model's performance rapidly dropped when encountering abruptly drifted data. Manual model retraining was required to replace with a new TRF model. However, the performance for both ARF models remained stable since model retraining was automatically taking place in the background. The algorithm would replace a base learner with the correspond background model if the base learner's performance degraded for some time.
2. The performance of ARF regressor that was trained from scratch slowly recovered when train on drifted sample, up to a point where the performance of both ARF models were similar at the end. This showed that ARF regressor that was trained from scratch required more training samples to get the same performance as pre-trained ARF regressor.

In short, the pre-trained ARF regressor generally had the best performance in both offline and online settings. Transfer learning from TRF regressor to ARF regressor combined the advantages of both TRF and ARF algorithms. The TRF algorithm provided batch learning that provided best split information to the pre-trained ARF regressor. While the ARF algorithm provided incremental learning that was robust to drift.

5.2.4 Application dataset: Car price dataset

The performance of traditional random forest (TRF) regressor, pre-trained adaptive random forest (ARF) regressor, and adaptive random forest (ARF) regressor that was trained from scratch were evaluated on the car price dataset. The model with the best overall performance would be deployed to the web service. 8533 training samples were used as the offline training set while 1094 samples were used as the offline test set. The information of the car price dataset could be found in Section 4.1.2.

Offline Training Performance

Figure 5.2.4.1: The offline training performance of tree regressor (Car price dataset)

The graph above compared the train performance between TRF regressor and ARF regressor in the offline setting. The result exactly matched with the one conducted in the experiment. It could be inferred that the transfer learning process was successful as the R-squared values of both TRF regressor and pretrained ARF regressor differed no more than 1%. There was a small difference in the performance since the leaf prediction mechanisms in the TRF and ARF were different. On the other hand, the training performance of ARF regressor that was trained from scratch was similar to the performance pretrained ARF regressor with a slight R-squared values difference of 1%.

Combining the results that were observed from the last section, it could be inferred that the initial performance boost provided by transfer learning varied from one dataset to another. Regardless, transfer learning regressor algorithm was useful in situations where the performance was absolutely critical (e.g., data science competitions, banking system).

Offline Generalization Performance

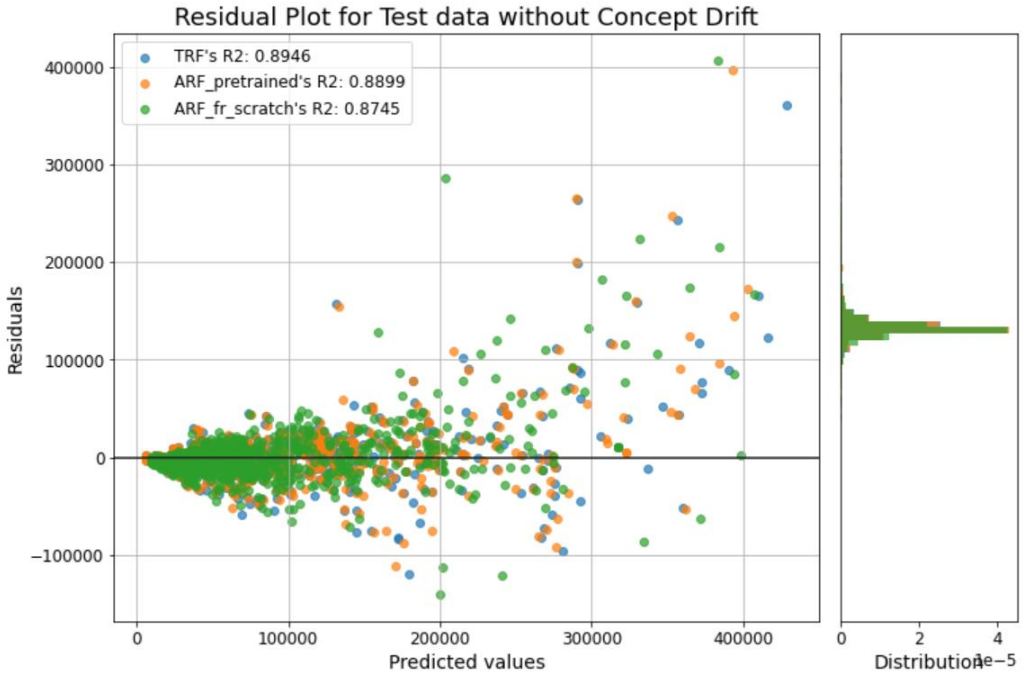


Figure 5.2.4.2: The offline generalization performance of tree regressor (Car price dataset)

The graph above compared the generalization performance between TRF regressor and ARF regressor in the offline setting. The graph again proved that transfer learning process was successful as the R-squared value of both TRF regressor and pretrained ARF regressor were similar with negligible difference. The generalization performance of ARF regressor that was trained from scratch was lower than the pretrained ARF with a R-squared value difference of 2%.

Model type	Diff R-squared in Offline set
TRF	0.8742 - 0.8946 = -0.0204 (-2.04%)
Pre-trained ARF	0.8717 - 0.8899 = -0.0182 (-1.82%)
ARF trained from scratch	0.8861 - 0.8745 = 0.0116 (1.16%)

Table 5.2.4.1: The table that checked the overfitting issues for tree regressor (Car price dataset)

The difference of R-squared in the offline set was calculated by subtracting the training R-squared score with the testing R-squared score for each set. Based on the table above, the models did not suffer from serious overfitting issues even if the hyperparameters used were the same as the ones in the experiment.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Model	Properties															
TRF	total_nodes	467	417	349	399	419	439	397	399	447	413	393	433	451	431	349
	max_height	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
ARF_fr_scratch	total_nodes	725	785	767	871	831	753	793	783	845	817	869	783	845	797	761
	max_height	19	17	24	21	17	19	21	22	20	22	23	21	20	26	24
ARF_pretrained	total_nodes	467	417	349	399	419	439	397	399	447	413	393	433	451	431	349
	max_height	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

Figure 5.2.4.3: The table that showed the tree structure of the base learners of each tree ensemble (Car price dataset)

The model selection could not be finalized since the test performance of pre-trained ARF regressor was similar to the ARF regressor that was trained from scratch. Hence, the tree structure of the base learners in both ARF models were analysed. Based on the table above, it could be observed that all the base learners of ARF regressor that was trained from scratch were deeper than the base learners of pretrained ARF. It meant that the pretrained ARF regressor could still perform as good as ARF regressor that was trained from scratch even if the tree height was fixed at 15. Hence, pretrained ARF regressor was chosen over ARF model regressor that was trained from scratch since it required lesser memory. All the tree heights were 15 since the max_depth parameter of TRF regressor, that was used to perform transfer learning, was set to 15. Hence, the pre-trained ARF regressor was chosen.

5.3 Model Tree Weight Extraction

A total of two tests were conducted to validate the correctness of the model tree weight extraction process.

Test 1: The performance of the tree SHAP explainer and tree SHAP loss explainer must be the same when the dictionary was passed in. Scikit-learn tree models were passed into the explainers to serve as the baseline performance. For example, the performance was compared between the tree SHAP explainer using the dictionary containing the tree weights of River adaptive random forest regressor and the tree SHAP explainer using the Scikit-learn random forest regressor. The result showed that the execution time was the same with negligible difference. The rest of the test results were found in jupyter_notebooks/FYP2_ARF_to_Dict_Validation.ipynb.

```
In [15]: # Compute SHAP value
         start = time.time()
         num_iter = 10

         print(f'Model used in initialization: {type(cp_arf_dict)}')
         for _ in range(num_iter):
             cp_tree_explainer = shap.TreeExplainer(
                 model = copy.deepcopy(cp_arf_dict),
                 feature_perturbation = 'interventional',
                 data = cp_X_test_subsample
             )
             _ = cp_tree_explainer.shap_values(cp_X_test)

         end = time.time()
         print(f'Average time taken to calculate SHAP value: {(end - start)/num_iter} seconds')

         Model used in initialization: <class 'dict'>
         Average time taken to calculate SHAP value: 2.697571587562561 seconds
```

Figure 5.3.1.1: Average time taken to calculate the SAHP value for dictionary containing the weights of ARF regressor (Car price dataset)

```
In [16]: # Compute SHAP value
         start = time.time()
         num_iter = 10

         print(f'Model used in initialization: {type(cp_trf_model)}')
         for _ in range(num_iter):
             cp_tree_explainer_tmp = shap.TreeExplainer(
                 model = cp_trf_model,
                 feature_perturbation = 'interventional',
                 data = cp_X_test_subsample
             )
             _ = cp_tree_explainer_tmp.shap_values(cp_X_test)

         end = time.time()
         print(f'Average time taken to calculate SHAP value: {(end - start)/num_iter} seconds')

         Model used in initialization: <class 'sklearn.ensemble._forest.RandomForestRegressor'>
         Average time taken to calculate SHAP value: 2.9476255655288695 seconds
```

Figure 5.3.1.2: Average time taken to calculate the SHAP value for Scikit-learn TRF regressor (Car price dataset)

Test 2: The position of all the nodes must be the same. It must be ensured that for every *i*th node, the split feature and split threshold must be the same between the two dictionaries. Note that the node's array index position for *i*th node was often not the same between the two dictionaries. Hence, the node index for decision tree and Hoeffding tree must be tracked separately to access the respective arrays of tree weights.

The results showed that the test was passed for both dictionary containing the weights of adaptive random forest regressor and dictionary containing the weights of adaptive random forest classifier.

```
df.head(10)
```

Out[21]:

	base_learner_no	dt_node_idx	ht_node_idx	same_feature?	same_threshold?
0	1	0	0	True	True
1	1	1	1	True	True
2	1	250	2	True	True
3	1	2	3	True	True
4	1	231	4	True	True
5	1	251	5	True	True
6	1	400	6	True	True
7	1	3	7	True	True
8	1	222	8	True	True
9	1	232	9	True	True

The number of problematic nodes should be zero as shown below.

In [22]:
```
problematic_nodes = df[(~df['same_feature?']) | (~df['same_threshold?'])]
print(f'Number of problematic nodes: {problematic_nodes.shape[0]}')
problematic_nodes
```

```
Number of problematic nodes: 0
```

Out[22]:

	base_learner_no	dt_node_idx	ht_node_idx	same_feature?	same_threshold?

Figure 5.3.1.3: Screenshot showing any problematics nodes in the dictionary containing the weights of ARF regressor (Car price dataset)

Out[41]:

	base_learner_no	dt_node_idx	ht_node_idx	same_feature?	same_threshold?
0	1	0	0	True	True
1	1	1	1	True	True
2	1	24	2	True	True
3	1	2	3	True	True
4	1	11	4	True	True
5	1	25	5	True	True
6	1	32	6	True	True
7	1	3	7	True	True
8	1	8	8	True	True
9	1	12	9	True	True

The number of problematic nodes should be zero as shown below.

In [42]:
```
problematic_nodes = df[(~df['same_feature?']) | (~df['same_threshold?']]
print(f'Number of problematic nodes: {problematic_nodes.shape[0]}')
problematic_nodes
```

```
Number of problematic nodes: 0
```

Out[42]:

	base_learner_no	dt_node_idx	ht_node_idx	same_feature?	same_threshold?

Figure 5.3.1.4: Screenshot showing any problematics nodes in the dictionary containing the weights of ARF classifier (Lead scoring dataset)

5.4 Tree SHAP Explainer

Setup

First, an experiment was conducted to prove that why tree SHAP explainer with tree_path_dependent approach could not be used.

To conduct the experiment using car price dataset, the adaptive random forest regressor was trained with test set and 10 model checkpoints were created. Then, the SHAP values were calculated for each model checkpoint. The difference of the SHAP values was computed by subtracting the ingested model predictions with the expected value. The difference should be close to 0 to indicate that the tree SHAP explainer was accurate in explaining model's predictions. In other words, a larger SHAP values difference indicated more inaccurate tree SHAP explainer.

Results

The result below showed the SHAP values difference for tree SHAP explainer that used tree_path_dependent approach. The tree SHAP explainer became more inaccurate as more samples were trained.

	Largest difference in SHAP values
0	0.0
1	5149.22402
2	10833.002394
3	16265.587504
4	21541.62672
5	28325.577186
6	33378.069441
7	38454.312109
8	43884.79866
9	49323.235785
10	54954.740067

Figure 5.4.1.1: The SHAP value differences for tree SHAP explainer that used tree path dependent approach across 10 different model checkpoints (Car price dataset)

To investigate the reason behind the inaccuracies in measuring the SHAP values, the node sample weight of each parent node and its child nodes were extracted into a table as shown below. By analysing the results below, it turned out that the

River Hoeffding tree regressor did not update node sample weight on the parent node. Tree SHAP explainer expected that for every parent node *i*, the node sample weight of the left and right child of parent node *i* must always be smaller than parent node *i* itself. Since the River only updated node sample weight at the leaf node, the expectation was broken. With the expectation broken, the SHAP values calculation would be inaccurate since the tree_path_dependent approach calculated SHAP values using node sample weight. Therefore, the River developer should give the option to update the node sample weight at the parent node to ensure compatibility and interoperability of River models with other Python libraries like SHAP.

	parent	left	right
53	74.0	115.0	12.0
72	79.0	123.0	8.0
82	64.0	91.0	7.0
155	73.0	106.0	18.0
165	86.0	110.0	12.0
...	...	...	...
8463	163.0	206.0	22.0
8471	72.0	122.0	18.0
8491	562.0	596.0	31.0
8555	166.0	184.0	7.0
8573	61.0	83.0	20.0

416 rows × 3 columns

Figure 5.4.1.2: The node sample weight of each parent node and its child nodes of the adaptive random forest regressor at the 10th checkpoint (Car price dataset)

To solve the issue, the tree SHAP explainer with the interventional approach was used instead of tree_path_dependent approach. The result below showed that the SHAP value differences were consistent in all the model checkpoints. The differences were negligible since the car price prediction values were in thousands and not sensitive to SHAP value differences that were less than 1.

	Largest difference in SHAP values
0	0.010849
1	0.014626
2	0.009889
3	0.010842
4	0.01027
5	0.008185
6	0.007507
7	0.008405
8	0.009215
9	0.007914
10	0.010849

Figure 5.4.1.3: The SHAP value differences for tree SHAP explainer that used interventional approach across 10 different model checkpoints (Car price dataset)

The same experiment was also conducted on the lead scoring dataset. The two results below again justified the use of tree SHAP explainer with interventional since the SHAP value differences were less than 1e-7. Note that the SHAP value differences for explainer that used tree_path_dependent approach were significant since the model outputted prediction probabilities ranging from 0 to 1.

	Largest difference in SHAP values
0	0.0
1	0.132005
2	0.137843
3	0.0855889
4	0.0440363
5	0.0764652
6	0.0929509
7	0.137198
8	0.148903
9	0.167896
10	0.213597

	Largest difference in SHAP values
0	0.0
1	2.41219e-08
2	2.38361e-08
3	2.46521e-08
4	1.92897e-08
5	1.93313e-08
6	1.8947e-08
7	2.07608e-08
8	1.78671e-08
9	2.11324e-08
10	2.40809e-08

Figure 5.4.1.4: The SHAP value differences for tree SHAP explainer that used tree path dependent approach (on the left) and interventional approach (on the right) across 10 different model checkpoints (Lead scoring dataset)

A total of three validation tests was conducted to further validate the tree explainer that used interventional approach to ensure that the visualization results were accurate. The three tests below were only valid if and only if the explainer used interventional approach and the "model_output" was set to "raw". There were no tests for the SHAP loss explainer that used interventional approach and the "model_output" was set to "log_loss". As of April 2022, the official documentation only provided an example of validation test on XGB gradient boosting classifier, but not classifiers like random forest classifier [20]. The validation test could not be used in both Scikit-learn random forest classifier and River adaptive random forest classifier since the SHAP loss value calculation was different for random forest models and gradient boosting models. Hence, the proposed three tests also served as a proxy test for the explainer that used interventional approach with the "model_output" set to "log_loss". Furthermore, the additivity check was only conducted when the explainer used tree_path_dependent approach or when the explainer used interventional approach and the "model_output" was set to "raw". In other words, there was no standardized test on validating the ingestion of tree classifiers into the explainers that used the interventional approach.

Test 1: Ensure that the expected value was equivalent to the average of all the model predictions on the provided background dataset.

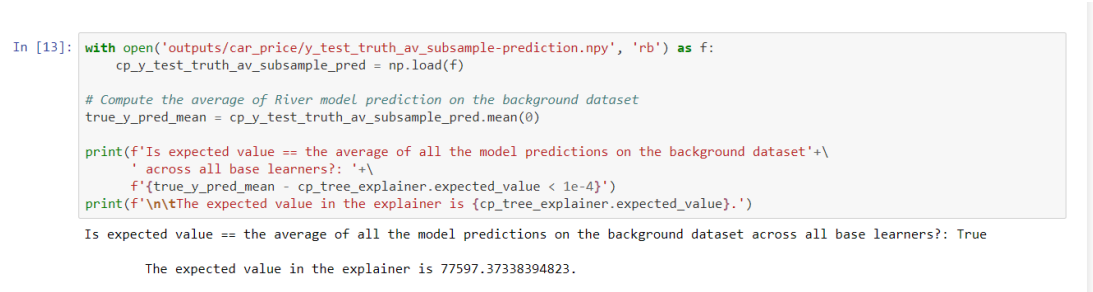


Figure 5.4.1.5: The first validation test of tree SHAP explainers (car price dataset)

In [13]:
```
with open('outputs/lead_scoring/y_test_truth_av_subsample-prediction.npy', 'rb') as f:
    ls_y_test_truth_av_subsample_pred = np.load(f)

# Compute the average of River model prediction on the background dataset
true_y_pred_mean = ls_y_test_truth_av_subsample_pred.mean(0)

biggest_diff = np.abs(true_y_pred_mean.sum(0) - ls_tree_explainer.expected_value.sum(0)).max()

print(f'Is expected value == the average of all the model predictions on the background dataset'+\
      ' across all base learners?: '+\
      f'{biggest_diff < 1e-4}')
print(f'\n\tThe expected value in the explainer is {ls_tree_explainer.expected_value}.')
```

```
Is expected value == the average of all the model predictions on the background dataset across all base
learners?: True

        The expected value in the explainer is [0.54051242 0.45948758].
```

Figure 5.4.1.6: The first validation test of tree SHAP explainers (lead scoring dataset)

Test 2: Ensure that the ingested SHAP model (a TreeEnsemble object) made the same predictions as the original model.

Test 2: Ensure that the ingested SHAP model (a TreeEnsemble object) makes the same predictions as the original model.

In [14]:
```
# Original model's prediction output
with open('outputs/car_price/y_test_truth_av-prediction.npy', 'rb') as f:
    cp_y_test_truth_av_pred = np.load(f)

true_y_pred = cp_y_test_truth_av_pred

# Ingested model's prediction output
y_pred_fr_ingested_model = cp_tree_explainer.model.predict(cp_X_test_truth_av)

largest_diff = np.abs(true_y_pred - y_pred_fr_ingested_model).max()

print(f'Does the ingested SHAP model make the same predictions as the original model?:'+\
      f' {largest_diff < 1e-4}')
print(f'\n\tThe largest difference is {largest_diff}.')
```

```
Does the ingested SHAP model make the same predictions as the original model?: True

        The largest difference is 8.731149137020111e-11.
```

Figure 5.4.1.7: The second validation test of tree SHAP explainers (car price dataset)

Test 2: Ensure that the ingested SHAP model (a TreeEnsemble object) makes the same prediction probabilities as the original model.

In [14]:
```
# Original model's prediction output
with open('outputs/lead_scoring/y_test_truth_av-prediction.npy', 'rb') as f:
    ls_y_test_truth_av_pred = np.load(f)

# Original model's prediction output
true_y_pred = ls_y_test_truth_av_pred

# Ingested model's prediction output
y_pred_fr_ingested_model = ls_tree_explainer.model.predict(ls_X_test_truth_av)

largest_diff = np.abs(true_y_pred - y_pred_fr_ingested_model).flatten().max()

print(f'Does the ingested SHAP model make the same predictions as the original model?: '+\
      f'{largest_diff < 1e-4}')
print(f'\n\tThe largest difference is {largest_diff}.')
```

```
Does the ingested SHAP model make the same predictions as the original model?: True

        The largest difference is 4.440892098500626e-16.
```

Figure 5.4.1.8: The second validation test of tree SHAP explainers (lead scoring dataset)

Test 3: For each sample, ensure that the sum of the SHAP values was equivalent to the difference between model output and the expected value. For the car price dataset, the result below showed that there was a noticeable difference (> 1e-4). However, since it was a regression problem, the difference was negligible since the prediction values were around thousands.

Test 3: For each sample, ensure that the sum of the SHAP values equals to the difference between model output and the expected value. The output below shows some noticeable differences (> 1e-4).

In [15]:
```
# Original model's prediction output
true_y_pred = cp_y_test_truth_av_pred

# Sum of the SHAP values and the expected value
y_pred_calulated_fr_shap_val = cp_tree_explainer.expected_value + cp_shap_values.sum(1)

largest_diff = np.abs(true_y_pred - y_pred_calulated_fr_shap_val).max()

print(f'model prediction output == expected value + sum of SHAP value of all features?: '+\
      f'{largest_diff < 1e-4}')
print(f'\n\tThe largest difference is {largest_diff}.')
```

```
model prediction output == expected value + sum of SHAP value of all features?: False

        The largest difference is 0.009681115567218512.
```

Figure 5.4.1.9: The third validation test of tree SHAP explainers (car price dataset)

Test 3: For each sample, ensure that the sum of the SHAP values equals to the difference between model output and the expected value.

In [15]:
```
# Original model's prediction output for positive class
# The shape is (number_of_samples, number_of_classes)
true_y_pred = ls_y_test_truth_av_pred

# Sum of the SHAP values and the expected value
# The expected value's shape is (number_of_classes, 1)
# The sum of SHAP values' shape is (number_of_classes, number_of_samples)
y_pred_calulated_fr_shap_val = \
ls_tree_explainer.expected_value[:, np.newaxis] + \
np.sum(np.array(ls_shap_values), axis=-1)

largest_diff = np.abs(true_y_pred - y_pred_calulated_fr_shap_val.T).flatten().max()

print(f'model prediction output == expected value + sum of SHAP value of all features?: '+\
      f'{largest_diff < 1e-4}')
print(f'\n\tThe largest difference is {largest_diff}.')
```

```
model prediction output == expected value + sum of SHAP value of all features?: True

        The largest difference is 2.3336163335052618e-08.
```

Figure 5.4.1.10: The third validation test of tree SHAP explainers (lead scoring dataset)

5.5 Monitoring Drift

5.5.1 SHAP loss

Setup

An experiment to was conducted to test the effectiveness of the model monitoring plot against drift or data error. The experiment setup was as shown below and the car price dataset was used. Due to the absence of drifted data, the existing car price test was manually modified to test the effectiveness of the model monitoring plot against drift or data error. The modifications were stated below:

1. To simulate data error for categorical features, all the values for label "brand_Proton" were converted to 0. This might happen when there was a logic error in persisting the car inventory record to the database.
2. To simulate data error for numerical features, all the values of the mileage were converted from kilometre to miles. This might happen when the users entered the mileage in miles in the web application.
3. To simulate gradual feature drift, the test records were separated into five stages. In each stage, the rows' "manufacture_year" field values were incremented by $[0, 2] * constant$. Each subsequent stage added the constant by 1.

Result

The graphs on the next page showed the monitoring plots on SHAP loss values of "manufacture_year", "mileage", and "brand_Proton" using Lundberg's SHAP loss monitoring function. It was expected that each of these plots had a vertical line since all the feature values have been modified. However, the Lundberg's function had failed to trigger the alarms for "manufacture_year" and "mileage" between the train data and the drifted test data.

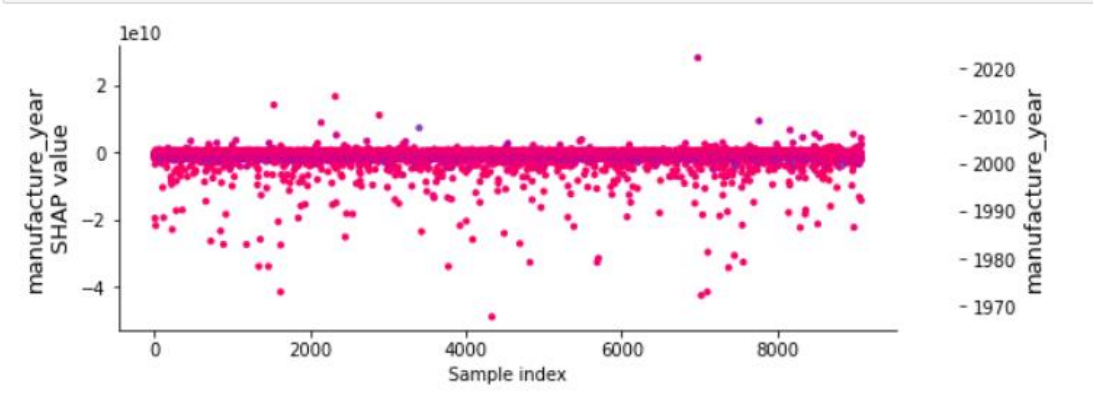


Figure 5.5.1.1: SHAP loss monitoring graph for "manufacture_year" plotted using Lundberg's function (Car price dataset)

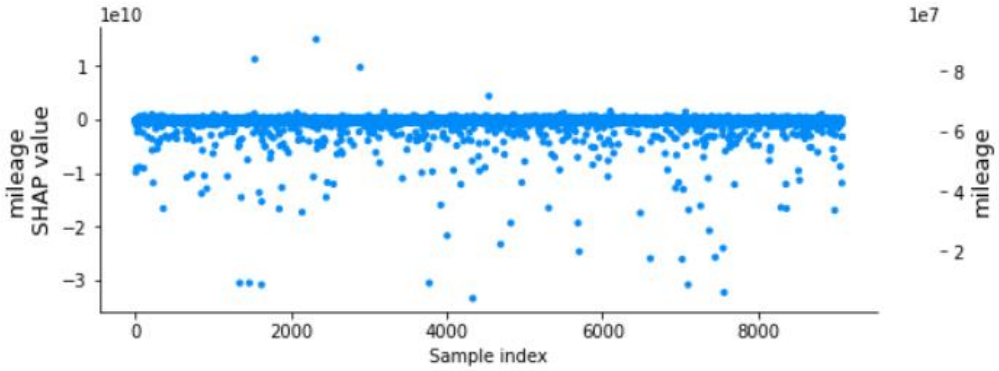


Figure 5.5.1.2: SHAP loss monitoring graph for "mileage" plotted using Lundberg's function (Car price dataset)

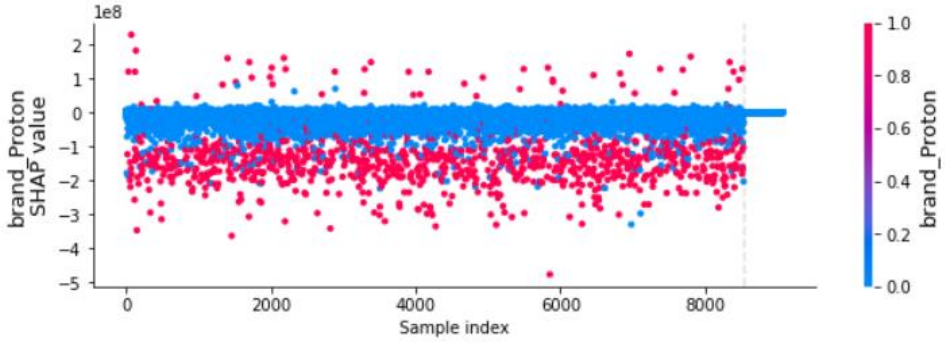


Figure 5.5.1.3: SHAP loss monitoring graph for "brand_Proton" plotted using Lundberg's function (Car price dataset)

Hence, the author enhanced the Lundberg's function to reduce the likelihood of having false negatives, even at the cost of higher likelihood of having false positives. The proposed function was validated with the same drifted car price test set.

The graphs on the next page showed the monitoring plots on SHAP loss values of "manufacture_year", "mileage", and "brand_Proton" using proposed SHAP loss monitoring function. The proposed function had successfully detected drift or error in all three modified features. In comparison with Lundberg's function, both functions had successfully raised alarm on the faulty "brand_Proton" feature values. Unlike Lundberg's function, the proposed function could raise alarms on the gradually drifted "manufacture_year" and the faulty "mileage" values using Welch's t-test.

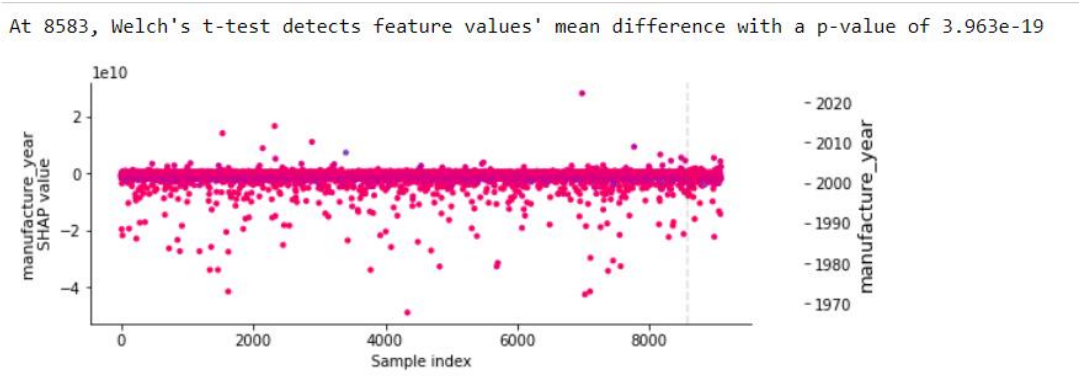


Figure 5.5.1.4: SHAP loss monitoring graph for "manufacture_year" plotted using proposed function (Car price dataset)

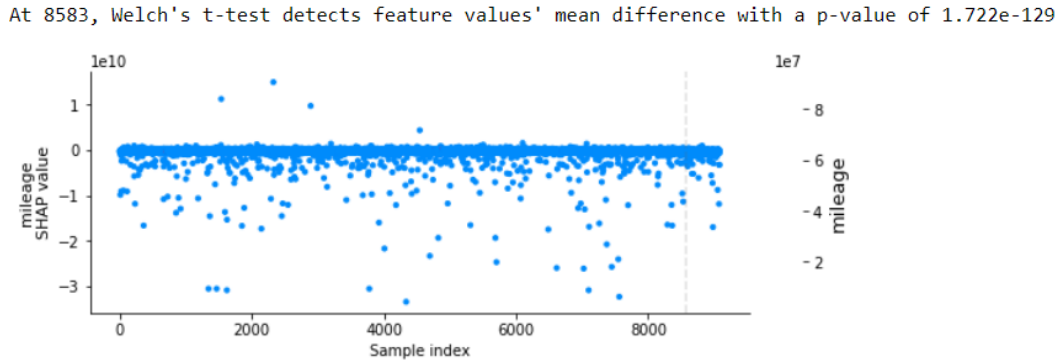


Figure 5.5.1.5: SHAP loss monitoring graph for "mileage" plotted using proposed function (Car price dataset)

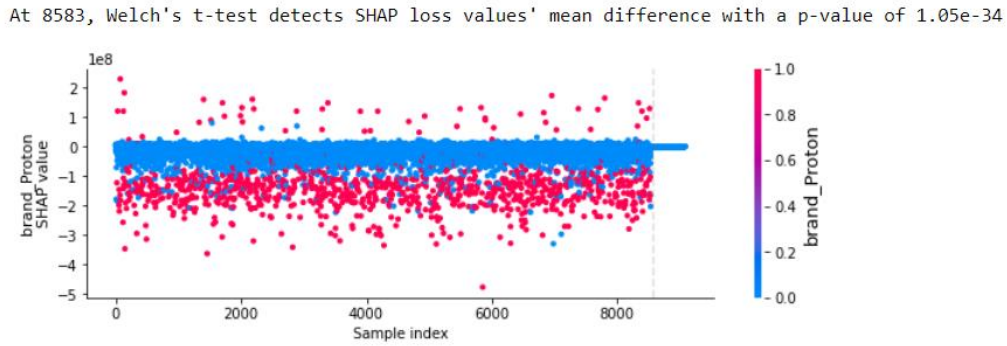


Figure 5.5.1.6: SHAP loss monitoring graph for "brand_Proton" plotted using proposed function (Car price dataset)

5.5.2 Statistical test

Population Stability Index

To check that the calculation was correct, an example of the calculation of PSI was given in the literature review. The same inputs were then used by the program to calculate the PSI values. The figure below showed that the calculation was accurate when compared with the manual calculation in the literature review.

```
The PSI value is 0.1342
```

Out[4]:

	Expected freq.	Observed freq.	Expected relative freq.	Observed relative freq.	Diff. in relative freqs.	log_e_diff_relative_freq	PSI value
0	3718	154	0.11	0.06	0.05	0.5604	0.0255
1	3795	172	0.11	0.07	0.04	0.4704	0.0191
2	3239	141	0.09	0.06	0.04	0.5107	0.0188
3	3537	195	0.10	0.08	0.02	0.2745	0.0066
4	3320	189	0.09	0.07	0.02	0.2424	0.0049
5	3596	301	0.10	0.12	-0.02	-0.1431	0.0023
6	3457	298	0.10	0.12	-0.02	-0.1725	0.0032
7	3515	369	0.10	0.14	-0.04	-0.3696	0.0165
8	3444	412	0.10	0.16	-0.06	-0.5002	0.0318
9	3503	317	0.10	0.12	-0.02	-0.2211	0.0055
Total	35124	2548	1.00	1.00	0.00	0.6518	0.1342

Figure 5.5.2.1: The PSI values calculated by the program

Chi-squared goodness of fit test

To check that the calculation was correct, an example of the calculation of the chi-squared goodness of fit test was given in the literature review. The same inputs were then used by the program to calculate the chi-squared values. The figure below showed that the calculation was accurate when compared with the manual calculation in the literature review.

```
The chi-square stats for "Test Feature" is 56.717 with the p-value of 4.8312019e-13.
```

Out[5]:

	Expected count	Expected count (Re-adjusted)	Observed count	Diff count	Squared Diff count	Chi2 value
Category 1	3718	212.32	301	-88.68	7864.35	37.0403
Category 2	3795	216.72	172	44.72	1999.52	9.2264
Category 3	3239	184.97	141	43.97	1932.94	10.4503
Total	10752	614.00	614	0.00	11796.81	56.7170

Figure 5.5.2.2: The chi-squared values calculated by the program

CHAPTER 6 SYSTEM EVALUATION

6.1 Evaluation on Web service

The web service was implemented as followed. The setup of the web service could be found in Section 4.4.1. All the API endpoints were implemented but two of the fourteen API endpoints had not been fully implemented. For the API endpoints '/car/global/review/performance' and '/lead/global/review/performance', the running metrics functionality was not implemented.

Lead scoring

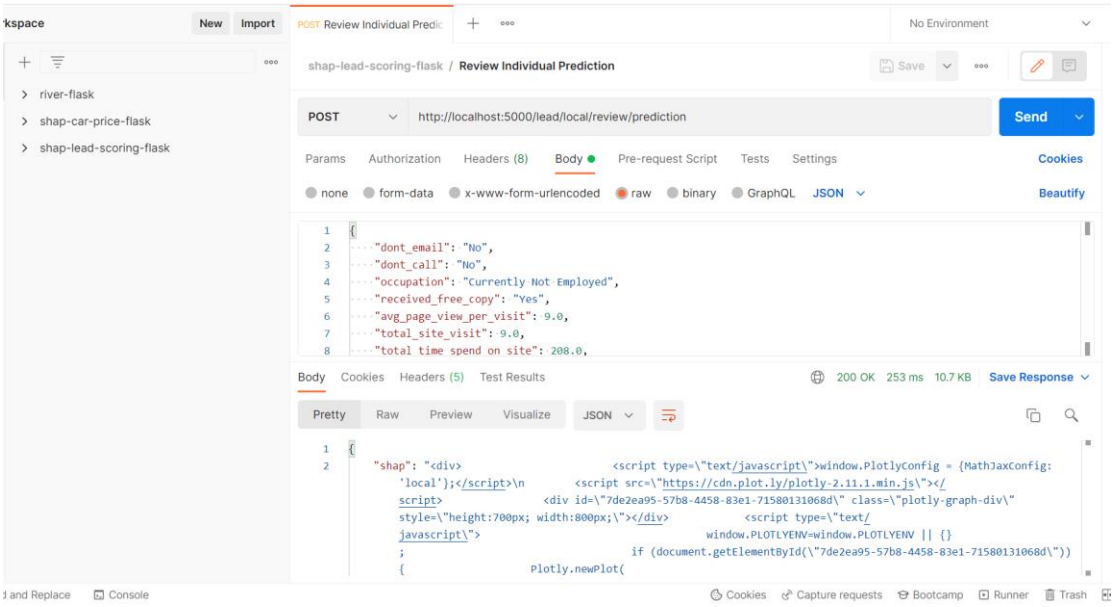


Figure 6.1.1: Screenshot of POST request to review individual prediction (Lead scoring dataset)

Based on the image above, a HTTP POST request was made to the http://localhost:5000/lead/local/review/prediction to review the individual lead scoring prediction. The input JSON could be found in web-service-test/lead-post-req-body.txt.

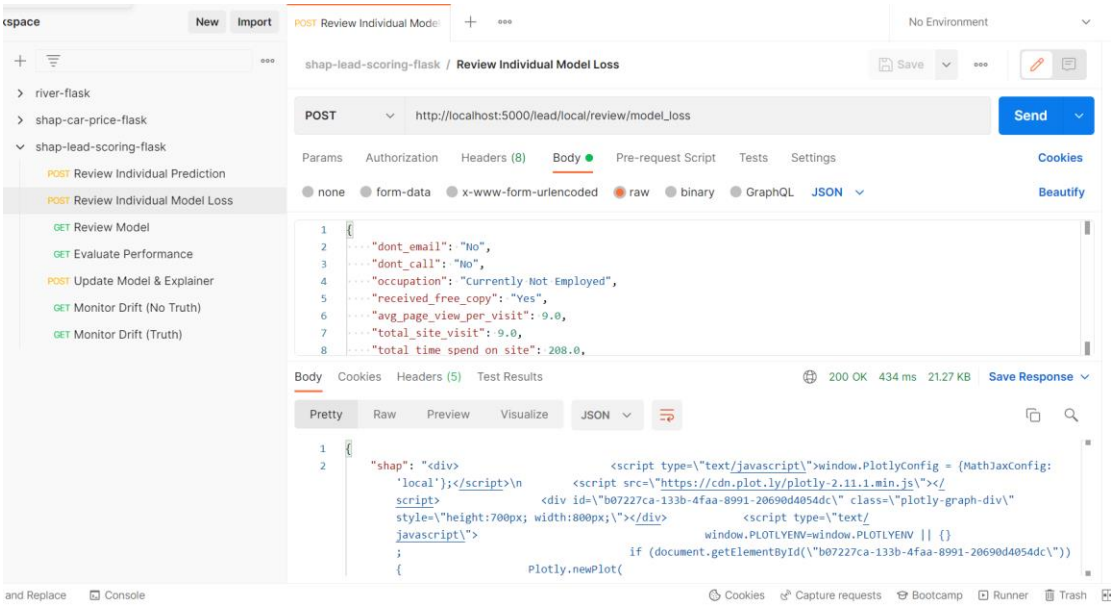


Figure 6.1.2: Screenshot of POST request to review individual model loss (Lead scoring dataset)

Based on the image above, a HTTP POST request was made to the http://localhost:5000/lead/local/review/model_loss to review the individual model loss on the lead record. The input JSON could be found in web-service-test/lead-post-req-body.txt.

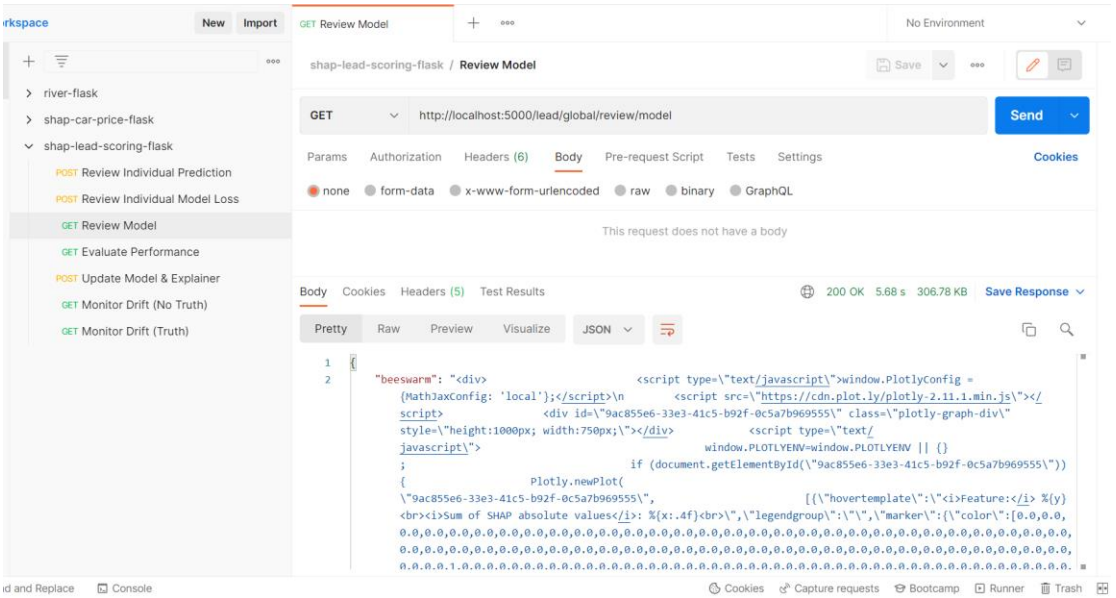


Figure 6.1.3: Screenshot of GET request to review model's overall prediction behaviour (Lead scoring dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/lead/global/review/model to review the lead scoring model's overall prediction behaviour.

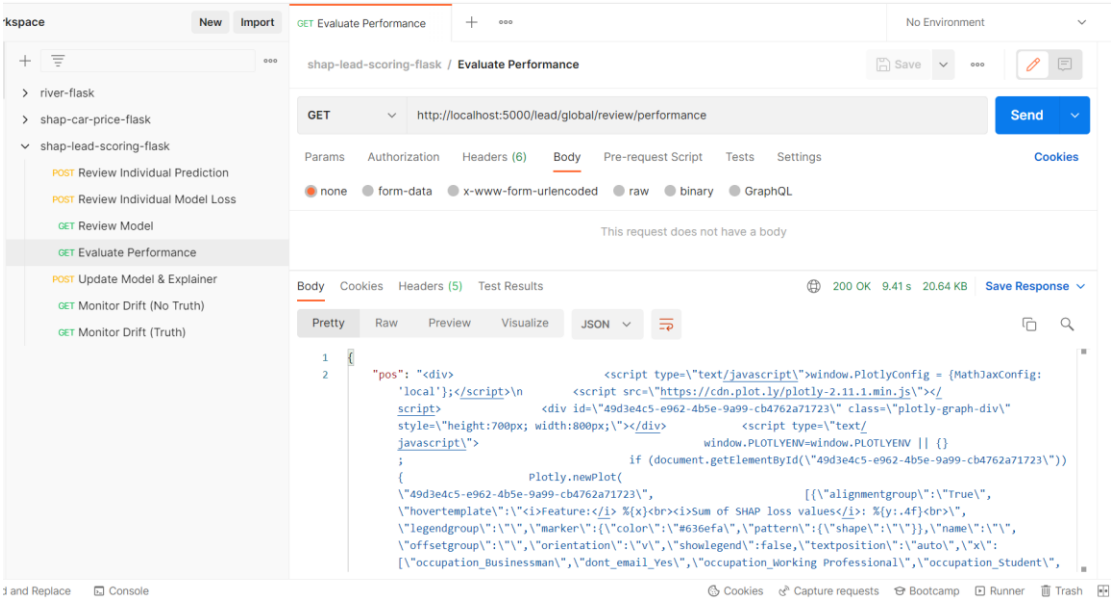


Figure 6.1.4: Screenshot of GET request to evaluate model's performance (Lead scoring dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/lead/global/review/performance to evaluate the lead scoring model's performance.

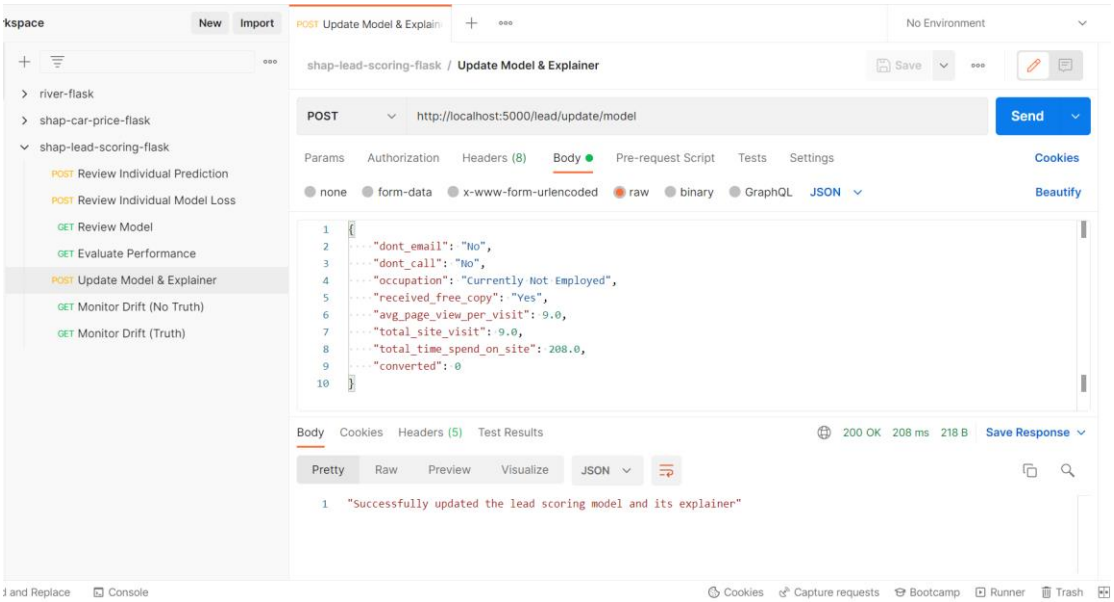


Figure 6.1.5: Screenshot of POST request to incrementally train model and update explainers (Lead scoring dataset)

Based on the image above, a HTTP POST request was made to the http://localhost:5000/lead/update/model to incrementally train the lead scoring model and update the tree SHAP explainer and tree SHAP loss explainer. The input JSON could be found in web-service-test/lead-post-req-body.txt.

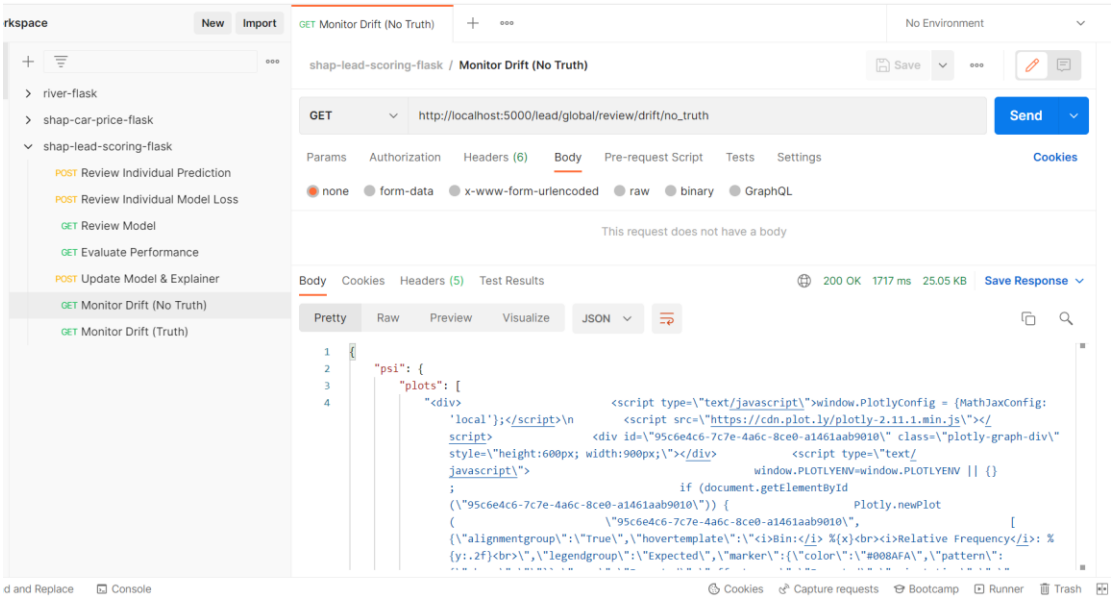


Figure 6.1.6: Screenshot of GET request to monitor drift on records that had no truth (Lead scoring dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/lead/global/review/drift/no_truth to monitor drift on lead records that had no truth.

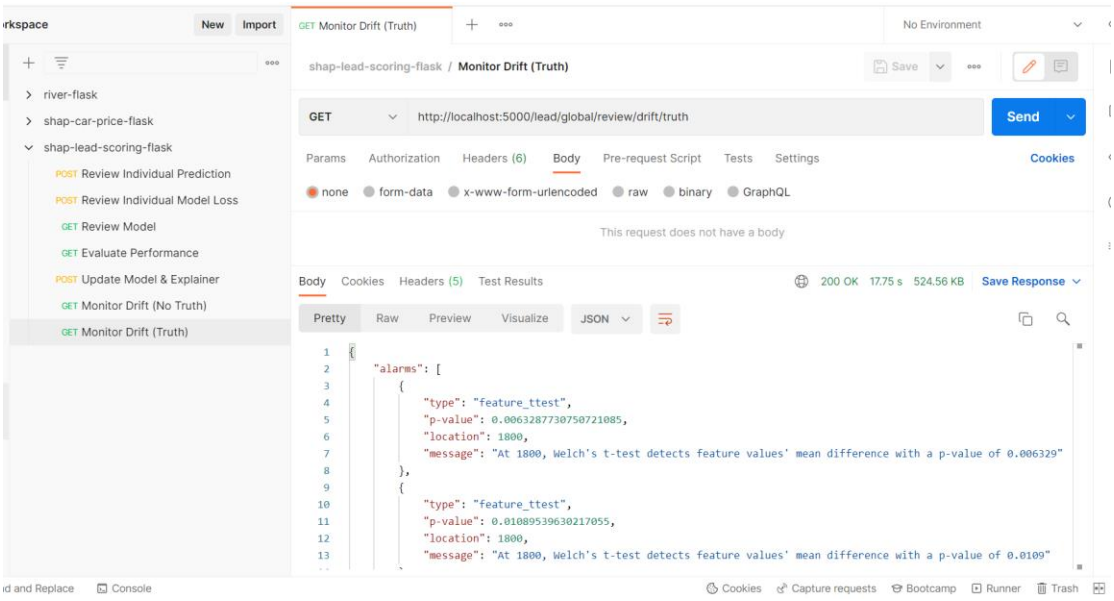


Figure 6.1.7: Screenshot of GET request to monitor drift on records that had truth (Lead scoring dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/lead/global/review/drift/truth to monitor drift on lead records that had truth.

Car price

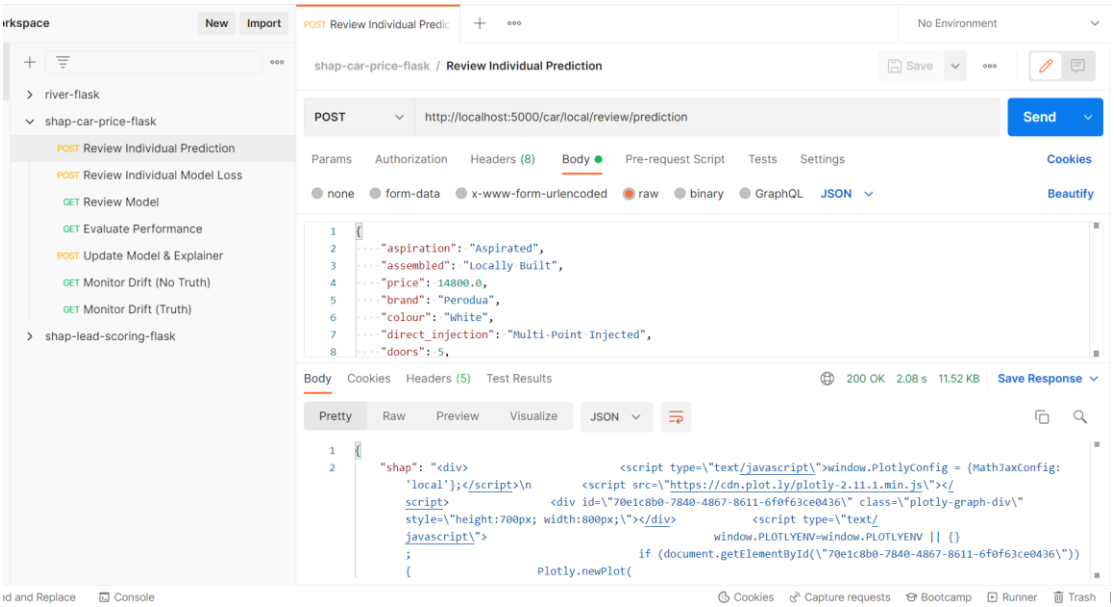


Figure 6.1.8: Screenshot of POST request to review individual prediction (Car price dataset)

Based on the image above, a HTTP POST request was made to the http://localhost:5000/car/local/review/prediction to review the individual car price prediction. The input JSON could be found in web-service-test/car-price-post-req-body.txt.

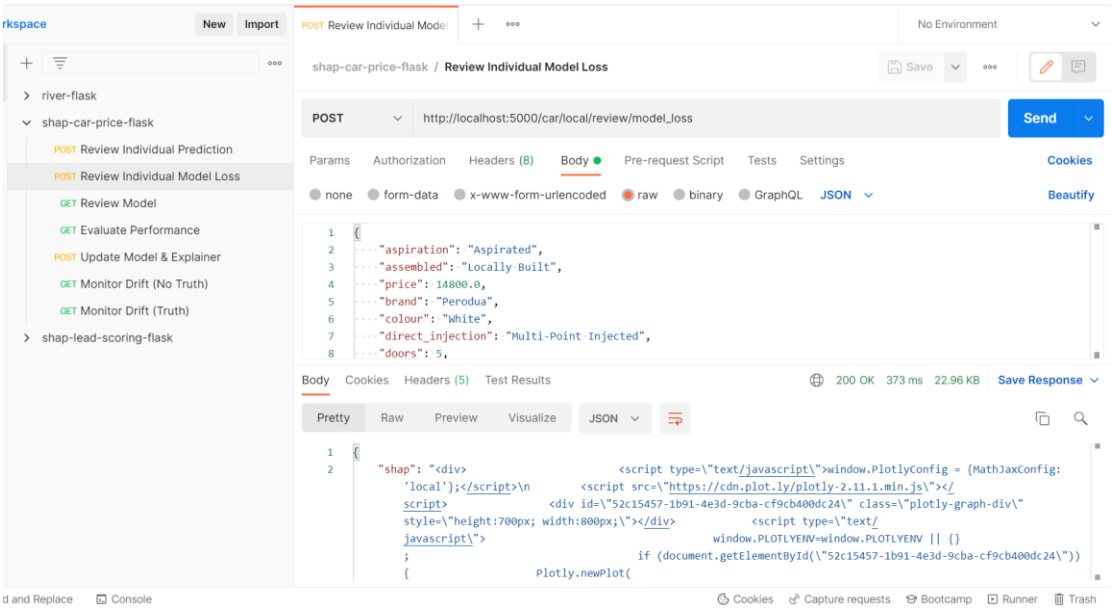


Figure 6.1.9: Screenshot of POST request to review individual model loss (Car price dataset)

Based on the image above, a HTTP POST request was made to the http://localhost:5000/car/local/review/model_loss to review the individual model loss on the car inventory record. The input JSON could be found in web-service-test/car-price-post-req-body.txt.

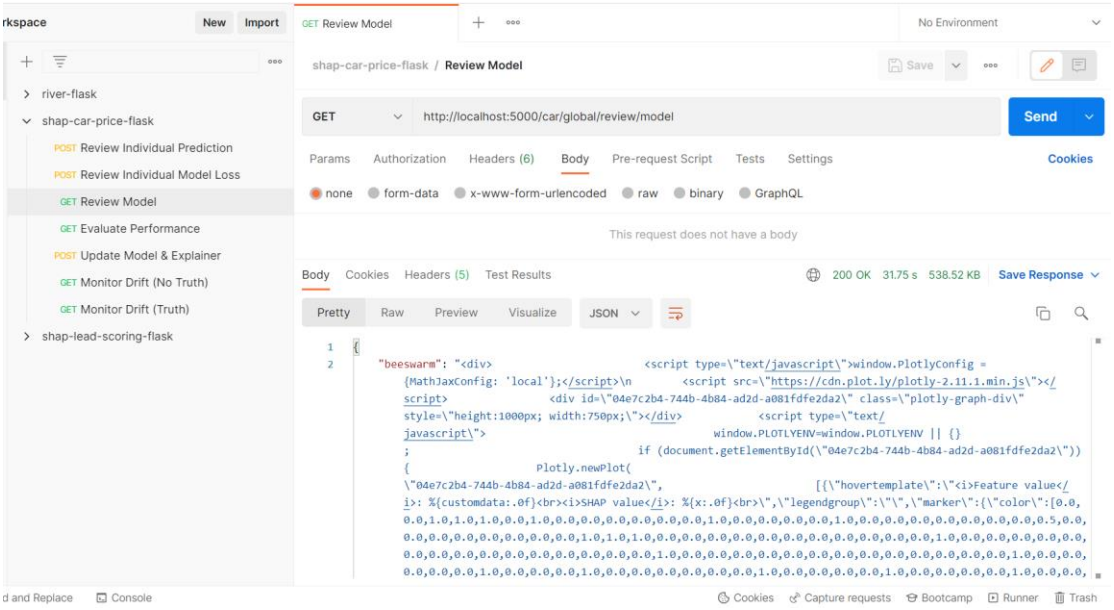


Figure 6.1.10: Screenshot of GET request to review model's overall prediction behaviour (Car price dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/car/global/review/model to review the car price model's overall prediction behaviour.

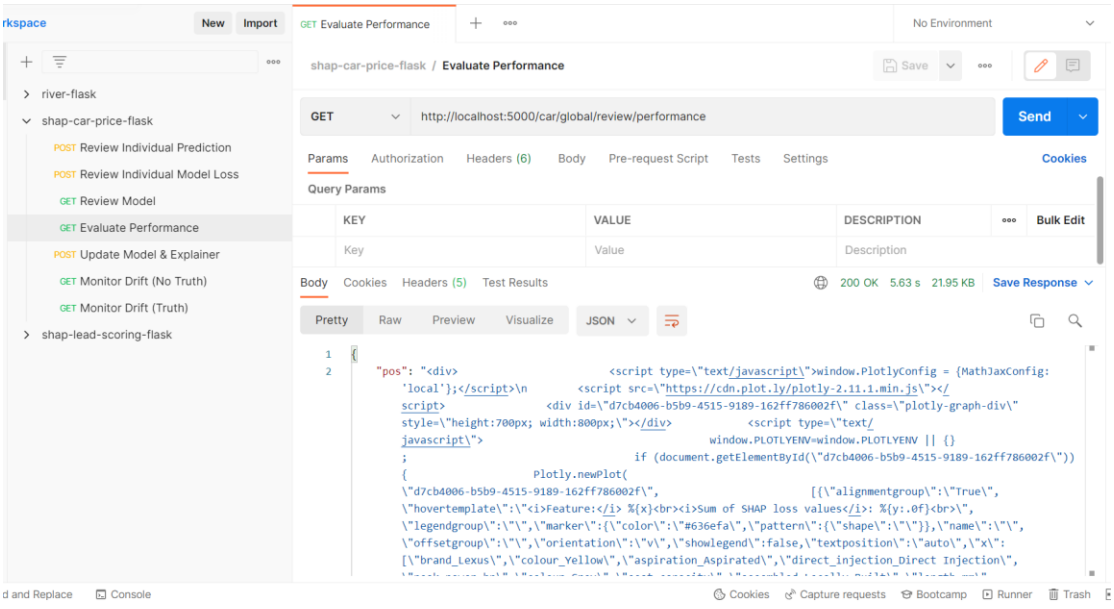


Figure 6.1.11: Screenshot of GET request to evaluate model's performance (Car price dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/car/global/review/performance to evaluate the car price model's performance.

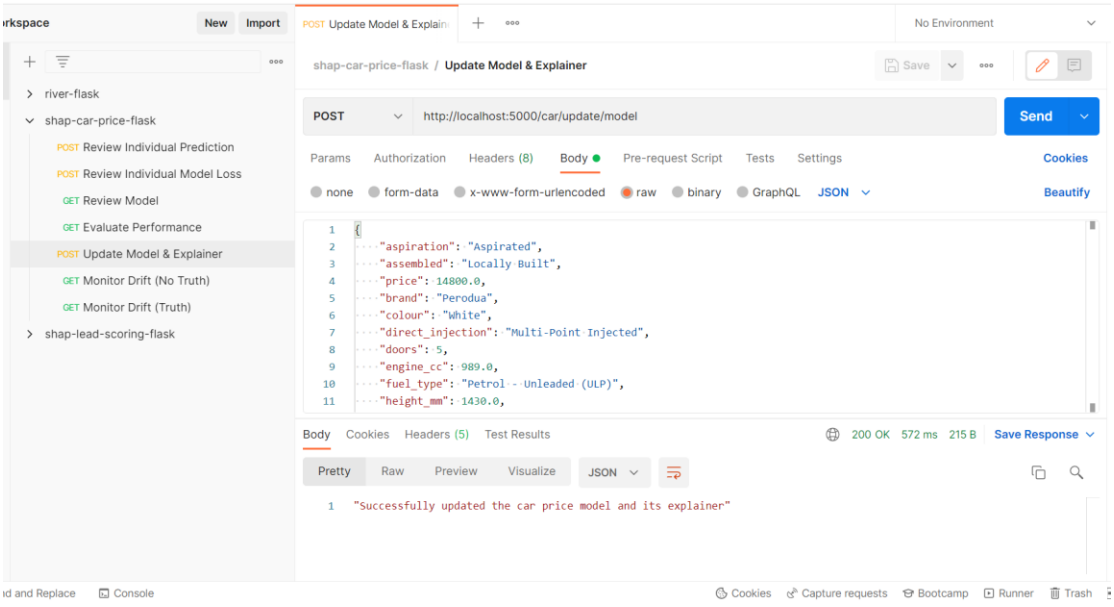


Figure 6.1.12: Screenshot of POST request to incrementally train model and update explainers (Car price dataset)

Based on the image above, a HTTP POST request was made to the http://localhost:5000/car/update/model to incrementally train the car price model and update the tree SHAP explainer and tree SHAP loss explainer. The input JSON could be found in web-service-test/car-price-post-req-body.txt.

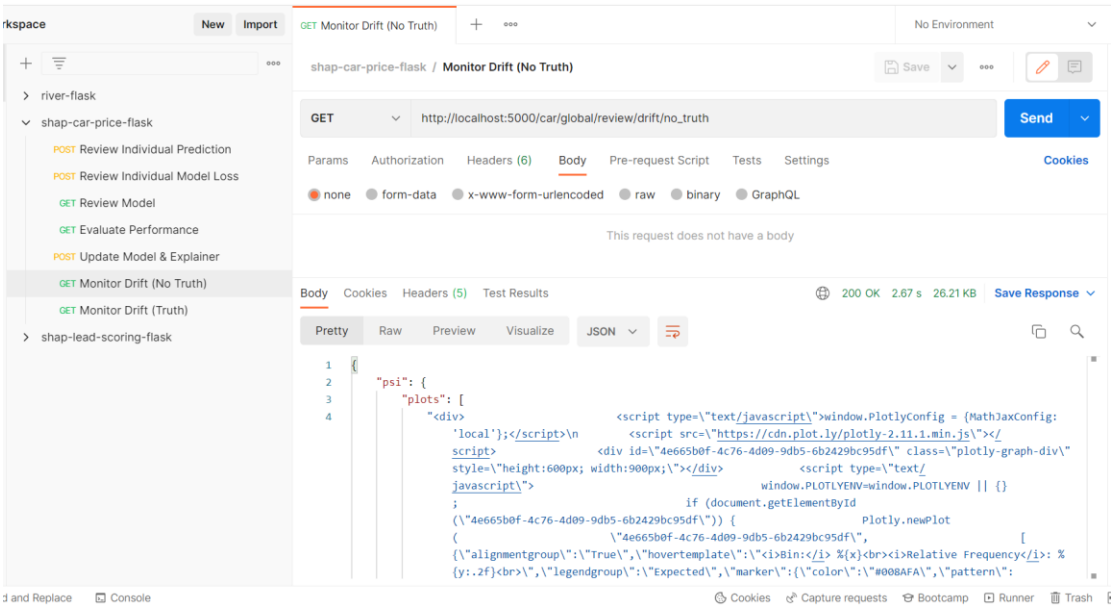


Figure 6.1.13: Screenshot of GET request to monitor drift on records that had no truth (Car price dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/car/global/review/drift/no_truth to monitor drift on car inventory records that had no truth.

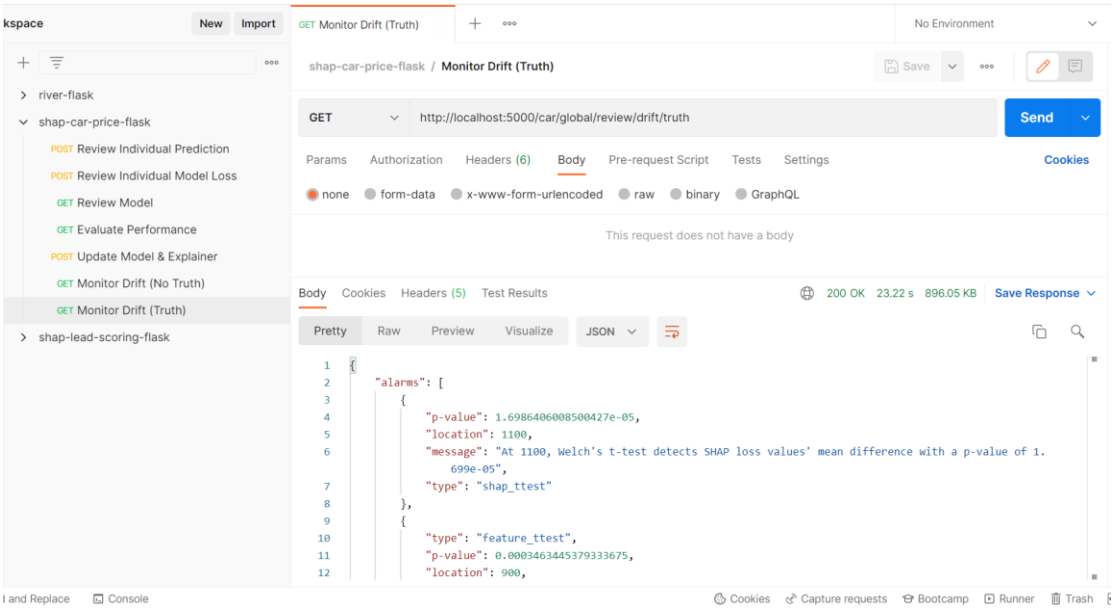


Figure 6.1.14: Screenshot of GET request to monitor drift on records that had truth (Car price dataset)

Based on the image above, a HTTP GET request was made to the http://localhost:5000/lead/global/review/drift/truth to monitor drift on lead records that had truth.

6.2 Evaluation on Web application

The controller actions were implemented to provide CRUD operations to both lead management module and the car inventory management module. Unfortunately, the author did not implement the controller actions to connect the web service functionalities to the web application. Regardless, the UI logic of these controller actions had been documented in the activity diagrams.

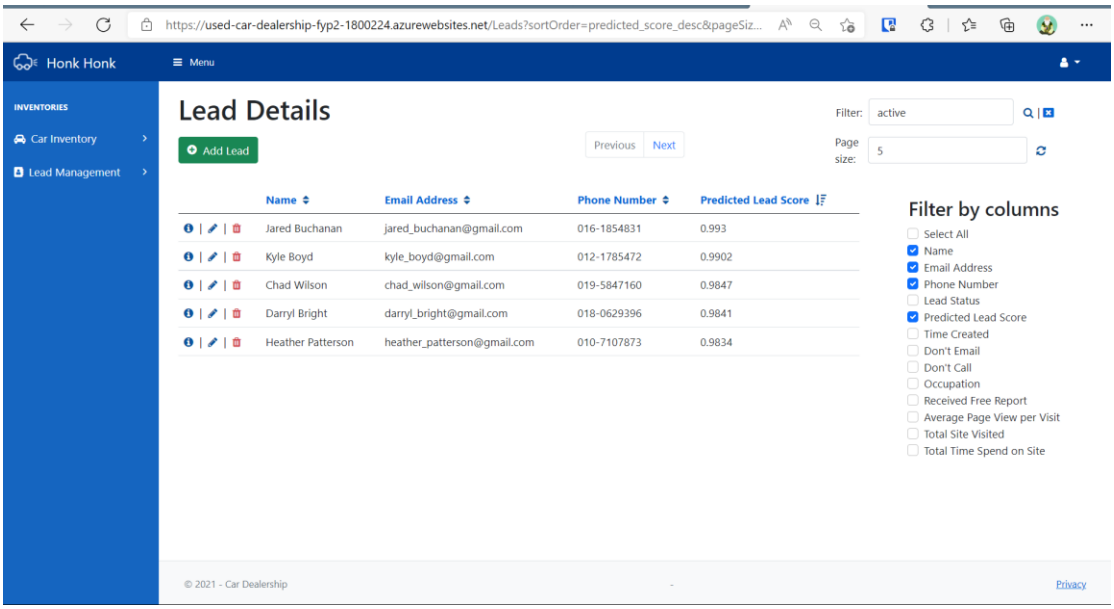


Figure 6.2.1: UI screenshot of the lead management page

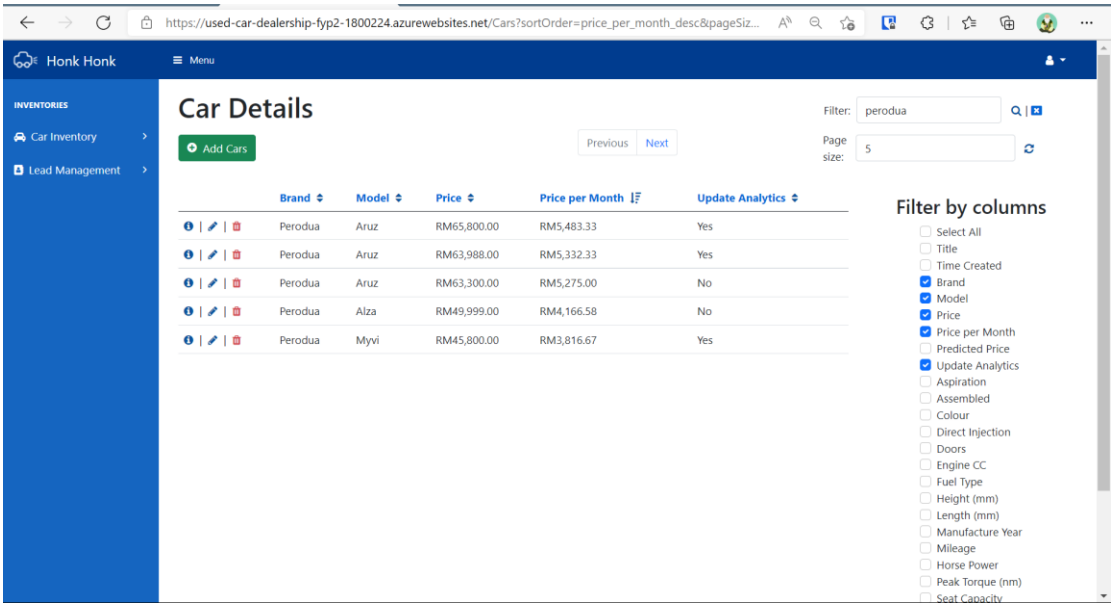


Figure 6.2.2: UI screenshot of the car inventory management page

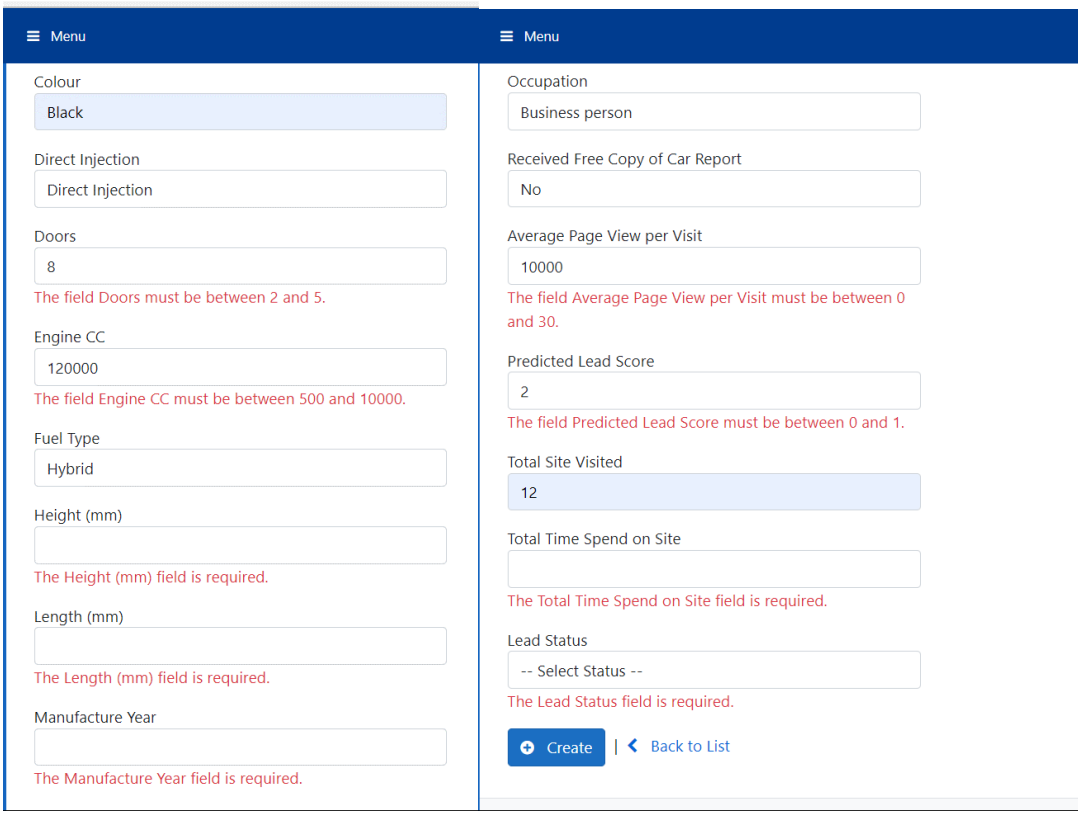


Figure 6.2.3: UI screenshot of the input validation functionality

Besides the CRUD functionalities, the input validations were also implemented to ensure that the values persisted to the database were not faulty so that the models would not get corrupted from training with erroneous data.

6.3 Project Challenges

Implementation Issues and Challenges

The main implementation issues and challenges were found at the web service. When proposing the web service in FYP I, it was initially assumed that the River library would worked with SHAP library without any technical complications. As a result, more time could be spent on implementing functionalities on the web application. However, that was not the case.

The first implementation issue was that the tree SHAP explainer from SHAP library did not directly support the weight extraction of adaptive random forest regressor and classifier from River API. The tree weights must be manually extracted into a dictionary since tree SHAP explainer still could accept Python dictionary to support any tree models. As a result, it was required to spend some time at reading the source code of both libraries to ensure that the model was correctly passed in into the tree SHAP explainer. Several new issues had emerged when implementing the extraction process. First, the split conditions were different for both libraries. Second, both River Hoeffding tree classifier and regressor did not update node sample weight on the parent node.

The second implementation issue was that the hyperparameter settings of both River adaptive random forest regressor and classifier must be limited to a certain value in order to ensure that the SHAP values' calculations were accurate. For example, the River adaptive random forest classifier's hyperparameter "disable_weighted_vote" must be set to False since the SHAP tree explainers could not perform weighted vote prediction. These specific technical details were obviously not in the River and SHAP official documentation and the author had to painstakingly identify the real issue. It was because of these two main implementation issues, the author had to spend time on conducting extensive experiments and validation testing to locate the problems.

The third implementation issue was the incompatibility of SHAP library with the River library. As of April 2021, SHAP library and River library had conflicting dependencies on NumPy library. SHAP library required NumPy version of 1.21.5 while River library required NumPy version of 1.22.3. The import of both libraries

would fail if the respective requirements were not met. The image below showed that the River library could not be loaded when the NumPy version was 1.21.5.

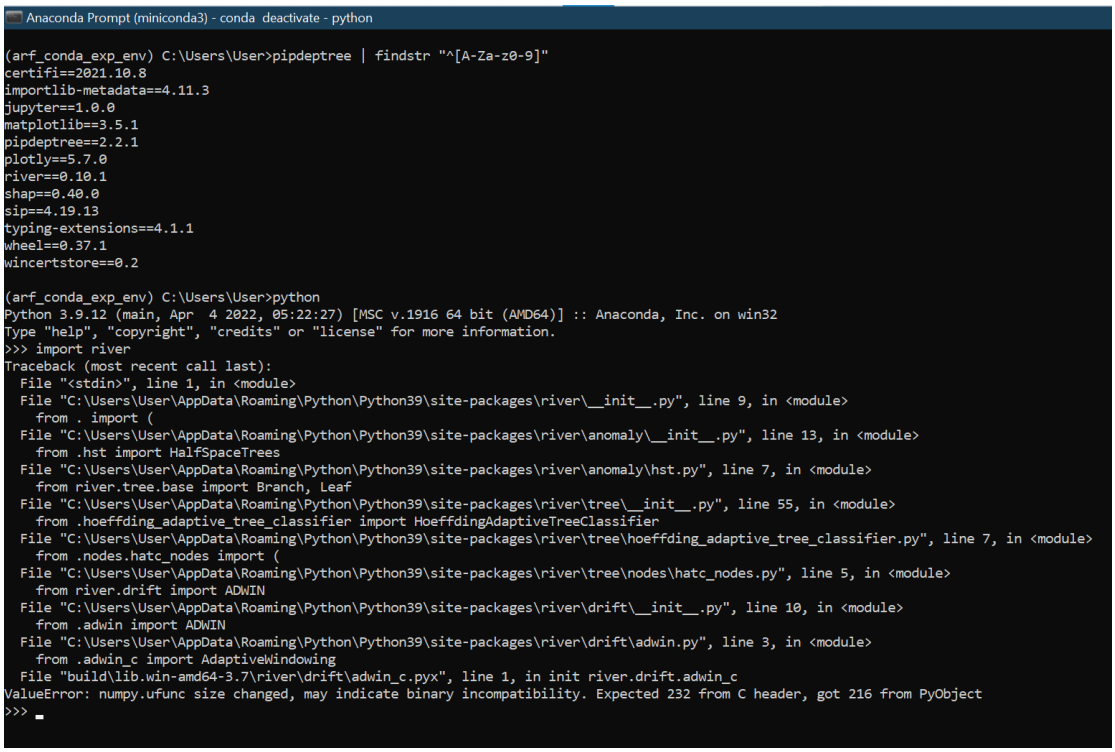


Figure 6.3.1: Screenshot showing that the River library could not be loaded in the "arf_conda_evp_env" conda environment

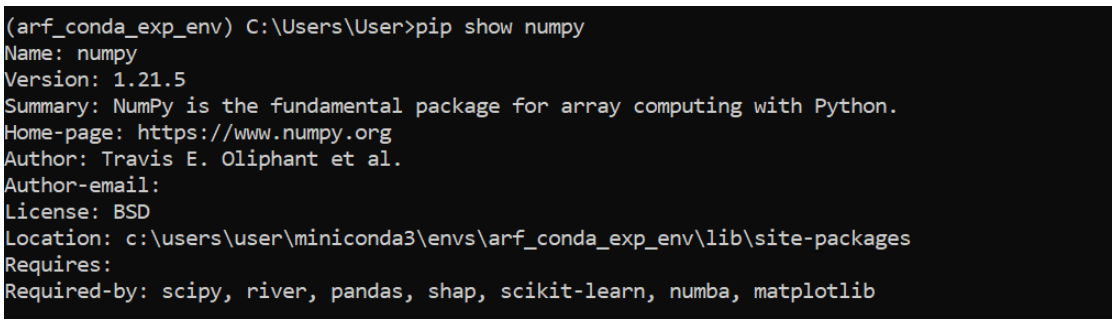


Figure 6.3.2: Screenshot showing the version of NumPy in the "arf_conda_evp_env" conda environment

The issues would be easily resolved when running notebooks since two separate conda environment could be created as shown in the image above. However, the deployment of the web service became more complicated.

One simple solution was to create two entirely separated Docker containers and deployed to the Azure Container Registry. Then, two Azure Container instances

would be created to separately pull the images, respectively. However, the communication would be cut off since the public IP addresses of the Azure Container instances that ran the River web service could change due to reasons like restarting. Manual update of IP address was required to ensure that the SHAP web service could communicate with River web service using the right URL. Hence, the author used Docker compose to combine the two Docker images into a multi-container web service. Unfortunately, the author had failed to deploy the multi-container web service to the cloud due to lack of time.

6.4 Objectives Evaluation

For the first and second objective, the lead management module and the inventory management module were implemented and the web service that provided the predictive analytics functionalities was implemented. However, the modules could not directly use the capabilities since the API calls were not implemented. Regardless, the activity diagram had documented the UI logics to make the API calls.

For the third objective, the web service had implemented the online AI learning and monitoring system. Adaptive random forest regressor and classifier were deployed so that the models could automatically detect drifts and retrained itself. The monitoring functionality was also implemented for two scenarios. The two scenarios were a scenario where the truth was available and another scenario where the truth was not available. Experiments had also been conducted to validate the techniques used to detect and monitor the drifts, which were the tree SHAP loss monitoring and statistical tests.

For the fourth objective, the web service had implemented the explainable AI functionalities to provide useful insights on individual predictions, individual model loss, model's overall prediction behaviour, and model's performance by plotting graphs. The report also discussed on how to interpret these plots so that the inventory managers, sales employees, and data scientists could know what insights could be derived.

CHAPTER 7 CONCLUSION

7.1 Conclusion

In conclusion, the web service had been successful implemented using SHAP library and River Python library. In order to implement the full proof web service, three tests were used to validate both transfer learning algorithms; two tests were used to validate the tree weight extraction function; three tests were used to validate the accuracy of the tree SHAP explainer in calculating SHAP values; an experiment was conducted to test the effectiveness of proposed SHAP loss monitoring function.

Besides, the performance of pre-trained adaptive random forest regressor was evaluated on the California housing dataset (experimental dataset) and car price dataset (application dataset), while the performance of pre-trained adaptive random forest classifier was evaluated on the AGRAWAL dataset (experimental dataset) and the lead scoring dataset (application dataset). In both experiments, the pre-trained adaptive random forest regressor and classifier were proven that:

1. The pre-trained ARF classifier's classification performance was at least good or better than ARF classifier that was trained from scratch in both offline settings and online settings.
2. The performance of the pre-trained ARF classifier was at least good or better than TRF classifier during initial training.
3. The performance of the ARF classifier was better than TRF classifier under the influence of data drift or concept drift.
4. The pre-trained ARF regressor's regression performance was at least good or better than ARF regressor that was trained from scratch in both offline settings and online settings.
5. The performance of the pre-trained ARF regressor was at least good or better than TRF regressor during initial training.
6. The performance of the ARF regressor was better than TRF regressor under the influence of data drift or concept drift.

Based on the experimental results above, the web service was usable for other tabular datasets since the transfer learning ensured that the performance of adaptive random forest models was always equivalent or better than the traditional random forest models. By using the transfer learning algorithm, the node split in the pre-

trained adaptive random forest models were guaranteed to be optimal since the traditional random forest models had the complete statistics for every attempt to split a leaf node, including the root node.

In addition, the validation and the experimentation of the web service had been well-documented in the Jupyter Notebook for future re-implementation or references. Thus, this project could serve as the technical reference for those who would like to customize the integration of the sophisticated models with the explainable AI approaches. For example, the image classifier could be integrated with Local Surrogate (LIME) to identity the sections of the image that increased or reduced the prediction probabilities.

7.2 Recommendation

7.2.1 Improvement on transfer learning algorithm

In the experiment, it was observed that the transfer learning process was still slower than the training of the TRF. For example, it took about 10 minutes to transfer 15 base learners of maximum depth of 15 from a traditional random forest classifier to an adaptive random forest classifier. There were two recommendations on improving the performance. First, the algorithm should be implemented using Cython instead of CPython. Cython was a superset of Python programming language that could reduce overhead and improve performance. Cython compiled the Cython code to C code or C++ code that ran more efficiently and faster as compared to CPython.

Second, the part of the CPython codes could be replaced with C extensions. In CPython, the Global Interpreter Lock (GIL) only allowed one and only one thread to execute CPython bytecode at any given time, which caused severe bottleneck in running the transfer learning algorithms. To solve this, the bottleneck code could be replaced with Cython extensions. Using “with nogil” statement, Cython extensions could temporarily release GIL when executing CPU-intensive C or C++ code, allowing GIL to be acquired by another thread to execute CPython bytecode or access Python objects. The examples of the implementation could be found in Scikit-learn or Numpy official source code on GitHub.

7.2.2 Other Improvements

The report and the notebook had documented the example usage of plots constructed using Tree SHAP algorithm. First, the viewers could extend the insights

that the web service provided by adding new plots like SHAP dependence plot. Second, the web service could also be implemented with other methods like Local Surrogate (LIME) instead of using SHAP. Third, the SHAP loss monitoring function could be further improved such that the function was more effective in detecting drifts or data errors. The function would not be effective if the occurrence of the false positive alarms had become more costly than the occurrence of the false negative alarms. In other words, the SHAP loss monitoring function would become less useful if the false positives were too high. Fourth, developers with rich statistical knowledge could also add more statistical tests like ANOVA to the list of techniques used to detect the drifts when truth was not available.

Finally, in critical situations, the online machine learning model deployed in the web service must be secured such that the model would not train with malicious data. Since the web service was only used in predicting car prices and assigning lead scores, securing the machine learning models did not have to be the priority. However, if the web service were used in critical scenarios such as in banking system, the attackers would send malicious data to train the online models in order to manipulate their performance and inference results.

.

BIBLIOGRAPHY

[1] A. Burt, “New AI Regulations Are Coming. Is Your Organization Ready?,” *Harvard Business Review*, Apr. 30, 2021. https://hbr.org/2021/04/new-ai-regulations-are-coming-is-your-organization-ready (accessed Aug. 16, 2021).

[2] Carsome Sdn. Bhd., “CARSOME CONSUMER SURVEY Consumer Sentiments towards: Car Buying and Selling Usage of Public Transport / Ride-hailing Services,” Jan. 2021. Accessed: Mar. 31, 2021. [Online]. Available: https://carsomemy.s3.amazonaws.com/wp/Carsome-Consumer-Survey.pdf

[3] A. Brennen, “What Do People Really Want When They Say They Want ‘Explainable AI?’ We Asked 60 Stakeholders.,” in *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, Honolulu, HI, USA, Apr. 2020, pp. 1–7. doi: 10.1145/3334480.3383047.

[4] Google, “Our Principles – Google AI,” *Google AI*, 2019. https://ai.google/principles/ (accessed Aug. 16, 2021).

[5] H. M. Gomes *et al.*, “Adaptive Random Forests for Evolving Data Stream Classification,” *Machine Learning*, vol. 106, no. 9–10, pp. 1469–1495, Jun. 2017, doi: 10.1007/s10994-017-5642-8.

[6] R. B. Kirkby, “Improving Hoeffding Trees,” PhD Thesis, University of Waikato, 2007. Accessed: Jul. 27, 2021. [Online]. Available: https://researchcommons.waikato.ac.nz/handle/10289/2568

[7] A. Chander and R. Srinivasan, “Evaluating Explanations by Cognitive Value,” in *Lecture Notes in Computer Science*, Hamburg, Germany, Aug. 2018, pp. 314–328. doi: 10.1007/978-3-319-99740-7_23.

[8] C. Molnar, *Interpretable Machine Learning*. Christoph Molnar, 2021. Accessed: Aug. 17, 2021. [Online]. Available: https://christophm.github.io/interpretable-ml-book/

[9] S. M. Lundberg, “The Science behind InterpretML: SHAP,” *www.youtube.com*, May 17, 2020. https://www.youtube.com/watch?v=-taOhqkiuIo (accessed Aug. 17, 2021).

[10] S. M. Lundberg *et al.*, “From Local Explanations to Global Understanding with Explainable AI for Trees,” *Nature Machine Intelligence*, vol. 2, no. 1, pp. 56–67, Jan. 2020, doi: 10.1038/s42256-019-0138-9.

[11] I. Žliobaitė, M. Pechenizkiy, and J. Gama, “An Overview of Concept Drift Applications,” in *Studies in Big Data*, Springer, Cham, 2015, pp. 91–114. doi: 10.1007/978-3-319-26989-4_4.

[12] scikit-learn, “Permutation Importance Vs Random Forest Feature Importance (MDI).” https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html#sphx-glr-auto-examples-inspection-plot-permutation-importance-py (accessed Aug. 18, 2021).

[13] V. Ivanov, “Stability-Based Model Selection in Non-Stationary Environment,” *repository.tudelft.nl*, 2017, Accessed: Apr. 03, 2021. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A31bbf0f5-c43a-4dc0-b2d5-59880368f2a3

[14] A. Lin, “Examining Distributional Shifts by Using Population Stability Index (PSI) for Model Validation and Diagnosis.” Accessed: Apr. 03, 2021. [Online]. Available: https://wuss19.wuss.org/wp-content/uploads/2018/01/47.pdf

[15] R. Taplin and C. Hunt, “The Population Accuracy Index: a New Measure of Population Stability for Model Monitoring,” *Risks*, vol. 7, no. 2, p. 53, May 2019, doi: 10.3390/risks7020053.

[16] A. Bifet and R. Gavald, “Learning from Time-Changing Data with Adaptive Windowing,” presented at the Proceedings of the Seventh SIAM International Conference on Data Minin, Minneapolis, Minnesota, USA, Apr. 2007. Accessed: Jul. 28, 2021. [Online]. Available: https://www.researchgate.net/publication/220907178_Learning_from_Time-Changing_Data_with_Adaptive_Windowing

[17] C. Lin, “Use SHAP Loss Values to debug/monitor Your Model,” *Medium*, Jun. 23, 2020. https://towardsdatascience.com/use-shap-loss-values-to-debug-monitor-your-model-83f7808af40f (accessed Apr. 21, 2022).

[18] A. Chatterjee, “Lead Scoring Dataset,” *www.kaggle.com*, Aug. 17, 2020. https://www.kaggle.com/datasets/amritachatterjee09/lead-scoring-dataset (accessed Apr. 21, 2022).

[19] D. Mauri, “Creating a REST API with Python and Azure SQL,” *GitHub*, Apr. 12, 2022. https://github.com/Azure-Samples/azure-sql-db-python-rest-api/ (accessed Apr. 21, 2022).

[20] S. Lundberg, “Explaining the Loss of a Tree Model - SHAP Latest Documentation,” *Readthedocs.io*, 2018. https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/tree_based_models/Explaining%20the%20Loss%20of%20a%20Model.html (accessed Apr. 21, 2022).

[21] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–37, Apr. 2014, doi: 10.1145/2523813.

[22] Microsoft, “Responsible AI Principles from Microsoft,” *Microsoft*. https://www.microsoft.com/en-us/ai/responsible-ai (accessed Apr. 22, 2022).

APPENDIX A: FORMULA OF ADAPTIVE RANDOM FORESTS ALGORITHM

ARF algorithm was innovated by Gomes and other researchers and this algorithm was directly adapted from the classical Random Forest with ground-breaking improvements [5].

```
1: function AdaptiveRandomForests(m, n, δ_w, δ_d )
2:     initialize T
3:     initialize B
4:     for each s in S do
5:         Extract x, y from s
6:         for each t in T do
7:             ŷ = predict(t, x)
8:             Measure the performance of t using ŷ, y
9:             call Train_RFTree(m, t, x, y)
10:            if Detector(δ_w, t, x, y) detects drift warning then
11:                initialize b and save to B
12:            end if
13:            if Detector(δ_w, t, x, y) detects drift then
14:                Replace t by the corresponding background tree
15:            end if
16:        end for
17:        for all b in B do
18:            call Train_RFTree(m, b, x, y)
19:        end for
20:    end for
21: end function
```

Figure A.1: Pseudocode of Adaptive random forest algorithm

The pseudocode above was the simplified and intuitive version of training adaptive random forests given a data stream of any size.

Label/Line of code	Description
m	Maximum features evaluated per split
n	Total number of base learners/ Hoeffding tree
δ_w	Warning threshold
δ_d	Drift threshold
B	Background trees
T	Current trees
S	Data stream
Detector(...)	Change detection method
Train_RFTree(...)	See the section below for the pseudocode of the function

Table A.2: Symbols used in pseudocode for Adaptive random forest algorithm

The pseudocode of the ARF algorithm as shown above. First, a total of n Hoeffding trees (t) were initialised. Then, for each of the instance (s) from the data stream (S), each t could be used to predict the input data while its learning performance was measured and incrementally trained. With the presence of the true label (y), the test-then-train metrics could be recorded. Test-then-train metrics evaluation allowed the maximum utilisation of data stream since the data instances were used in prediction before training [6]. Meanwhile, the drift warning detector and drift detector in each t was updated with current s. If drift warning detection occurred, then a background tree (b) was initialised to be trained later. If the drift detection occurred, then the current t would be replaced by its corresponding b. Besides looping T, each b was iterated to incrementally trained with the current s.

APPENDIX B: FORMULA OF HOEFFDING TREE

Hoeffding tree was the base learner of the ARF algorithm instead of the decision tree. Hoeffding tree was the perfect solution for solving the inefficiencies of offline machine learning algorithms. It was because the Hoeffding tree could be incrementally trained with data of any size in the setting of high-speed data stream processing. As a result, only the Hoeffding tree itself and other information that were crucial to train the tree were stored in the memory [6]. This saved a lot of memory space and sped up the training process.

```
1: function Train_RFTree(m, t, x, y)
2:     if Poisson(λ = 6) > 0 then
3:         Find the l in t that corresponds to the given current train sample
4:         call IMPORTANT_STATS(l)
5:         if CONDITION_1(x) is true and CONDITION_2 is true then
6:             Compute Ḡ for each x feature
7:             Compute ϵ
8:             if (Ḡ(X_1st) − Ḡ(X_2nd)) > ϵ or ϵ < τ then
9:                 Split the current node
10:            end if
11:        end if
12:    end if
13: end function
```

Figure B.1: Pseudocode of Hoeffding tree

Label/Line of code	Description
l	Leaf node
ϵ	The Hoeffding bound
$\bar{G}$	Estimated information gain
$X_{1^{st}}$	The attribute with the highest $\bar{G}$
$X_{2^{nd}}$	The attribute with second-highest $\bar{G}$
CONDITION_1(x)	A Boolean flag indicating whether x was the n^{th} n_{min} of data. The model was updated every n_{min}
CONDITION_2(x)	A Boolean flag indicating whether samples of belong than one class was observed
IMPORTANT_STATS(l)	A code snippet to increase the observed number of samples and their labels at the leaf node by one.
$X_{1^{st}} \neq X_{\phi}$	A form of pre-pruning to stop splitting if the $X_{1^{st}}$ did not look better than X_{ϕ} (Kirkby, 2007).

$[\bar{G}(X_{1^{st}}) - \bar{G}(X_{2^{nd}})] > \epsilon$	Logic condition for the Hoeffding bound to decide whether $X_{1^{st}}$ could proceed to be split (Kirkby, 2007).
$\epsilon < \tau$	If the condition was true, then the top and first two attributes were selected without considering other attributes (Kirkby, 2007).

Table B.2: Symbols used in pseudocode for Hoeffding Tree

The pseudocode to train the Hoeffding tree was shown in the diagram above. First, the function Train_RFTree received four parameters, where m represented the maximum number of features to be evaluated per split, t represented the current Hoeffding tree (background or old), x represented input features, while y represented output label. Then, the leaf node l in t that corresponds to the given current train sample was located. In l, some important statistics was updated to provide the necessary information for estimating the information gain of splitting on each attribute.

To check if the split should proceed, it must be ensured that the current single sample being received was the n^{th} of n_{min} observed samples, where n_{th} can be any number from 1 to ∞. For example, if the grace period was 200, then the split only proceeded if the sample was the $200^{th}, 400^{th}, 600^{th}, 800^{th}, \ldots$ sample. The grace period was important since insufficient samples would not provide sufficient information to know which attribute was best to split. If the previous condition held true, it must be ensured that the observed samples since the last split belong to more than one class. In other words, there was nothing to split if all samples had the same output.

If both the aforementioned logical conditions were true, then the estimated information gain ($\bar{G}$) was computed for each x feature. Then, two features with the highest $\bar{G}$, which were $X_{1^{st}}$ and $X_{2^{nd}}$ were obtained respectively. Along the way, ϵ was computed using the formula:

$$\epsilon = \sqrt{\frac{R^2 \ln\left(\frac{1}{\delta}\right)}{2n}}$$

, where R^2 was the range of random variable, δ was the split confidence, and n was the number of observations. Split confidence, known as δ, was one of the parameters that influence the Hoeffding bound. When the split confidence was low, then the

probability of choosing the correct attribute to split in the current leaf node increased [6].

After that, it must be ensured that the difference between $\bar{G}$ of $X_{1^{st}}$ and $X_{2^{nd}}$ was more than ϵ to proceed with the splitting process [6]. However, there might exist a tie situation such that two or more attributes had the same estimated information gain value. In this situation, the top two attributes with the highest estimated information gain cannot be determined and the growth of the tree would be stagnant [6]. To solve this, a threshold called tie-breaking threshold (τ) was required to break the tie. If the ϵ was small than the τ, then the top and first two attributes were selected without considering others [6]. Finally, the current node was split based on $X_{1^{st}}$.

APPENDIX C: FORMULA OF ADWIN

Adaptive windowing (ADWIN) was one of the adaptive windows methods for checking for any distribution difference in the current window of data. As new data was added to the window, ADWIN would automatically adjust the window size (W) by growing when the data was stationary or shrinking when the drift was detected [16]. As a result, the distribution of data after adjusting W would always represent the latest and most accurate distribution [16]. ADWIN could function as an effective data drift detector for the Hoeffding tree. It was because ADWIN could automatically perform these three functions without requiring manual adjustment of hyperparameters like window size. The three included functions were detecting which sub-window within the current W where the change occurred, deciding which data instances to keep and which one to forgot to reduce memory footprint, and alerted the models when significant changes were detected [16].

Mathematically, in a window (W) of size n, an optimal separation between two sub-windows, W_0 and W_1 needed to be located to estimate the latest data distribution as accurate as possible and discarded the older data distribution to reduce memory footprint. To do so, first, the formulas were applied as shown below:

$$m = \frac{1}{\frac{1}{n_0} + \frac{1}{n_1}}, \delta' = \frac{\delta}{(\ln n)}, and\ \epsilon_{cut} = \sqrt{\frac{2}{m} \cdot \sigma_W^2 \cdot \ln\left(\frac{2}{\delta'}\right)} + \frac{2}{3m} \cdot \ln\frac{2}{\delta'}$$

, where n_0 was the length of W_0, n_1 was the length of W_1, m was the harmonic means of n_0 and n_1, δ' was the global threshold to ensure that all the possible n of W_0 and W_1 below δ, and ϵ_{cut} was the threshold for checking whether the observed $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ were statistically significantly different in both W_0 and W_1 [16].

```
1: initialize W
2: for each t > 0
3:     do add x_t to the head of W
4:         repeat Drop elements from the tail of W
5:             until |μ̂_W0 − μ̂_W1| ≥ ε_cut holds
6:                 for every split of W into W = W_0 · W_1
7:         output μ̂_W
```

Figure C.1: Pseudocode of algorithm ADWIN

Based on the pseudocode above, for every new value x_t arrived at time t, the size of the windows was adaptively adjusted to achieve the purposes mentioned above. The W_0 and W_1 were repeatedly split for up to maximum $\log_2 n$ possible times until the observed average in both sub window differed more than ϵ_{cut}. In order to limit the number of tries to split the W_0 and W_1, the formula was changed from $\delta' = \frac{\delta}{n}$ to $\delta' = \frac{\delta}{(\ln n)}$ [16].

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: JAN, 2022	**Study week no.:** 2
Student Name & ID: YAP JHENG KHIN (18ACB00224)	
Supervisor: Ts Sun Teik Heng @ San Teik Heng	
Project Title: Car Dealership Web Application	

1. WORK DONE

- The transfer learning algorithm was implemented to improve the initial performance of the adaptive random classifier.

2. WORK TO BE DONE

- Develop a similar transfer learning algorithm to improve the initial performance of the adaptive random regressor.

3. PROBLEMS ENCOUNTERED

- The transfer learning algorithm contained bugs. More time was required to analyze the River official source code.

4. SELF EVALUATION OF THE PROGRESS

Supervisor's signature

Student's signature

Trimester, Year: JAN, 2022	**Study week no.:** 4
Student Name & ID: YAP JHENG KHIN (18ACB00224)	
Supervisor: Ts Sun Teik Heng @ San Teik Heng	
Project Title: Car Dealership Web Application	

1. WORK DONE

- Both transfer learning algorithms were implemented.

2. WORK TO BE DONE

- Analyze the performance of the pre-trained adaptive random forest regressor and classifier.

3. PROBLEMS ENCOUNTERED

- The pre-trained random forest classifier failed when trained on new samples. More time was needed to identity the root problem.

4. SELF EVALUATION OF THE PROGRESS

The progress was slow due to other subject's assignment. Hence, the assignment must be completed as soon as possible.

Sun Teik Heng

Supervisor's signature

Student's signature

Trimester, Year: JAN, 2022	**Study week no.:** 6
Student Name & ID: YAP JHENG KHIN (18ACB00224)	
Supervisor: Ts Sun Teik Heng @ San Teik Heng	
Project Title: Car Dealership Web Application	

1. WORK DONE

- Two experiments were conducted to measure the performance of the adaptive random forest regressor and classifier.

2. WORK TO BE DONE

- Train the adaptive random forest regressor and classifier on the car price and lead scoring dataset.
- Preprocess the lead scoring dataset.

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

The progress was slow due to other subject's assignment. Hence, the assignment must be completed as soon as possible.

Sun Teik Heng

Supervisor's signature

Student's signature

Trimester, Year: JAN, 2022	**Study week no.:** 8
Student Name & ID: YAP JHENG KHIN (18ACB00224)	
Supervisor: Ts Sun Teik Heng @ San Teik Heng	
Project Title: Car Dealership Web Application	

1. WORK DONE

- The car price model and the lead scoring model had been trained and their performance were evaluated.

2. WORK TO BE DONE

- Pass in the car price model and the lead scoring model into the tree SHAP API to calculate the SHAP values.
- Design a function to extract the tree weights from the car price model and the lead scoring model.
- Implement the drift monitoring using statistical tests.

3. PROBLEMS ENCOUNTERED

- The tree SHAP explainers did not support the weight extraction of the River models. However, the explainers did accept a Python dictionary that contained the weights of a tree model. More time was required to discover how to extract the weights manually.

4. SELF EVALUATION OF THE PROGRESS

The progress was slow due to other subject's assignment. Hence, the assignment must be completed as soon as possible.

Sun Teik Heng

Supervisor's signature

Student's signature

Trimester, Year: JAN, 2022	**Study week no.:** 10
Student Name & ID: YAP JHENG KHIN (18ACB00224)	
Supervisor: Ts Sun Teik Heng @ San Teik Heng	
Project Title: Car Dealership Web Application	

1. WORK DONE

- The function that extracted the weights from both adaptive random forest regressor and classifier was implemented.
- The drift monitoring using statistical tests had been implemented.

2. WORK TO BE DONE

- Re-evaluate the transfer learning algorithms.
- Initialize the tree SHAP explainers with the dictionaries and use the explainers to visualize graphs.
- Complete the CRUD functionalities for the lead management module.
- Draw the activity diagram of the web application and the web service.
- Convert the car price dataset and lead scoring dataset into application data.
- Figure out how to seed the application data into Azure SQL database.
- Develop the web service using the source code from the Jupyter notebooks.
- Figure out how the web service could query the application data form the Azure SQL database.

3. PROBLEMS ENCOUNTERED

- The calculation of the SHAP values were not accurate, more time was required to debug the issue.

4. SELF EVALUATION OF THE PROGRESS

Sun Teik Heng

Supervisor's signature

Student's signature

Trimester, Year: JAN, 2022	**Study week no.:** 12
Student Name & ID: YAP JHENG KHIN (18ACB00224)	
Supervisor: Ts Sun Teik Heng @ San Teik Heng	
Project Title: Car Dealership Web Application	

1. WORK DONE

- The problem that caused the inaccuracies in calculating the SHAP values was identified.
- The CRUD functionalities for the lead management module were implemented.
- The web application also had successfully deployed to the cloud. At the same time, the application data was successfully seeded to the cloud database as well.
- The activity diagram of the web application and the web service were drawn.

2. WORK TO BE DONE

- Develop the web service by transferring the source code from the Jupyter notebooks.
- Figure out which Docker image to use.
- Copy the important documentations from the Jupyter notebooks to the report.
- Complete the report.

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

Sun Teik Heng

Supervisor's signature

Student's signature

Trimester, Year: JAN, 2022	**Study week no.:** 13
Student Name & ID: YAP JHENG KHIN (18ACB00224)	
Supervisor: Ts Sun Teik Heng @ San Teik Heng	
Project Title: Car Dealership Web Application	

1. WORK DONE

- All the functionalities of the web service had been implemented except for constructing running metric plots.
- Two docker images were created and merged into one single application using Docker compose.

2. WORK TO BE DONE

- Figure out how to deploy the multi-container applications to the Azure Container Instances.
- Copy the important documentations from the Jupyter notebooks to the report.
- Complete the report.

3. PROBLEMS ENCOUNTERED

- The multi-container applications could not be deployed due to technical issues.

4. SELF EVALUATION OF THE PROGRESS

Supervisor's signature

Student's signature

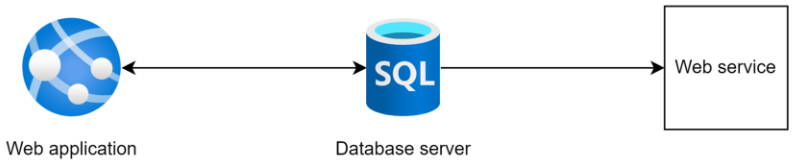
POSTER

FYP II

Car Dealership Web Application

Student: Yap Jheng Khin
Supervisor: Ts Sun Teik Heng @ San Teik Heng

Problem	Solution
• People did not understand and trust AI	• Implement explainable AI functionalities using Tree SHAP
• Data pattern evolved from time to time due to external changes • Model became irrelevant over time	• Use statistical tests and SHAP loss monitoring function to monitor drift • Use adaptive random forest (ARF) models for automatic model retraining



Web Application

Inventory management module
- Car price analytics

Lead management module
- Lead scoring prediction

Web Service

SHAP web service
- Explain individual prediction
- Explain individual model loss
- Explain model's overall prediction behavior
- Evaluate model's performance
- Monitor drift

River web service
- Incrementally train adaptive random forest models

Project Contribution

- Serve as a technical reference on how to customize the integration of the sophisticated models with the explainable AI approaches.
- Encourage the adoption of sophisticated machine learning models into the commercial applications

Implementation

Transfer learning algorithm
- Copy training weights from traditional random forest models to ARF models
- **Improve** the initial performance of the ARF models

SHAP loss monitoring function
- **Improve** existing function
- Add statistical tests to be more effective in detecting drifts and data errors

Tree weight extraction function
- Extract weights into dictionary before passing to the SHAP explainer
- Ensure **compatibility** between River models and Tree SHAP explainers

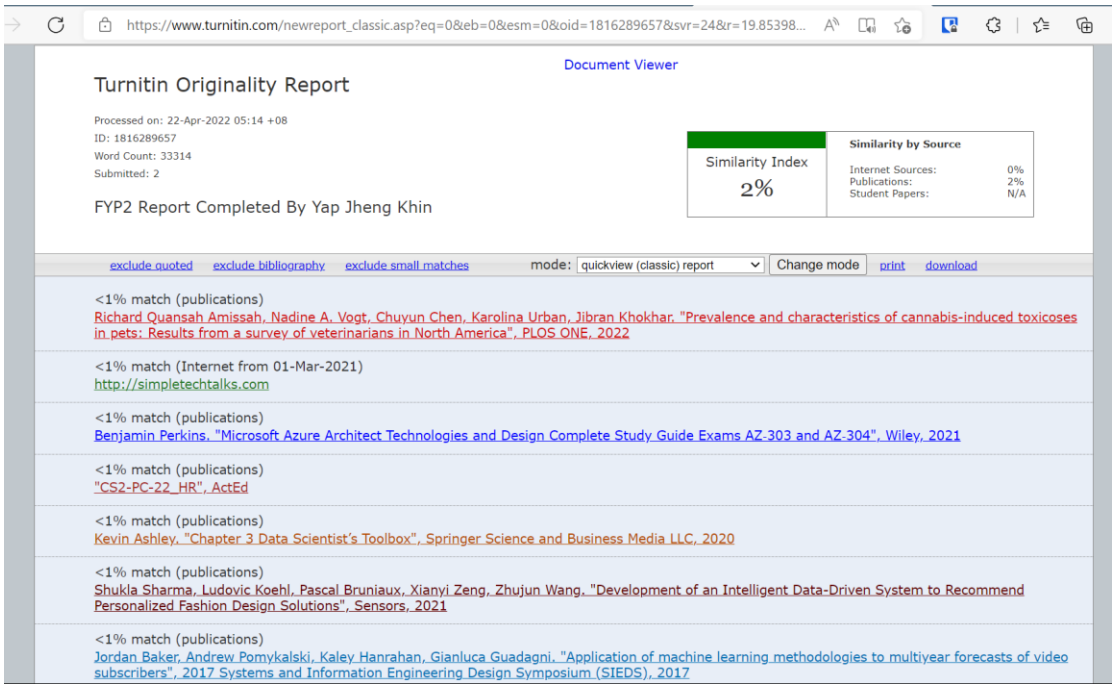
Experimentation & Validation

Transfer learning algorithm
- Initial performance boosted? ☑

SHAP loss monitoring function
- More effective in detecting drifts and data errors? ☑

Tree SHAP algorithm
- The calculation of SHAP values was accurate? ☑

PLAGIARISM CHECK RESULT

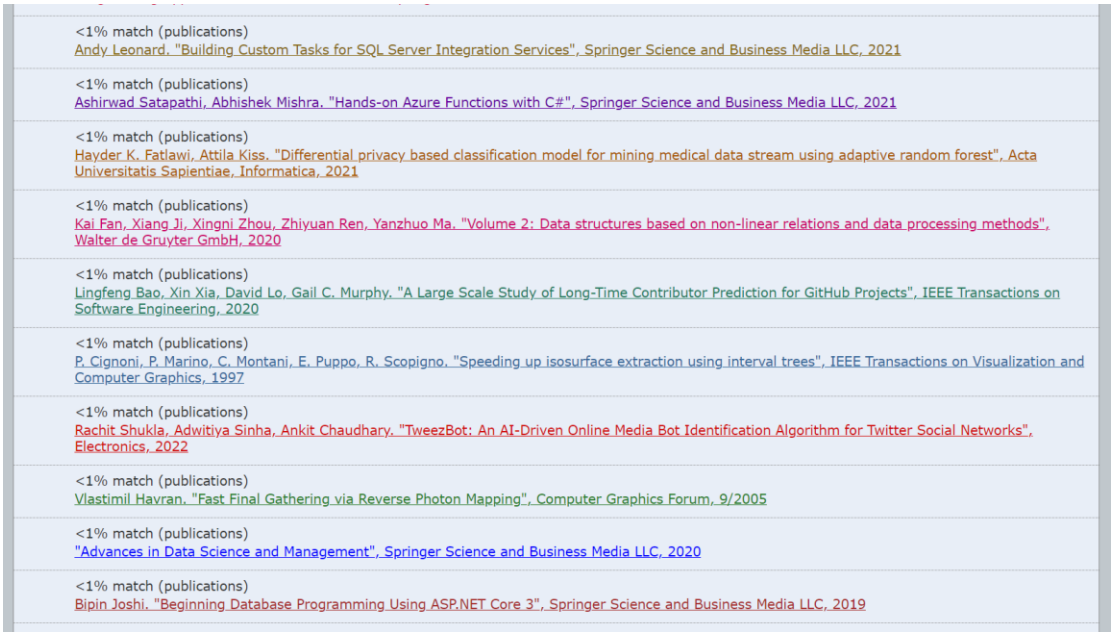


Plagiarism Classic Report (Part I)

Plagiarism Classic Report (Part II)

Plagiarism Classic Report (Part III)

Plagiarism Classic Report (Part IV)



<1% match (publications)
Andy Leonard, "Building Custom Tasks for SQL Server Integration Services", Springer Science and Business Media LLC, 2021

<1% match (publications)
Ashirwad Satapathi, Abhishek Mishra, "Hands-on Azure Functions with C#", Springer Science and Business Media LLC, 2021

<1% match (publications)
Hayder K. Fatlawi, Attila Kiss, "Differential privacy based classification model for mining medical data stream using adaptive random forest", Acta Universitatis Sapientiae, Informatica, 2021

<1% match (publications)
Kai Fan, Xiang Ji, Xingni Zhou, Zhiyuan Ren, Yanzhuo Ma, "Volume 2: Data structures based on non-linear relations and data processing methods", Walter de Gruyter GmbH, 2020

<1% match (publications)
Lingfeng Bao, Xin Xia, David Lo, Gail C. Murphy, "A Large Scale Study of Long-Time Contributor Prediction for GitHub Projects", IEEE Transactions on Software Engineering, 2020

<1% match (publications)
P. Cignoni, P. Marino, C. Montani, E. Puppo, R. Scopigno, "Speeding up isosurface extraction using interval trees", IEEE Transactions on Visualization and Computer Graphics, 1997

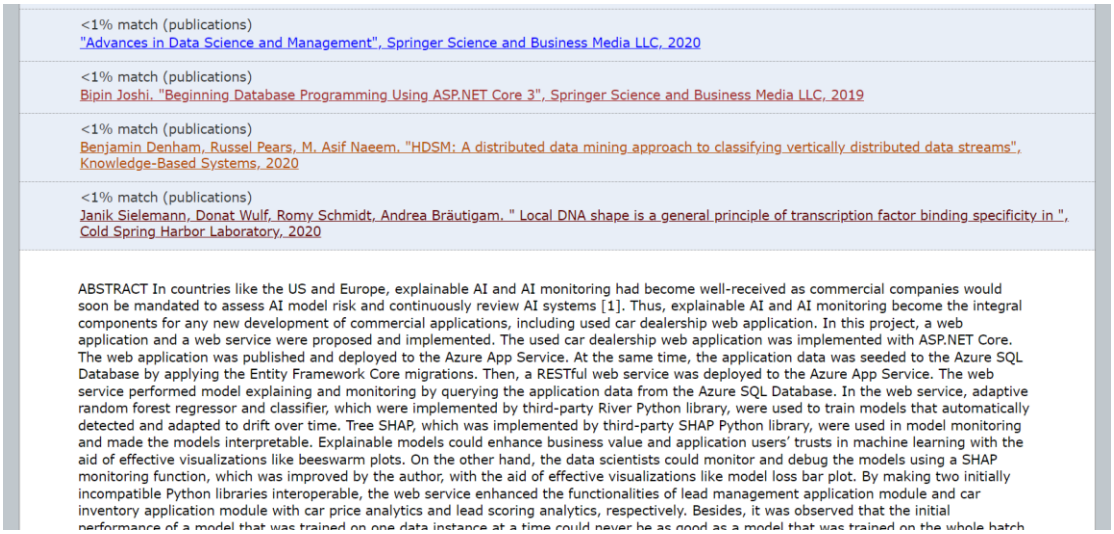
<1% match (publications)
Rachit Shukla, Adwitiya Sinha, Ankit Chaudhary, "TweezBot: An AI-Driven Online Media Bot Identification Algorithm for Twitter Social Networks", Electronics, 2022

<1% match (publications)
Vlastimil Havran, "Fast Final Gathering via Reverse Photon Mapping", Computer Graphics Forum, 9/2005

<1% match (publications)
"Advances in Data Science and Management", Springer Science and Business Media LLC, 2020

<1% match (publications)
Bipin Joshi, "Beginning Database Programming Using ASP.NET Core 3", Springer Science and Business Media LLC, 2019

Plagiarism Classic Report (Part V)



<1% match (publications)
"Advances in Data Science and Management", Springer Science and Business Media LLC, 2020

<1% match (publications)
Bipin Joshi, "Beginning Database Programming Using ASP.NET Core 3", Springer Science and Business Media LLC, 2019

<1% match (publications)
Benjamin Denham, Russel Pears, M. Asif Naeem, "HDSM: A distributed data mining approach to classifying vertically distributed data streams", Knowledge-Based Systems, 2020

<1% match (publications)
Janik Sielemann, Donat Wulf, Romy Schmidt, Andrea Bräutigam, " Local DNA shape is a general principle of transcription factor binding specificity in ", Cold Spring Harbor Laboratory, 2020

ABSTRACT In countries like the US and Europe, explainable AI and AI monitoring had become well-received as commercial companies would soon be mandated to assess AI model risk and continuously review AI systems [1]. Thus, explainable AI and AI monitoring become the integral components for any new development of commercial applications, including used car dealership web application. In this project, a web application and a web service were proposed and implemented. The used car dealership web application was implemented with ASP.NET Core. The web application was published and deployed to the Azure App Service. At the same time, the application data was seeded to the Azure SQL Database by applying the Entity Framework Core migrations. Then, a RESTful web service was deployed to the Azure App Service. The web service performed model explaining and monitoring by querying the application data from the Azure SQL Database. In the web service, adaptive random forest regressor and classifier, which were implemented by third-party River Python library, were used to train models that automatically detected and adapted to drift over time. Tree SHAP, which was implemented by third-party SHAP Python library, were used in model monitoring and made the models interpretable. Explainable models could enhance business value and application users' trusts in machine learning with the aid of effective visualizations like beeswarm plots. On the other hand, the data scientists could monitor and debug the models using a SHAP monitoring function, which was improved by the author, with the aid of effective visualizations like model loss bar plot. By making two initially incompatible Python libraries interoperable, the web service enhanced the functionalities of lead management application module and car inventory application module with car price analytics and lead scoring analytics, respectively. Besides, it was observed that the initial performance of a model that was trained on one data instance at a time could never be as good as a model that was trained on the whole batch

Plagiarism Classic Report (Part VI)

Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name of Candidate	YAP JHENG KHIN
ID Number(s)	18ACB00224
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	Car Dealership Web Application

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)**
Overall similarity index: 2% **Similarity by source** Internet Sources: 0 % Publications: 2 % Student Papers: 0 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required, and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words.**	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Sun Teik Heng

Signature of Supervisor

Name: Sun Teik Heng @ San Teik Heng

Date: 22/4/2022

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

(KAMPAR CAMPUS)

FYP2 CHECKLIST

Student Id	18ACB00224
Student Name	Yap Jheng Khin
Supervisor Name	Ts Sun Teik Heng @ San Teik Heng

TICK (√)	DOCUMENT ITEMS
	Front Plastic Cover (for hardcopy)
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

I, the author, have checked and confirmed all the items listed in the table are included in my report.

Date: 21-04-2022