IoT DATABASE FOR NON-STRUCTURED DATA TYPE

BY

LOH YI LING

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) INFORMATION SYSTEMS ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2023

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: IoT DATABASE FOR NON-STRUCTURED DATA TYPE

Academic Session: MAY 2023

I LOH YI LING

(CAPITAL LETTER)

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,
[illegible]	[illegible]
(Author's signature)	(Supervisor's signature)
Address:	
	TS DR OOI BOON YAIK
	Supervisor's name
Date: 7/9/2023	**Date**: 15/9/202

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 7/9/2023

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ***LOH YI LING*** (ID No: ***19ACB06235***) has completed this final year project/ dissertation/ thesis* entitled " ***IoT DATABASE FOR NON-STRUCTURED DATA TYPE*** " under the supervision of ***TS DR OOI BOON YAIK*** (Supervisor) from the Department of Computer Science, Faculty of Information And Communication Technology.

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

Yours truly,

_Loh Yi Ling____
(*Student Name*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**IoT DATABASE FOR NON-STRUCTURED DATA TYPE**" 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	:	[illegible]
Name	:	LOH YI LING
Date	:	7/9/2023

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Ts Dr. Ooi Boon Yaik, for his invaluable guidance and unwavering support throughout the duration of my Final Year Project (FYP), titled "IoT Database for Non-Structured Data Type." I am grateful for the countless hours he devoted to mentoring me, providing critical insights, and helping me navigate the challenges that inevitably arise during such a project. I would like to extend my appreciation to my moderator, Dr Abdulkarim Kanaan Jebna as well. The insightful feedback, and willingness to share his knowledge have enriched my understanding of the subject matter and significantly contributed to the quality of this research.

I also want to extend my appreciation to my family and friends for their unwavering support and encouragement throughout this journey. Their belief in me has been a constant source of motivation.

Once again, thank you, Dr. Ooi Boon Yaik, for being an exceptional mentor, and for your role in making this FYP a meaningful and rewarding experience.

ABSTRACT

This Final Year Project (FYP) delves into the development and evaluation of FastTextProcessor, a Python library designed to simplify two fundamental data processing tasks: querying by string and calculating mean values. It deftly addresses the first objective by offering a data querying mechanism that liberates users from the hassles of populating data like existing NoSQL databases. With FastTextProcessor, users can query data directly from the file directory, accelerate the data extraction process. Next, the second objective finds its fulfillment in FastTextProcessor's innate ability to query the data and perform calculation directly from the file directory without the need of formatting data like conventional databases or other data extraction tools. In a world where time is money, FastTextProcessor's capacity to expedite coding workflows underscores its importance. In summary, FastTextProcessor stands as a beacon of innovation in the realm of data extraction. It aligns seamlessly with the project's objectives, enabling data to be queried directly and processed without the burden of extensive formatting. Its potential to streamline data analytics workflows is bound to empower users and revolutionize their approach to data analysis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

LIST OF ABBREVIATIONS

CSV	Comma-separated values
JSON	JavaScript Object Notation
XML	Extensible Markup Language
SQL	Structured Query Language
IOT	Internet of Things
SSIS	*SQL Server Integration Services*
RAM	Random Access Memory
ETL	Extract, transform, load
PyPI	Python Package Index

Chapter 1

Introduction

1.1 Problem Statement and Motivation

The problem statement of this project can be summarized as below:

1. The data of existing NoSQL database requires to be populated before it can be query. Generally, it requires a pre-processing program for data analytics.
2. The data usually has to be formatted into the right format depending on the data analytics software before it can be run by data analytics software. This allows two programs to be developed just to perform data extraction, transformation, and loading.

The problem statement of this project is the data of existing NoSQL database requires to be populated before it can be query. In general, it requires a pre-processing program for data analytics. The conventional database requires data modeling process before queries. It is very time-consuming and labour intensive since the developer needs to implement the schema before storing into database after the injection of raw data. For instance, [5] stated that the conventional database has many limitations such as two dimensional table to represent the data, strict consistency of database transactions, more time to read and write and the complexity of queries. Besides, the schema database is difficult to accelerate the database's launch which slows down the project. [6] has proposed that the designer's knowledge and skillfulness have a large impact on the quality of the resulting database schema. Thus, this is a major obstacle facing by many developers when they want to extract data using conventional database and data analysis tools.

Furthermore, the second problem of this project is the data usually has to be formatted into the right format depending on the data analytics software before it can be run by data analytics software. This allows two programs to be developed just to perform data extraction, transformation, and loading. For instance, the raw data requisites to change to other formats such as CSV, JSON, XML etc. The schema-less database such as MongoDB eschewed the traditional table based relational database structure in favor of JSON like documents. The raw data in another format such as videos, pictures and audios need to be convert into BSON (Binary JSON) format to be stored in MongoDB[7]. Hence, it is a laborious and complicated process for the user to perform query since it requires to change from one format to another format.

The main goal of this FYP is to address challenges in conventional databases and data analytics. Conventional databases often require extensive effort to insert raw data before querying. For example, inserting data into an SQL server involves many steps, which can be time-consuming. This can lead to data being queried infrequently, making the effort seem unwarranted.

To solve this problem, the project aims to develop innovative solutions for data extraction and processing to speed up querying. This will enhance the efficiency of data analytics, especially in the era of IoT, where vast data requires efficient storage and management. The tool created by this project will boost efficiency and convenience for engineers working with large datasets.

In conclusion, the project's main goal is to simplify the user experience and expedite data-driven decision-making by making data management and analysis easier. It seeks to empower developers and users with tools to streamline data processing and querying, making large datasets more accessible and efficient to handle.

1.2 Objectives

The objectives of this project be summarized as below:

1. Implement a preprocessing program for data analytics so that data does not need to be populated before it can be queried like existing NoSQL database.
2. Develop a program that allows data to be run directly by data analytics software without formatting the data.

The first project objective is to revolutionize data preprocessing, specifically targeting the time-consuming process before querying in conventional databases. Conventional database systems often require structuring the data schema before querying, leading to significant time and labor overhead. This project aims to develop a cutting-edge preprocessing program for data analytics, allowing immediate query execution without populating the data. This streamlined approach will boost efficiency and free data analysts and developers to focus on deriving insights from the data.

The project also aims to simplify and speed up data preparation. The program will act as a bridge between raw data and analytics software, eliminating intermediary steps like data formatting. It can process various file types directly from the file directory, eliminating the need for complex data conversion. Additionally, it enhances query efficiency and reduces labor-intensive data preparation efforts.

1.3 Project Scope and Direction

The project's expected outcome is a program capable of processing and filtering files directly from the provided directory, even in various formats like text, CSV, and JSON. This eliminates the need for manual data population and formatting before querying. The program will have at least two essential functions, including a "search and match" feature to search for user-specified keywords accurately.

Additionally, the project will include arithmetic operations, such as calculating average values within files and an overall average based on user input. To enhance user convenience, the program will support log and output files, automatically generated after each query operation, providing a comprehensive record of interactions. In summary, the project aims to streamline data querying with search and matching functions, arithmetic operations, and additional features like log and output files for an improved user experience.

1.4 Contributions

This project makes several significant contributions to the field of data management and analysis:

1. **Streamlined Data Processing**: To simplify data processing and querying. It allows direct processing and filtering of files from a given directory, regardless of their format (e.g., text, CSV, JSON), eliminating the need for time-consuming data population and formatting and greatly enhancing efficiency.
2. **Data Query Functionality**: Introduces the "search and match" and the ability to calculate the mean. It streamlines the process of locating and retrieving data, simplifying the data query process and reducing the effort required for data queries.

3. User-Friendly Features: Offers automated log and output file generation, providing users with a detailed record of their program interactions for easy result tracking and analysis.

In conclusion, this project's contributions encompass improved data processing efficiency, enhanced user experience through data query operations, and the inclusion of user-friendly features. These contributions collectively address the challenges associated with data management and analysis in today's data-rich environments, making it a valuable asset for developers and users alike.

1.5 Report Organization

This report is organized into 7 chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Methodology, Chapter 4 System Design, Chapter 5 System Implementation, Chapter 6 System Evaluation and Discussion, and Chapter 7 Conclusion. The first chapter is the introduction of this project which includes problem statement and motivation, project scope, project objectives, project contribution, and report organization. . In Chapter 2, it shows the literature review of tools like

MongoDB, SQLite, Apache Parquet, and Pandas DataFrame. The literature review for each journal also written in Chapter 2. The introduction, strengths and drawbacks are analyzed in this chapter. For Chapter 3, it reveals the system approach and also the project workflow of this project. The technologies involved in this project and Gantt Chart also recorded in Chapter 3. Next, the block diagram of proposed solution is in Chapter 4. Chapter 5 covers hardware and software setup, while Chapter 6 presents testing results comparing the proposed solution with other tools. The last chapter provides conclusions and recommendations.

Chapter 2

Literature Review

2.1.1 Apache Spark

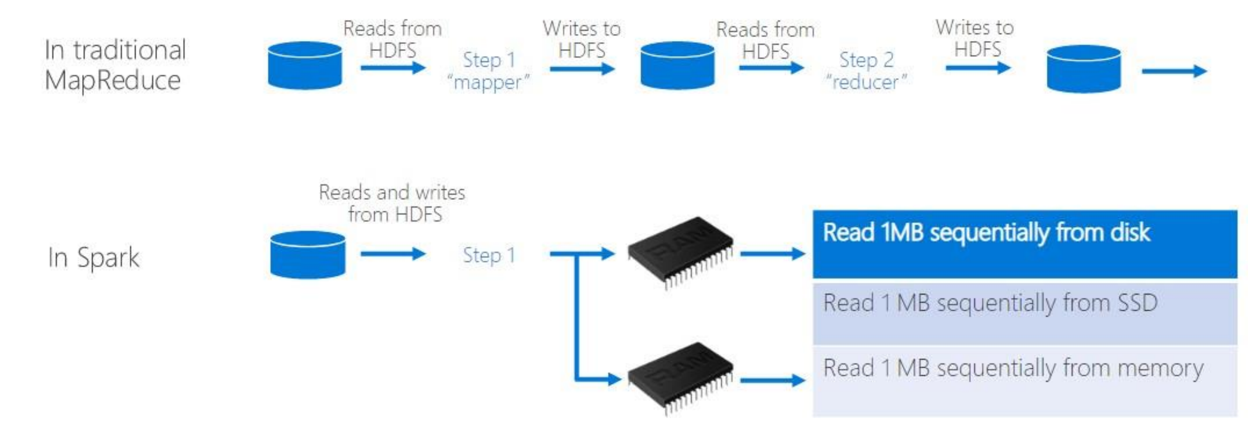


Figure 2.1.1 Operations of Apache Spark compared to traditional MapReduce [8]

Apache Spark enables user to perform data operations using modern functional programming languages such as Scala and Python, which are already widely used by data experts. [9] stated that it is a general-purpose cluster computing framework with language-integrated APIs in Scala, Java, Python and R. Figure 2.1.1 indicates the operations of Apache Spark compared to traditional MapReduce. Besides, Apache Spark also introduces an abstraction as known as resilient distributed datasets (RDDs) which aims to retain the scalability and fault tolerance of MapReduce. Furthermore, it supports the applications with working sets. The RDD represents a read-only collection of objects partitioned across a set of machines and rebuildable if a partition is lost [10]. In addition, Apache Spark can load, cache, and query data in memory repeatedly. Besides, according to [10], Spark is the first program that allows an efficient, general-purpose programming language to be used interactively on a cluster to process large datasets.

Strengths of Apache Spark:

- **High Speed and Great Performance**

  The fast speed of performing processing tasks on huge datasets. This is due to the in-memory computation in Spark, the speed of processing has increase.

Besides, the data is being cached and it allows user not to fetch data from the disk every time. So, the time being saved by this feature. Furthermore, [11] claimed that PySpark has a DAG (Directed Acyclic Graph) execution engine that aids in-memory computation and acyclic data flow, resulting in high speed.

- **A Unified Engine**

  It has a unified engine which comes packaged with higher-level of libraries. For instance, it includes support for streaming data, machine learning, graph processing and SQL queries.
- **Developer-Friendly Tools**

  Apache Spark provides popular language bindings for Python and R, Java and Scala, so application developers and data scientists can take advantage of the scalable tools and dependable performance and speed without going into too much detail [12].

Drawbacks of Apache Spark:

- **Lack of File Management System**

  Apache Spark does not come with its own file management system. Hence, it requires to be integrated with external platforms due to the lack of file management system in it.
- **Small Files Issue**

  One of the limitations of Apache Spark is the small files issue. This problem arises when developer use Spark with Hadoop. It is due to the reason that Hadoop Distributed File System (HDFS) offers a small number of huge files rather than a large number of small files.
- **Lack of Support for Complete Real-Time Processing**

  In fact, Apache Spark has no support for fully real-time data stream processing. For example, the live data stream is automatically portioned into batches in Spark streaming. When those batches are of a predetermined interval, each batch of data is processed as a Spark RDD [13].
- **High Memory Consumption**

  Apache Spark requires large RAM to process in-memory, so the cost is extremely high. This is because the memory consumption is very high when the developer works with Spark to process data.

- **Manual Optimization**

  Developers need to specify the number of Spark partitions on their own in order to create partitions. The number of fixed partitions must be passed as a parameter to the parallelize method [13]. For certain aspects, including as partitioning and caching in Spark, must be managed manually.

- **Higher Latency but Lower Throughput**

  The term “latency” refers to the total amount of time spending between starting a job and getting initial results. Next, the definition of “throughput” is it determines the amount of work done over a given time period which can also be thought of as a measure of efficiency [14]. The latency of Apache Spark is generally greater which results in lower throughput when compared to other tools such as Apache Flink which has lower latency but higher throughput.

2.1.2 RapidMiner

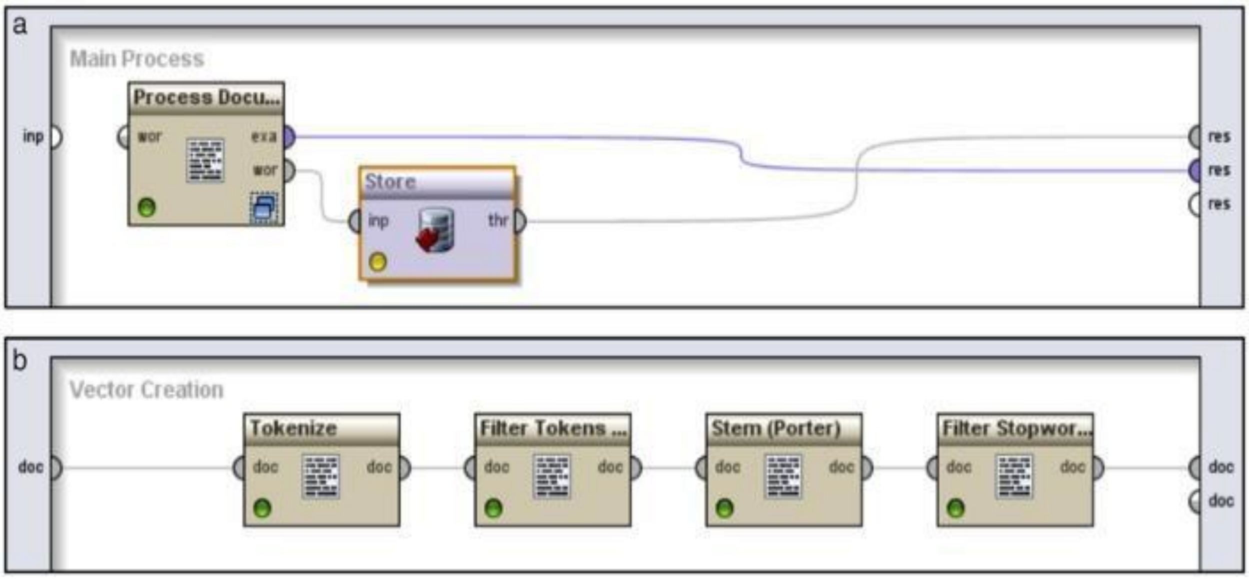


Figure 2.1.2 Rapid miner main process flow for data preprocessing and subprocess flow for vector creation[15]

Rapid Miner is a general integrated machine learning and predictive analytic environment [15]. It is a data science and data mining platform that lets users extract, transform, and load data to draw insights. The research paper [16] has carried out a research to investigate the importance of text data preprocessing and implementation using RapidMiner. According to [16], it stated that the data preparation is done by data preprocessing while the preprocessing of text refers to cleaning of noise. In rapid miner, the document will be converted into vector model via “Process Document from Data” after the completion of preprocessing.

Strengths of RapidMiner:

- **Reading databases at high speed**

  Rapid miner can read and merge all types of databases such as SQL Server, Informix, MySQL, and Oracle at fast speed. Furthermore, the configuring access is also straightforward because the drivers are inbuilt. Besides, it is also easy to find new java drivers to let RapidMiner to connect to other databases.

- **Convenience**

  RapidMiner allows user to perform various types of transformations, calculations like date and percentages, joins, and filter without writing any coding. It is convenient for users especially when there are several databases to be integrated.

- **Great data visualization**

  RapidMiner is a feature-rich visual workflow designer programme for developers. The data visualization process can aid user to perform better data preprocessing and modelling. After a few clicks, drags, and drops, new learners may immediately see and understand the effect of applying various algorithms and functions.

Drawbacks of RapidMiner:

- **Unable to change the behavior of existing machine learning algorithm**

  It includes a repository with many machine learning algorithms and functions but developers are unable to modify the way a genetic algorithms mutates.

- **Limited partitioning abilities for dataset to training and testing sets**

  It does not contain enough number of tutorials and samples. It is due to the reason that users require more real examples and real cases to study and understand the best practices in data modelling. Although RapidMiner has provided some datasets for training and testing purposes, but it is limited as users need more sample data and examples.

- **Unfriendliness design and functions**

  RapidMiner should try to improve the friendliness with using multimedia. For example, the raw audio is difficult to get connected with its related text data to perform data analytics. Moreover, the user interface design and intuitiveness should also be improved by RapidMiner.

- **Hard to sharing RapidMiner analysis**
  However, there are some limitations of RapidMiner such as the sharing of RapidMiner Studio is difficult. It is more automated and useful to run models on a website. For example, it is even hard to share if the developers need to use it for Business Analytics dashboards.

2.1.3 Weka

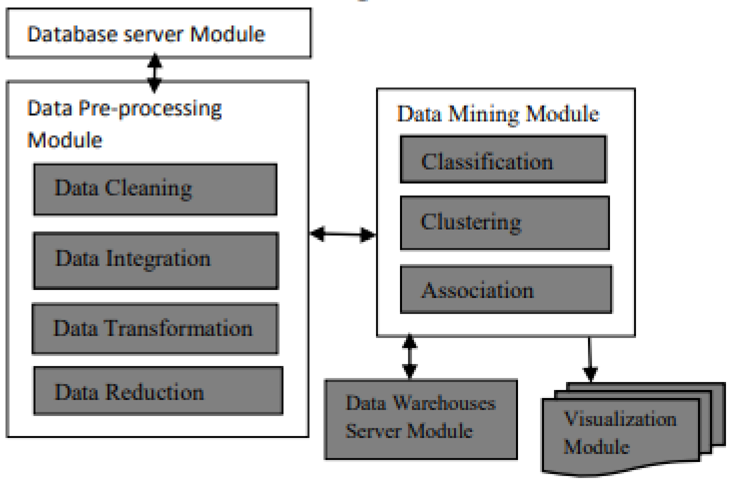


Figure 2.1.3 The workflow of WEKA would be as follows:
Data → Pre-processing → Data Mining → Knowledge [17]

Weka is a library of machine learning algorithms for use in data mining job [18]. The first step to perform data preprocessing in Weka is choose the "Preprocess" tab after launching the explorer window in Weka software. Next, the user requires to enter the type of attributes and classes contained in the data set. When the user successfully inserted the data set under "Filter" option, select the "Standardize" or "Normalize" filter and apply it to all attributes. It also allows user to interpret the graph and figure out which attribute discriminate best between the classes in the dataset [19].

Strengths of Weka:

- **Straightforward initial setup**
  The initial setup in Weka is easy and straightforward for the users. It is because all courses are provided in the first course of the Weka library. User can directly implement Weka solution along with Java for any customer via setting a dependency of their JAR file inside the project.
- **Great performance**
  Weka provides a comprehensive collection of data preprocessing and modeling techniques. It allows user to perform data preprocessing and modelling faster. Besides, Weka has stable performance which does not make any problems arise. It does not require any maintenance once it is correctly installed. In addition,

Drawbacks of Weka:

- **Slow to handle huge amount of data**
  The limitation of Weka is that it is not suitable to handle huge amount of data. For example, the hardware might need to spend some time to process the large data sets. Users need to divide the data into the smallest possible chunks and then ran these algorithms on that lowest possible data set.
- **Lack of transformation tools**
  In the filter section of Weka, it allows user to choose the method of filtering. However, the drawback is that it lacks some particular modification tools. For example, there is no functions to select if the user wishes to change the variable from numeric to categorical.

2.1.4 SQLite

SQLite is a lightweight, embedded, open-source relational database management system that is widely used in software applications for desktop and mobile platforms. It is self-contained and does not require a separate server process or administration, making it easy to deploy and manage.

Strengths:

- **High Performance**: It is a serverless database that stores data in a file on disk, it has a low overhead and can access data quickly. A benchmark study conducted by SQLite showed that it outperformed other popular embedded databases in terms of both read and write operations, demonstrating its strong performance capabilities.
- **Cross-platform compatibility**: SQLite is a cross-platform database that can be used on various operating systems, including Windows, Mac, Linux, and mobile platforms like Android and iOS.
- **High Reliability**: SQLite is highly reliable and provides robust data integrity, even in cases of power failure or system crashes. A study conducted by MIT found that SQLite had a low incidence of data corruption and was able to recover from failures quickly and reliably.
- **Cost-effective**: It does not require a separate server process or administration, further reducing costs. This affordability and ease of use have contributed to the popularity of SQLite in both commercial and non-commercial applications (Hipp, 2019).

Drawbacks:

- **Limited concurrency**: SQLite is not suitable for high-concurrency applications, as it uses a file-based locking mechanism to ensure data consistency. A study conducted by Oracle compared the performance of SQLite and Oracle Database under high-concurrency workloads and found that SQLite performed poorly in this scenario.
- **Limited scalability**: It stores all data in a single file, it can become slow and unwieldy when the file size grows beyond a certain point. Additionally, SQLite does not support distributed transactions or replication, which limits its ability to scale horizontally. A study conducted by Microsoft compared the

performance of SQLite and Microsoft SQL Server under a high-volume workload and found that SQLite was unable to handle the workload, whereas SQL Server performed well.

- **Limited functionality**: SQLite is a lightweight database that does not support many of the advanced features found in larger databases, such as stored procedures, triggers, or user-defined functions. This can limit its usefulness in certain types of applications that require these features. A study conducted by the University of Washington compared the feature sets of SQLite and MySQL and found that SQLite lacked many of the advanced features found in MySQL .
- **Lack of support**: Because SQLite is an open-source database, it may not have the same level of support and resources available as commercial databases. This can be a concern for developers who require timely support or bug fixes. However, SQLite has an active community of developers who provide support and contribute to its development.

2.1.5 Apache Parquet

Apache Parquet is an open-source data storage format that is designed for efficient and scalable processing of large datasets. It is columnar storage, meaning that it stores data in columns rather than rows, which allows for faster data retrieval and compression. Parquet was developed as part of the Apache Hadoop ecosystem, but it can be used with a variety of other data processing frameworks and tools.

Strengths:

- **High Performance**: According to a performance benchmark study by Dremio, Apache Parquet outperformed other file formats like ORC and Avro in terms of query execution time and CPU usage [1]. This makes Apache Parquet a great choice for big data applications that require high performance.
- **Flexibility**: Apache Parquet is a flexible format that can be used with a wide range of data processing frameworks, including Apache Spark, Apache Hive, and Apache Impala. This makes it easy to integrate with existing data processing pipelines and enables efficient data processing across different frameworks.
- **Cross-Platform Compatibility**: Apache Parquet is a cross-platform format that can be used with various programming languages, including Java, Python, and

C++. It can also be used on different operating systems, including Windows, Linux, and macOS. This makes it easy to use and deploy across different environments [2].

- **Schema Evolution**: Apache Parquet supports schema evolution, which means that the schema of the data can be modified without having to rewrite or reload the entire dataset. This can be especially useful in cases where data schemas change frequently, such as in data warehousing and analytics [3].

Drawbacks:

- **Write Performance**: Although Apache Parquet is optimized for read performance, it can have slower write performance compared to other file formats. This is because Parquet requires more CPU resources and memory when writing data, especially when encoding and compressing data [4].
- **Lack of Support for Updates and Deletes**: Apache Parquet is designed for efficient read-only data access and does not support updates or deletes. This means that any modifications to the data require rewriting the entire dataset, which can be time-consuming and resource-intensive [5].
- **Complex Data Types**: Apache Parquet does not support all complex data types, such as nested data structures or arrays. This can limit the flexibility of developers when designing complex data processing applications [6].
- **Difficulty in Debugging**: As a columnar data format, Apache Parquet can make it difficult to debug data processing errors, especially when working with large datasets. This is because it stores data in a column-wise format, which can make it challenging to locate errors in specific rows or records [4].

2.1.6 MongoDB

MongoDB is a well-known open-source document-oriented NoSQL database with a focus on scalability, performance, and agility. It offers a completely distinct technique for data storage and retrieval. This storage format is known as BSON. In MongoDB, a record is a document, which is a data structure made up of field and value pairs. Documents in MongoDB are identical to JSON objects.

Strengths:

- **Scalability**: MongoDB is highly scalable and can handle large amounts of data and high volumes of read and write operations. It uses a sharding mechanism to distribute data across multiple servers, allowing for horizontal scaling. According to a performance benchmark study by Pythian, MongoDB was able to scale up to 10 billion documents and 1 million write operations per second [7].
- **Flexibility**: MongoDB's flexible data model allows developers to store and query data in a way that best fits their application's needs. It supports dynamic schemas, allowing for the easy addition or removal of fields from documents. MongoDB also supports a variety of data types, including arrays, dates, and geospatial data [8].
- **High Performance**: MongoDB's architecture is optimized for high performance. It uses a document-oriented data model and stores data in a binary-encoded format called BSON. This makes it efficient for read and write operations, and it also supports indexing to further improve query performance. According to a performance benchmark study by EnterpriseDB, MongoDB outperformed other NoSQL databases like Cassandra and Couchbase in terms of query latency and throughput [9].
- **Availability and Durability**: MongoDB provides high availability and durability through features like replica sets and automatic failover. MongoDB also provides durability through write operations that are automatically recorded to a write-ahead log (WAL) and can be recovered in case of failure [10].

Drawbacks:

- **Data Consistency**: In MongoDB's default write concern, the acknowledgment is sent to the client before the write operation has been applied to all replicas, which can result in inconsistencies if one replica lags behind the others [11].
- **Complex Data Modeling**: MongoDB's flexible schema can make data modeling more challenging compared to traditional relational databases. In MongoDB, the structure of each document can be different, which can lead to inconsistencies and redundancies in the data.
- **Limited Transaction Support**: MongoDB provides support for multi-document transactions in some cases, but it has some limitations. For example, transactions are only supported on replica sets and sharded clusters with a replica set as the shard according to [11].

2.1.7 Pandas dataframe

Pandas DataFrame is a popular data manipulation tool that is built on top of the Python programming language. It is widely used in data science and data analysis tasks because of its flexibility, ease of use, and powerful features. With Pandas DataFrame, users can easily load, manipulate, and analyze data in a tabular format, similar to a spreadsheet. The tool provides a wide range of functions and methods for working with data, including filtering, sorting, grouping, and joining data. In addition, it integrates seamlessly with other Python libraries, such as NumPy and Matplotlib, to provide a comprehensive data analysis and visualization environment. Overall, Pandas DataFrame is a versatile tool that is essential for anyone working with data in Python.

Strengths:

- **Data Wrangling**: Pandas provides powerful tools for data wrangling, which includes cleaning, transforming, and manipulating data. This is achieved through its built-in functions and methods such as filtering, grouping, and pivoting. These features make it easier for analysts to work with large datasets efficiently.
- **High Performance:** Pandas is optimized for high performance and can handle large datasets efficiently. It achieves this through its use of NumPy arrays, which are fast and efficient data structures for numerical computations. Additionally, pandas uses vectorized operations, which perform computations on entire arrays instead of individual elements, further increasing performance.

Drawbacks:

- **Performance**: While pandas is generally fast and efficient, certain operations can be slow, especially when working with large datasets. For example, sorting and grouping operations can be computationally expensive, and pandas may not be the best tool for real-time data processing or very large datasets. In some cases, users may need to consider alternative tools such as Apache Spark or Dask to achieve better performance [17].
- **Data Integrity**: Another potential issue with pandas is that it can be prone to data integrity problems. For example, when manipulating data, users may accidentally introduce errors or inconsistencies, especially when working with complex data structures. Additionally, pandas does not perform data validation

by default, which means that users must manually check data for correctness [16].

- **Limited Functionality**: Although pandas provides a wide range of functionality for data manipulation and analysis, it may not be suitable for all tasks. For example, pandas does not provide built-in support for machine learning or statistical modeling, which may require additional libraries or tools. Additionally, pandas may not be the best choice for working with unstructured data or non-tabular data formats [18].

2.1.8 Journal 1: Pandas vs. Polars: A Syntax and Speed Comparison

In the field of data analytics, the choice of the right tools and libraries can greatly impact the efficiency and accuracy of data processing. The article "Pandas vs. Polars: A Syntax and Speed Comparison" by Romain Guillebert compares two popular data manipulation libraries in Python, namely Pandas and Polars, with a focus on their syntax and speed [26].

Strengths:

1. **Syntax Comparison**: The article provides a comprehensive syntax comparison between Pandas and Polars, highlighting the similarities and differences between the two libraries. This comparison can be helpful for users who are familiar with one library and want to learn the other or those who are considering which library to use for their data analysis tasks.
2. **Performance Benchmarking**: The article includes a performance benchmarking section that compares the speed of Pandas and Polars for various data manipulation tasks. The benchmarking results show that Polars outperforms Pandas in several tasks, such as group by aggregation and filtering, making it a promising alternative to Pandas for large-scale data processing [26].
3. **Memory Efficiency**: The article also compares the memory efficiency of Pandas and Polars, showing that Polars is more memory-efficient for certain data manipulation tasks. This can be especially important for users working with large datasets that require significant memory usage.

Drawbacks:

1. **Lack of Comparison with Other Data Storage and Processing Technologies** : The paper only compares Pandas with Polars, without benchmarking against other popular data analysis libraries like MongoDB, Apache Parquet, and SQLite. This limits the generalizability of the results and may not reflect how Pandas performs in comparison to other widely used libraries.
2. **No Exploration of Trade-offs**: The paper does not explore the trade-offs between performance and other important factors like memory usage, scalability, or ease of use. While performance is an important consideration, it is not the only factor that determines the suitability of a library for a given use case.

Overall, while the paper provides useful insights into the performance of Pandas and Polars for basic data analysis tasks, it is limited in its scope and may not fully represent the performance of Pandas in all possible scenarios. Further research that benchmarks Pandas against other popular data analysis libraries and explores the trade-offs between performance and other important factors would be beneficial.

2.1.9 Journal 2: A Comparison of HDFS Compact Data Formats: Avro Versus Parquet

In this study, the authors compare two popular HDFS data formats - Avro and Parquet - for their compactness and performance efficiency in storing and processing big data. In order to assess the performance of the data queries, file formats like Avro and Parquet are compared with text formats in this study. The effectiveness of various data query patterns has been examined. The tests described in this article have been conducted using CDH 5.4, an open-source Apache Hadoop distribution from Cloudera. Because of the advantages of binary data formats and compression, the results demonstrate that compact data formats (Avro and Parquet) require less storage space than plain text data formats. In addition, compared to text data formats like Avro, data queries from the column-based data format Parquet are quicker.

Strengths:

1. **Efficient Data Compression**: The study finds that both Avro and Parquet are effective in compressing large data sets, but Parquet provides better compression rates for certain data types, such as string and timestamp data. This makes Parquet a better choice for applications that require efficient storage and reduced storage costs.
2. **High Performance**: The study shows that both Avro and Parquet offer high performance in data processing [27]. However, Parquet outperforms Avro in scenarios that require complex querying and data analysis, making it a better choice for applications that demand high performance in these areas.
3. **Scalability**: The study demonstrates that both Avro and Parquet are highly scalable and can handle large data sets. However, Parquet's columnar storage design allows for better parallel processing, making it more scalable for certain use cases.

Drawbacks:

1. **Lack of Flexibility**: The study notes that both Avro and Parquet are rigid in their data schema definition, which can limit their flexibility in handling unstructured or semi-structured data. This may make them less suitable for applications that require handling data with changing or evolving schemas.
2. **Limited Data Type Support**: The study shows that both Avro and Parquet have limitations in supporting certain data types, such as nested or complex data

structures. This can pose challenges in handling complex data sets, making them less suitable for applications that require handling such data.

3. **Complex Data Processing**: The study finds that both Avro and Parquet require additional processing steps, such as schema inference or data projection, before data can be effectively queried or analyzed [27]. This can add complexity to the data processing pipeline and make them less suitable for applications that require rapid data analysis.

In summary, the study provides a comprehensive comparison of the Avro and Parquet data formats for big data management in HDFS. While both formats offer strengths in terms of efficient compression, high performance, and scalability, they also have limitations in terms of flexibility, data type support, and data processing complexity. The study's findings can help inform the selection of the most appropriate data format for specific big data applications.

2.1.10 Journal 3: Comparing Relational and NoSQL Databases for carrying IoT data

Storage and retrieval of sensory data is the vital bottleneck and establishes the boundary requirements for IoT services due to the huge quantity of database capacity and processing required, as well as the exponential growth and use of IoT devices. In order to address the question of which open source relational database or document database performs better than the other while handling IoT datasets containing either binary big objects or small-size IoT data records or documents, this article compares open source relational databases and document databases. This study compares the performance of MySQL and PostgreSQL, two SQL databases, and the NoSQL MongoDB database, which is the most widely used DMS.

Strengths:

1. **Comparative Analysis**: The study provides a comprehensive comparative analysis of two popular types of databases, relational and NoSQL, in handling IoT data, which can help in making an informed decision on selecting the appropriate database for a particular IoT application [28].
2. **IoT-specific Metrics**: The authors used specific IoT-related metrics to evaluate the performance of the databases, such as data ingestion rate, query execution time, and storage space efficiency, which is important for IoT applications. The authors used detailed evaluation criteria for the performance comparison, such as data consistency, query capabilities, and standardization. This provides a clear and objective framework for the evaluation of the databases.
3. **Real-world Experiment**: The authors conducted a real-world experiment using IoT data collected from a smart home, which makes the results more applicable to practical scenarios. This approach provides a more realistic assessment of the capabilities of these databases when handling IoT data.

Drawbacks:

1. **Lack of Standardization**: NoSQL databases lack standardization, which can make data management and migration more complex compared to relational databases. This can lead to difficulties in data integration and interoperability between different NoSQL databases.
2. **Limited Query Capabilities**: NoSQL databases often have limited query capabilities compared to relational databases, which can make it difficult to

perform complex data analysis [28]. This may require additional data processing tools or techniques to be used in conjunction with NoSQL databases to achieve desired results.

3. **Data Consistency**: In some cases, NoSQL databases sacrifice data consistency for performance and scalability, which can lead to data integrity issues. This is because NoSQL databases often use a distributed architecture where data is replicated across multiple nodes, which can result in inconsistencies in data values due to latency issues or network failures.

In conclusion, the journal provides valuable insights into the performance comparison between relational and NoSQL databases in handling IoT data. However, it is important to consider the drawbacks such as the lack of standardization, limited query capabilities, and potential data consistency issues when selecting a database model for IoT applications. These drawbacks highlight the need for further research and evaluation to determine the most suitable database model for specific IoT use cases.

2.1.11 Journal 4: Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments

As the use of IoT devices and applications continues to grow, the demand for efficient and scalable data management solutions also increases. In this study, the authors evaluated the performance of two popular databases, MongoDB and MySQL, for IoT data management in different cloud environments. This comparison is based on assessing how well the two types of databases perform when working with resources in cloud computing that have varying specifications and inserting and retrieving a sizable volume of IoT data. In order to estimate the reaction time given the size of the database and the requirements of the cloud instance, this research also suggests two prediction models and contrasts them.

Strengths:

1. **Comprehensive Evaluation**: The study evaluated the performance of MongoDB and MySQL databases in different cloud environments, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). This comprehensive evaluation provides insights into how the performance of these databases varies in different cloud environments.
2. **Detailed Methodology**: The study provided a detailed methodology for the performance evaluation of these databases, including the data set used, the query workload, and the hardware and software configurations [29]. This methodology can serve as a useful reference for future studies in this area.
3. **Performance Metrics**: The study evaluated the performance of these databases based on various performance metrics, including query execution time, throughput, and scalability. This provides a comprehensive view of the performance of these databases in different cloud environments.

Drawbacks:

1. **No Comparison of Data Model**: The study only compared the performance of MongoDB and MySQL in different cloud environments for IoT data management [29]. However, the study did not compare the data model of both databases, which is an important factor that can impact database performance in specific scenarios. The data model determines how data is stored, organized, and accessed in the database, and can affect the speed of data retrieval and

processing. In some cases, a database with a certain data model may perform better for a specific use case compared to another database with a different data model. Therefore, the lack of comparison of data model in this study limits the generalizability of the results and may not provide a complete understanding of the performance of both databases for IoT data management.

2. **Limited Range of Workload Types**: The study only evaluated the performance of MongoDB and MySQL using a limited range of workload types, such as read-heavy and write-heavy workloads. This may not be representative of all possible workload scenarios and could limit the applicability of the study's findings. Workload types can vary widely based on the application and use case. In this study, the authors only evaluated read-heavy and write-heavy workloads, which are common workload types for IoT data management. However, there are other workload types, such as mixed workloads, that could impact the performance of the databases differently. Therefore, the limited range of workload types tested in this study may not be representative of all possible scenarios and may limit the applicability of the study's findings.
3. **Limited Cloud Environment:** The study only evaluated the performance of MongoDB and MySQL in one cloud environment, which may not be representative of other cloud environments that are used in practice. Cloud environments can vary widely in terms of architecture, performance, and features. In this study, the authors only evaluated the performance of MongoDB and MySQL in a limited number of cloud environments, such as AWS and GCP. However, there are other cloud environments, such as Microsoft Azure and IBM Cloud, that could impact the performance of the databases differently. Therefore, the limited number of cloud environments tested in this study may not be representative of all possible scenarios and may limit the applicability of the study's findings.

2.2 Proposed Solution and Conventional Approach

Conventional Approach

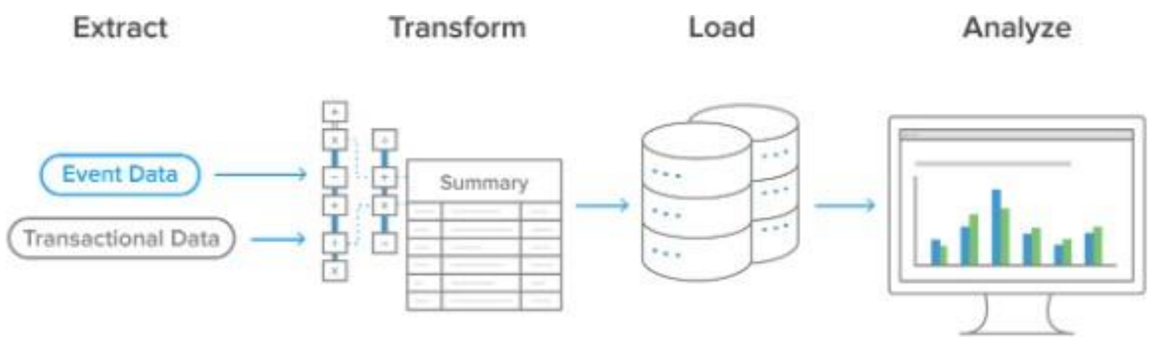


Figure 2.2.1 Traditional process for data extraction, transformation, and loading.[21]

The process of ETL (Extract, Transform, Load) involves the extraction of data from source systems, transforming it to meet the desired format or structure, and then loading it into a target system, such as a data warehouse or a database as shown in Figure 2.2.1. ETL implies that data must be populated before it can be queried depends on the specific context.

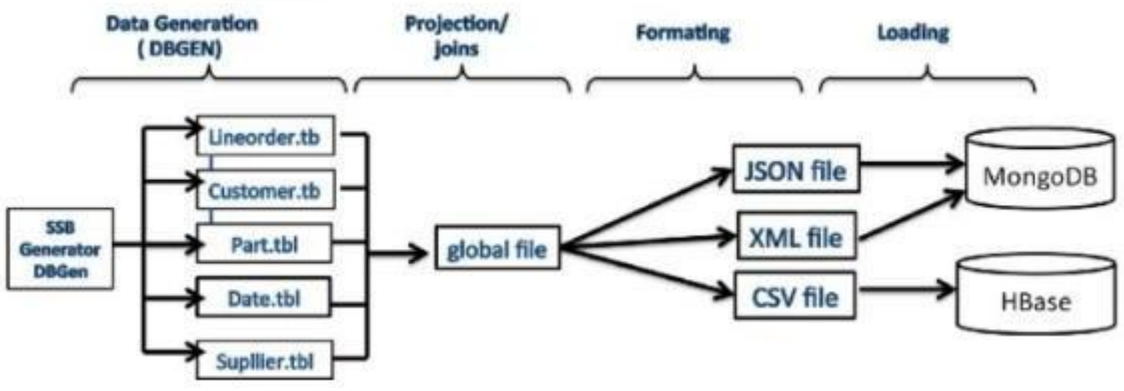


Figure 2.2.2 Visualization of populating data of existing NoSQL database.[a]

The existing data tools need data to be populated before the data can be query. The process of populating data before it can be queried typically depends on the data storage and management system used. Most of the NoSQL databases will require following steps :

i. Create Collections: In MongoDB, collections have to create to store data.

ii. Document Insertion: Insert JSON-like documents into collections. Each document can have different fields.

iii. Data Transformation: Users can transform and structure data within documents depending on the needs.

Figure 2.2.2 displayed the steps of populating data of MongoDB and HBase which includes the steps of joining few different files into one global file, and then formatting the file into accepted file extension like JSON, XML or CSV. After that, the data only able to load into MongoDB and HBase.

Proposed Solution

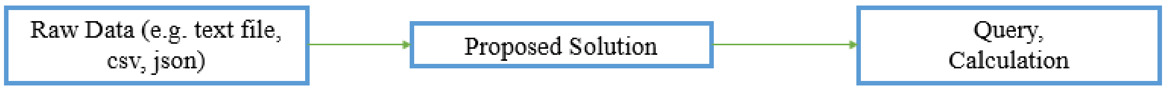


Figure 2.2.3 Visualization of proposed system

The proposed system aims to develop a preprocessing program that eliminates the need for data population before querying as visualized in Figure 2.2.3. This program will facilitate data querying and calculations directly from unpopulated data and provide accurate results. Additionally, the proposed solution will enable data to be processed directly without requiring data formatting. It will have the capability to handle raw data in inconsistent formats, such as text files, CSV files, and JSON files, each time data querying is performed. The program will empower developers to conduct various data operations, including querying, searching, matching, and calculations, on unstructured data types. Eliminating the requirement to populate large volumes of raw data before conducting queries significantly reduces the time consumed by the entire process. This approach makes it effortless for developers to extract data for data analytics purposes.

2.3 Comparison Between Reviewed Systems and Proposed Solutions

	Create Session and Data Frame	Defined Schema	Pre-Formatting Raw Data	Establish Database Connection
Apache Spark	√	√		
RapidMiner		√	Varies	
Weka		√	√	
MongoDB				√
SQLite			√	√
Pandas DataFrame	√		Varies	
Apache Parquet			√	
Proposed Solution				

Table 2.3.1 Comparison between existing software and proposed solution.

As illustrated in Table 2.3.1, the proposed solution reads and processes data from files without the need to create sessions, define schemas, perform extensive pre-formatting, or establish database connections. It is primarily designed for searching, extracting, and calculating mean values from various types of files, offering a minimal number of steps before querying raw data.

The table demonstrates that Apache Spark and Pandas DataFrame require the initialization of a session and data frame. Apache Spark is a distributed data processing framework that utilizes the concept of a session to manage resources and operations. Additionally, Pandas requires the creation of a DataFrame when used to work with structured data in a tabular format, as it is a Python library for data manipulation and analysis.

The second criteria compared between existing software and the proposed solution are related to database schemas. As indicated by [22], a database schema refers to a blueprint or architecture that describes the structure of the data. Thus, tools like Apache Spark, RapidMiner, and Weka typically work with structured data and may require a defined schema or structure for data insertion, although they may offer some flexibility in how the data is organized or processed within that schema.

Furthermore, Weka, SQLite, and Apache Parquet require pre-formatting of raw data before loading it, whereas RapidMiner and Pandas DataFrame may or may not require pre-formatting, depending on the data structure and analysis goals. Firstly, SQLite is a relational database system, and although it can accommodate unstructured data to some extent, it generally requires a defined schema for structured data. Pre-formatting or defining a schema is often necessary for organized storage and querying. Next, Apache Parquet is a columnar storage format used for structured data, so raw data often needs to be pre-structured and converted into a columnar format. This typically involves pre-processing the data into a compatible schema. Some algorithms in Weka may also require specific data formats or preprocessing steps.

Additionally, MongoDB and SQLite require connections to the database to load and query data. MongoDB is a NoSQL database that requires a database connection to load and query data stored within it. Similarly, SQLite is a relational database that loads and queries data stored in SQLite databases via a database connection.

In conclusion, the proposed solution does not necessitate data population before it can be queried, unlike existing tools. Furthermore, it allows data to be run directly without requiring data formatting.

Chapter 3

System Methodology

3.1 System Methodology

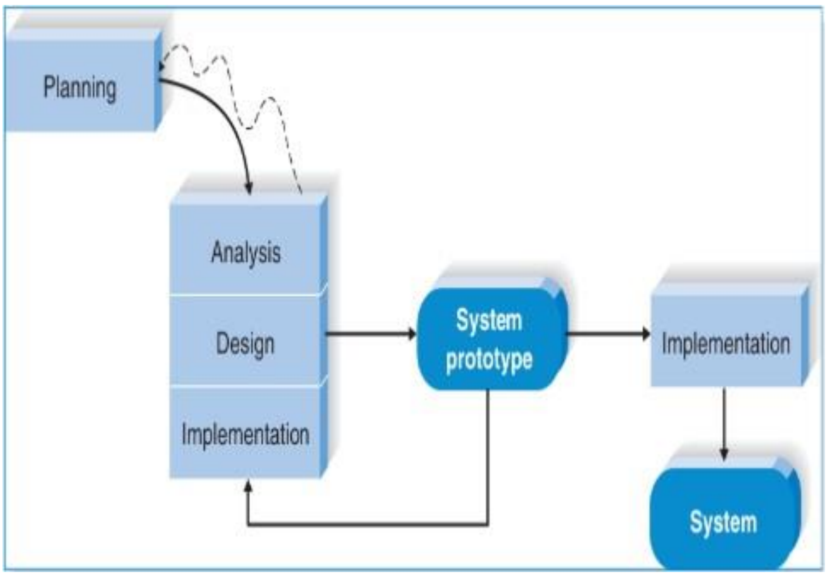


Figure 3.1.1 System Prototyping methodology [23]

The system methodology chosen for this project is rapid application development (RAD). According to [24], RAD refers to a methodology that emphasizes on implementing applications rapidly through multiple iterations and constant feedback. In the planning phase, stakeholders define project specifications, scope, and user requirements, including data storage preferences and extraction methods.

The analysis phase finalizes requirements, leading to the design phase, where diagrams are created using tools like PowerPoint. Subsequently, the initial model and two prototypes are developed. The first prototype assesses query execution times for raw unstructured data across different data tools, while the second prototype focuses on developing the proposed solution.

Stakeholder feedback guides the iterative process, cycling through analysis, design, and implementation phases until the final prototype fully aligns with user requirements. This RAD approach, supported by a Gantt Chart in Figure 3.1.2, ensures efficient progress toward IoT database project objectives.

3.1.1 Timeline

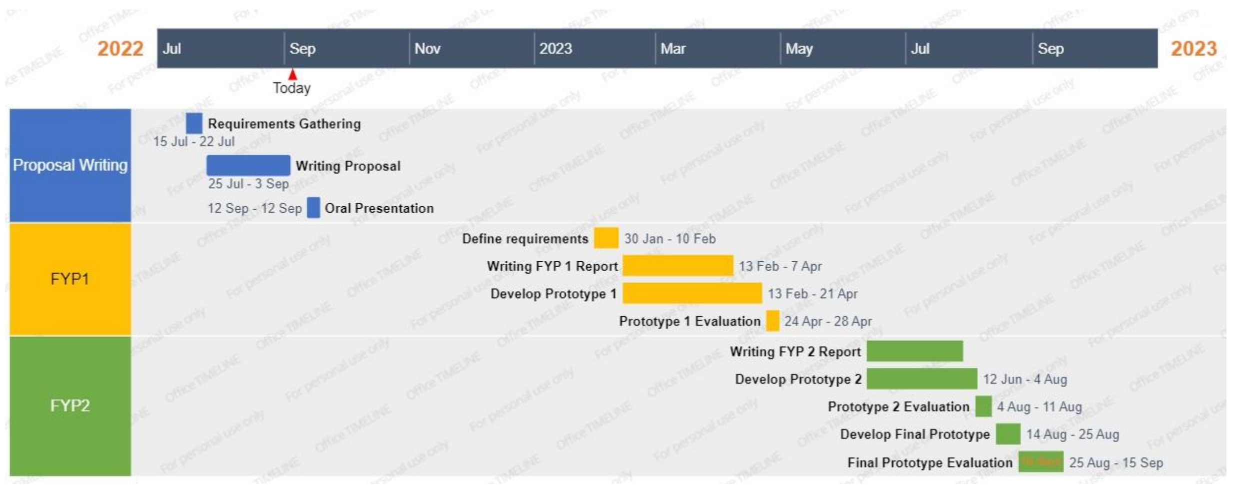


Figure 3.1.2 Gantt Chart

The Gantt Chart helps to schedule and visualize tasks and activities over time in this FYP.

3.1.2 System Architecture Diagram

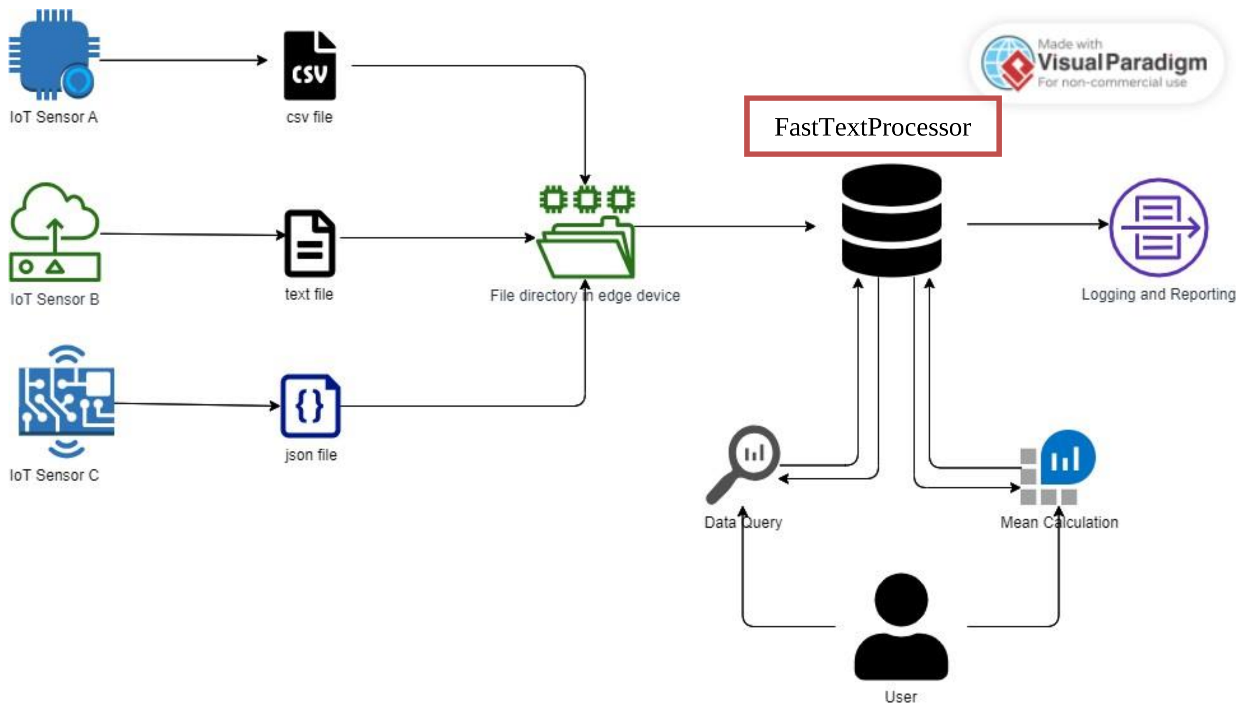


Figure 3.1.2 System architecture diagram of FastTextProcessor

In Figure 3.1.2, multiple files from diverse sources, including IoT sensors and online datasets, reside in a shared directory. These files encompass various formats like CSV, JSON, and text. Users employ FastTextProcessor, leveraging Python packages, to query and calculate means. FastTextProcessor directly accesses data, processes it, and presents results via the user interface. Additionally, results are logged for reference and error detection

3.1.3 Use Case Diagram and Description

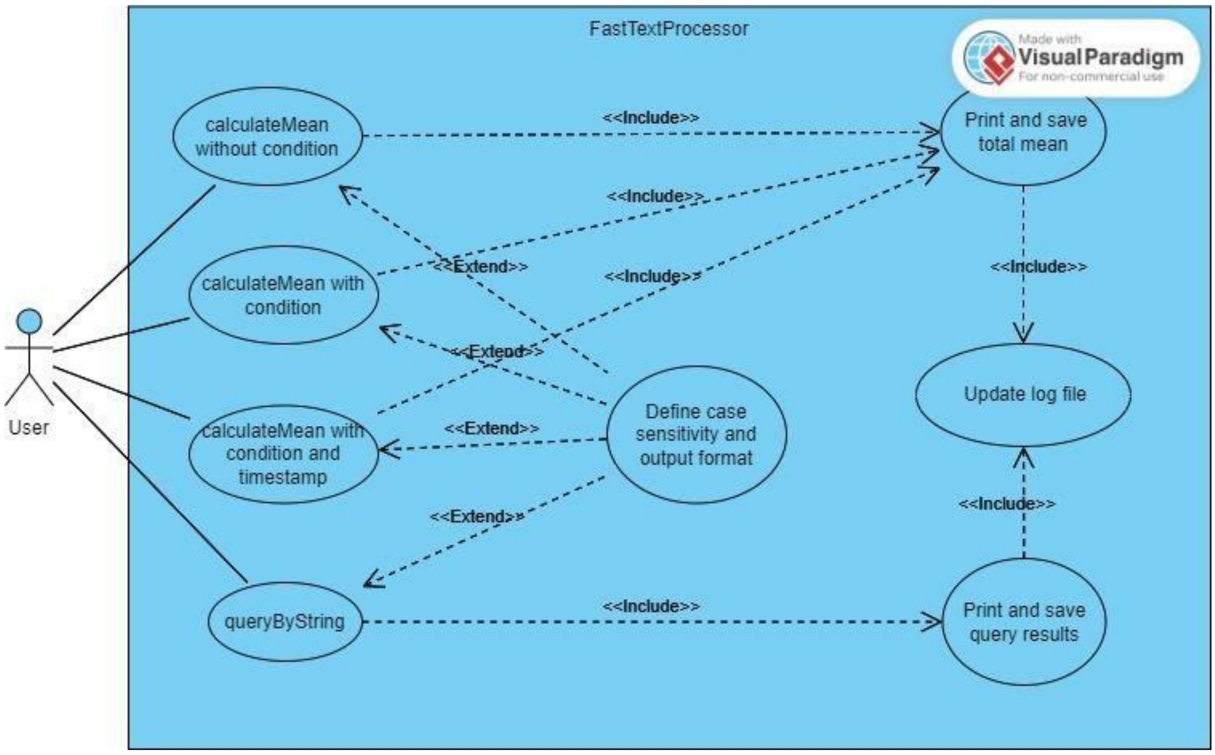
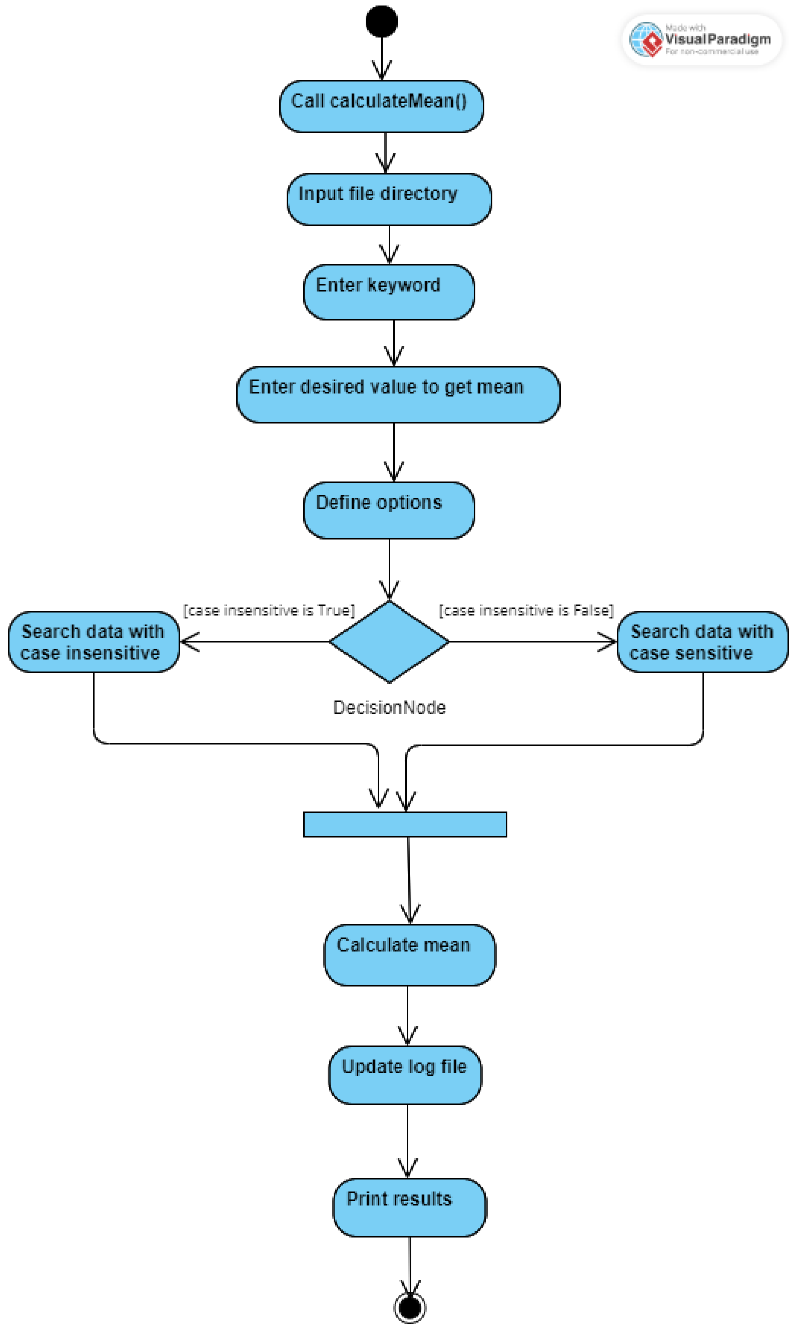

FastTextProcessor
Made with Visual Paradigm For non-commercial use
calculateMean without condition
calculateMean with condition
calculateMean with condition and timestamp
queryByString
User
Define case sensitivity and output format
Print and save total mean
Update log file
Print and save query results
<<Include>>
<<Extend>>

Figure 3.1.3.1 Use case diagram of FastTextProcessor.

Figure 3.1.3.1 shows that the user can perform 4 actions which are query by string, calculate mean without condition, get mean value with condition, and also calculate average value with specified condition and timestamp. Users can also choose to define case sensitivity and output format in either JSON format or text file. If users leave them blank, the default value of true case insensitivity and save output format as text file will be used. After calculating the desired mean value, the results will be printed and saved. Similarly, the match results of query by string will also printed and saved. Lastly, the results produced will then be recorded and updated in the log file.

3.1.4. Activity Diagram

i. Calculate mean without condition module.



Made with Visual Paradigm For non-commercial use
Call calculateMean()
Input file directory
Enter keyword
Enter desired value to get mean
Define options
[case insensitive is True]
[case insensitive is False]
Search data with case insensitive
Search data with case sensitive
DecisionNode
Calculate mean
Update log file
Print results

Figure 3.1.4.1 Activity diagram of calculate mean without condition.

The figure 3.1.4.1 reveals the process from users call the function of calculateMean(), and then enter the values like file directory path, search keyword, and value that the users wish to get the average value. After running the code, it will execute and calculate based on the inputs including the case sensitivity as defined in options dictionary. After that, the mean value will be computed, then the log file will be updated, and the results printed according to the output format specified or default value.

ii. Calculate mean with condition module.

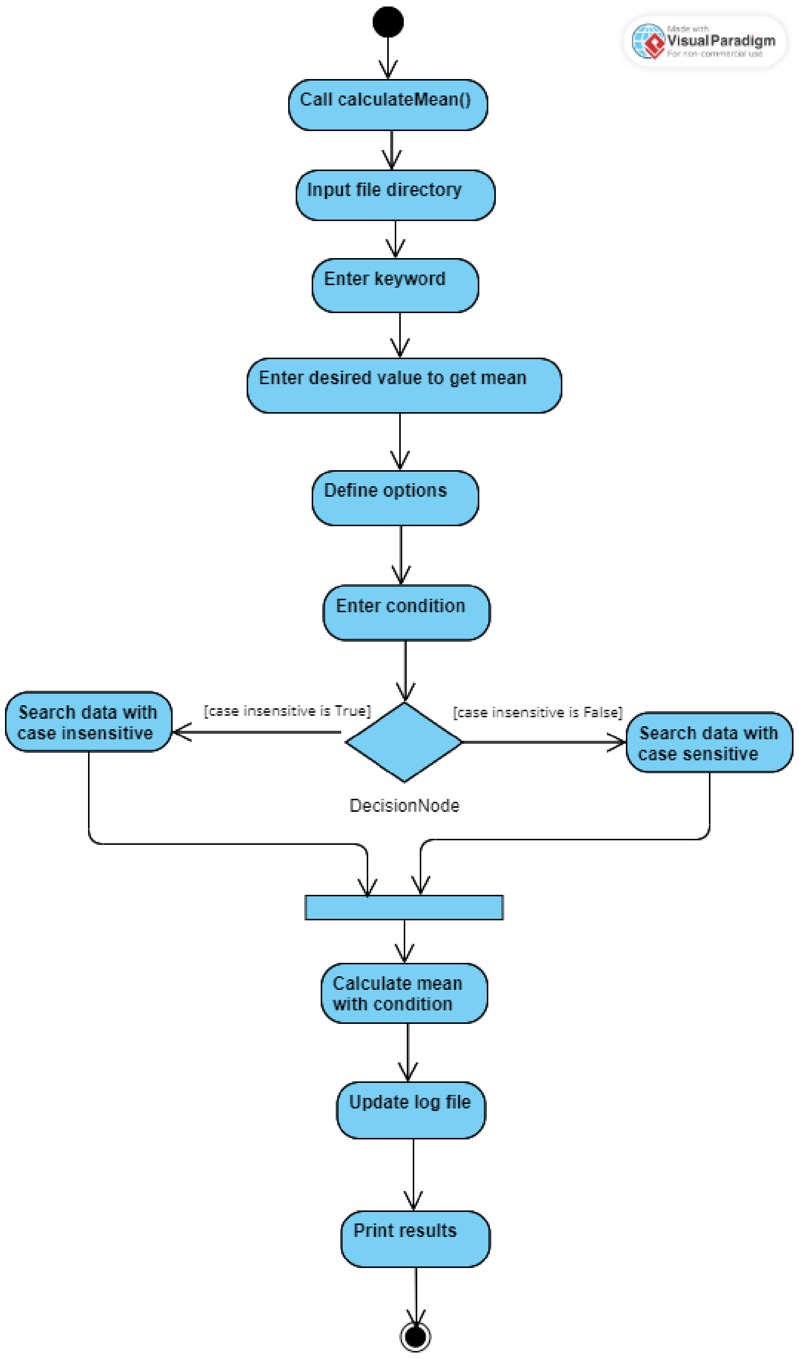
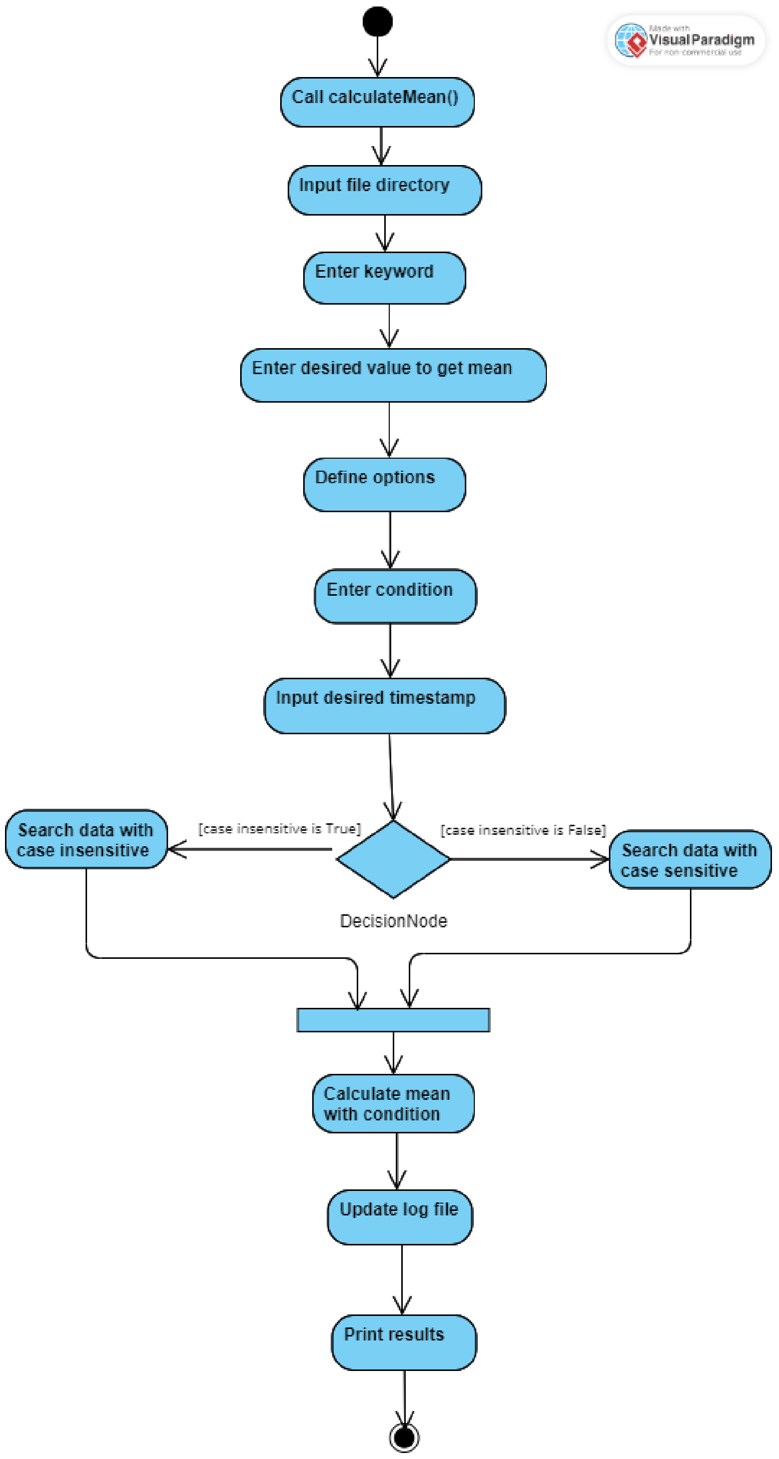

Made with Visual Paradigm For non-commercial use
Call calculateMean()
Input file directory
Enter keyword
Enter desired value to get mean
Define options
Enter condition
Search data with case insensitive
[case insensitive is True]
[case insensitive is False]
Search data with case sensitive
DecisionNode
Calculate mean with condition
Update log file
Print results

Figure 3.1.4.2 Activity diagram of calculate mean with condition.

Figure 3.1.4.2 describes the similar process as the Figure 3.1.4.1 but there is an extra step which is the user have to enter the desired condition. After that, the filtered results will be printed.

iii. Calculate mean with condition and timestamp module.



Made with Visual Paradigm For non-commercial use
Call calculateMean()
Input file directory
Enter keyword
Enter desired value to get mean
Define options
Enter condition
Input desired timestamp
[case insensitive is True]
[case insensitive is False]
Search data with case insensitive
Search data with case sensitive
DecisionNode
Calculate mean with condition
Update log file
Print results

Figure 3.1.4.3 Activity diagram of calculate mean with condition and timestamp.

Figure 3.1.4.3 shows the similar process as the previous modules. However, there is some differences where the user can enter the desired condition and also the specific timestamp. After that, the final results will be printed out.

iv. Query by string module.

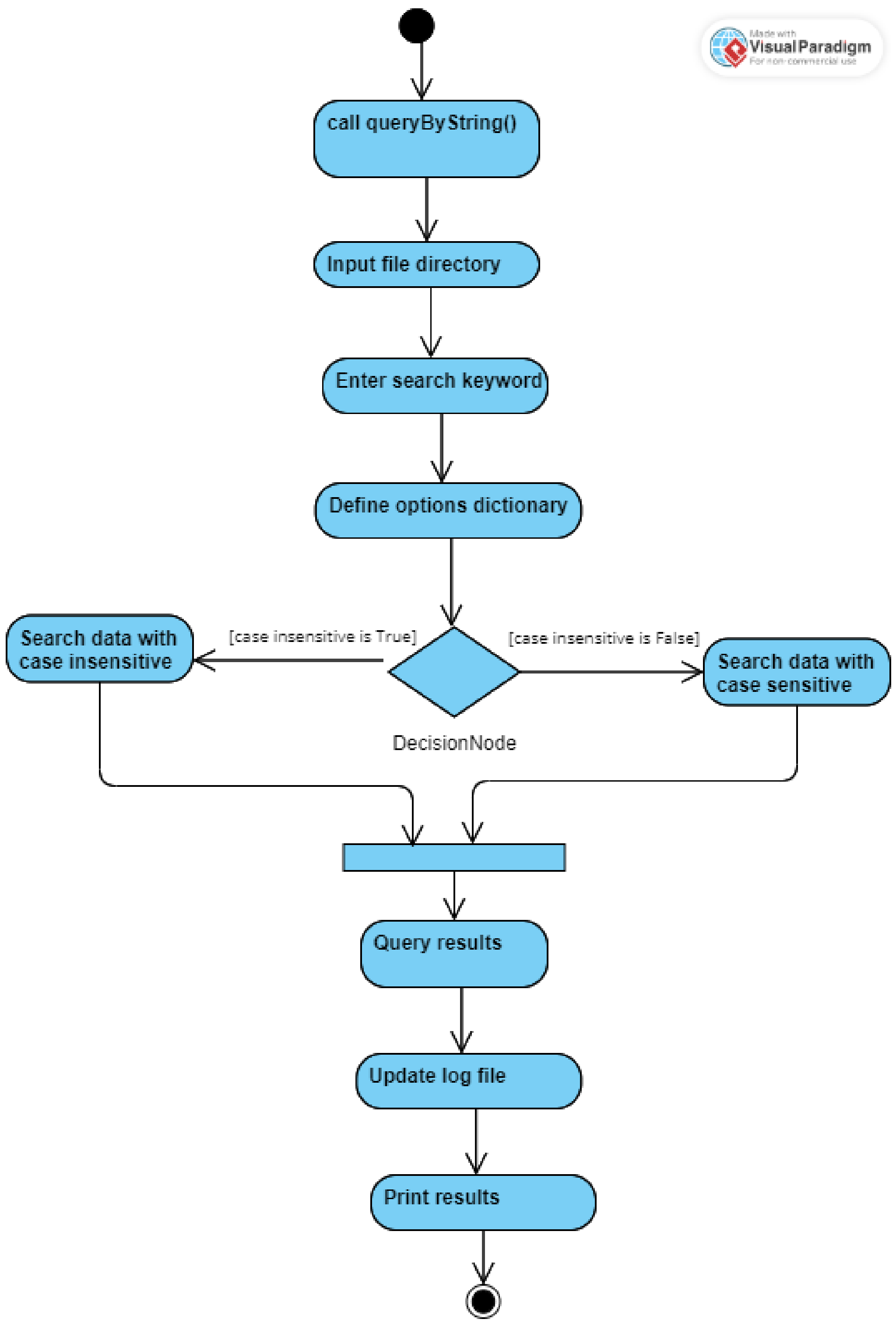

call queryByString()
Input file directory
Enter search keyword
Define options dictionary
Search data with case insensitive
[case insensitive is True]
[case insensitive is False]
Search data with case sensitive
DecisionNode
Query results
Update log file
Print results

Figure 3.1.4.4 Activity diagram of query by string.

Figure 3.1.4.4 reveals that users are able to search the desired keyword and get the relevant results from the data. The search results will be updated in the search results file and the search log file will record the details after the execution of the code.

Chapter 4

System Design

4.1 System Block Diagram

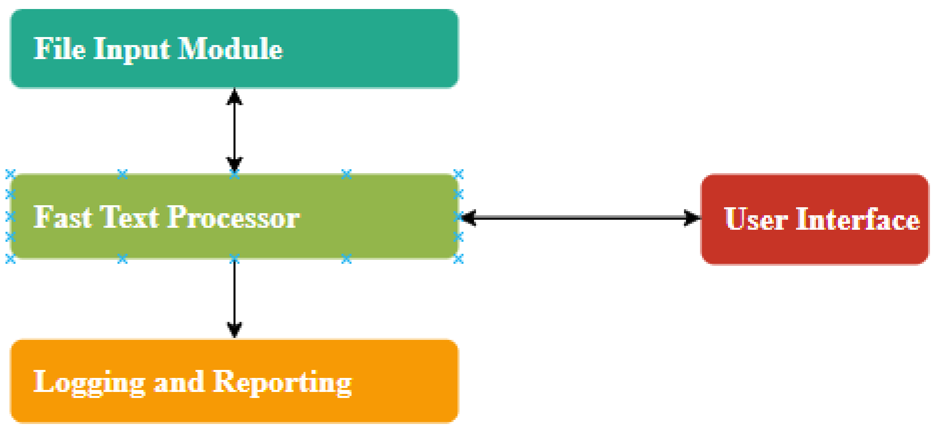


Figure 4.1 System block diagram

The system block diagram in Figure 4.1 provides an overview of the major functional blocks within the system and how they interact. Key components in the system block diagram:

a. File Input Module: Responsible for acquiring data files from a specified directory.
b. Fast Text Processor: Performs data query and calculation operations.
c. Logging and Reporting: Records system activities and reports results.
d. User Interface: Allows users to configure search queries and interact with the system.

These components collaborate to achieve the system's objectives.

4.2 System Components Specifications

This system is designed to process and analyze data from various sources, including JSON, CSV, and text files. It consists of several key components that work together to achieve its goals. Detailed specifications of system components are essential for understanding their roles and capabilities. Here are specifications for some key components:

a. File Input Module**:**

Function: Loads data files from a file directory specified by users.

Supported Formats: JSON, CSV, text.

Interaction: Communicates with Fast Text Processor.

b. Fast Text Processor:

Function: Processes and matches data from various file formats.

Subcomponents: JSON Data Processor, CSV Data Processor, Text Data Processor

Interaction: Communicates with File Input Module, Logging and Reporting, and User Interface.

c. Logging and Reporting:

Function: Logs system activities and reports results.

Output Formats: Log files for auditing and error detection.

d. User Interface:

Function: Allows users to input search queries and configure options.

Interaction: Communicates with Data Processing Engine.

4.3 System Components Interaction Operations

Connections:

a. File Input Module communicates with the Data Processing Engine to provide data files for processing.
b. Data Processing Engine communicates with the Logging and Reporting component to log activities and results.
c. User Interface interacts with the Data Processing Engine to configure search options and initiate data processing.

Data Flow:

1. Data files flow from the File Input Module to the Data Processing Engine for processing and matching.
2. Matched data and search results flow from the Data Processing Engine to the Logging and Reporting component for storage and reporting.
3. Users interact with the User Interface to input search queries and configure search options.

This simplified system design diagram provides an overview of the major components and their interactions within the data processing and calculation system.

4.3.1 Flow Diagram in Data Query

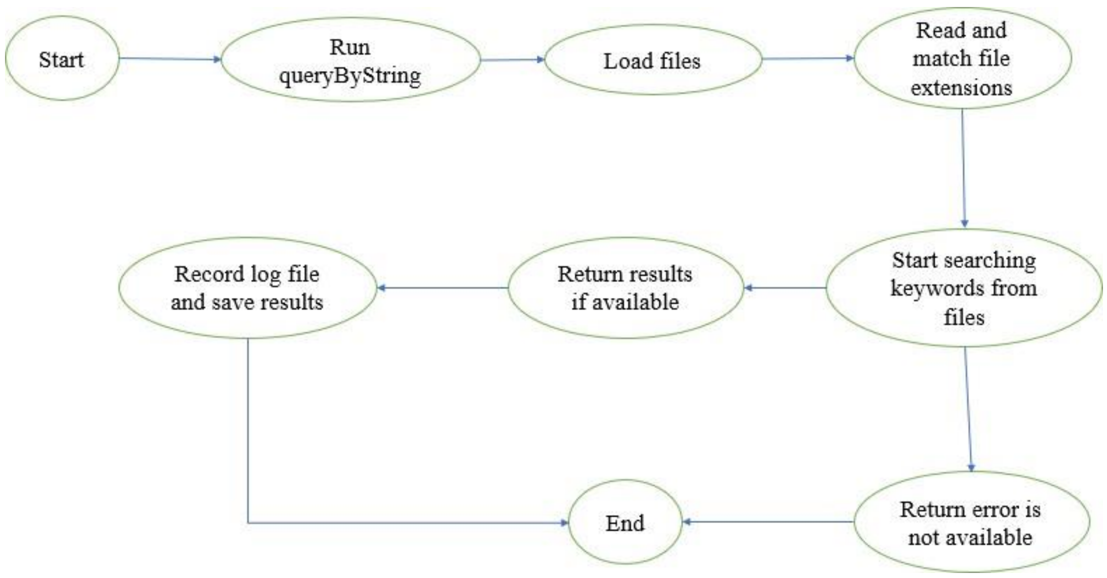

Start
Run queryByString
Load files
Read and match file extensions
Start searching keywords from files
Return results if available
Record log file and save results
Return error is not available
End

Figure 4.3.1 Flow Diagram in Data Query

The flow diagram in Figure 4.3.1 shows that the workflow when the data query is performed by user.

4.3.2 Flow Diagram in Mean Calculation

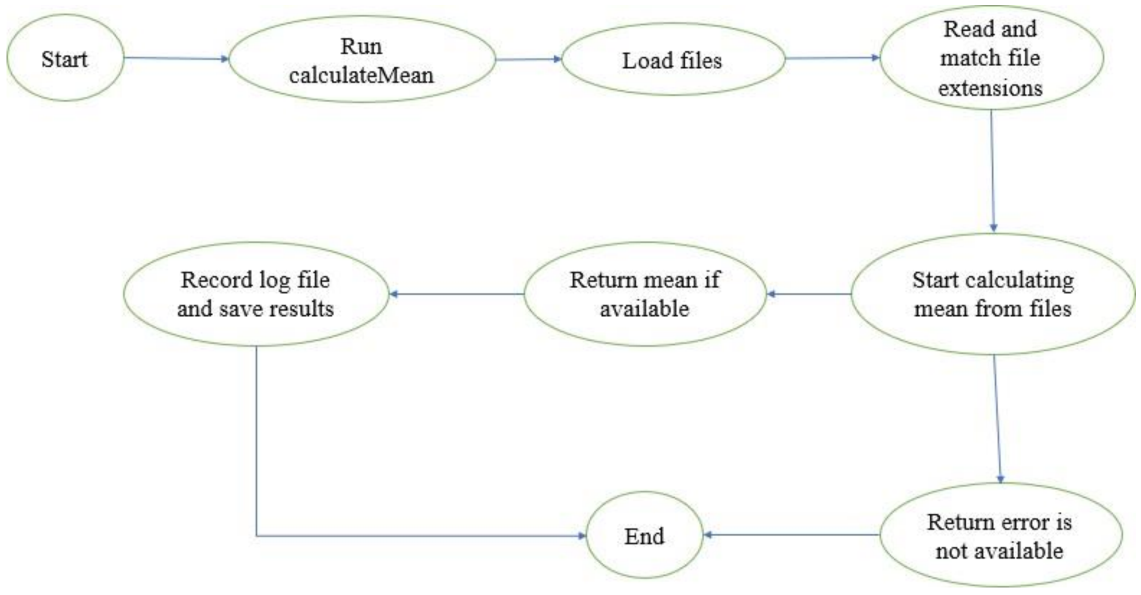

Start
Run calculateMean
Load files
Read and match file extensions
Start calculating mean from files
Return mean if available
Record log file and save results
Return error is not available
End

Figure 4.3.2 Flow Diagram in Mean Calculation

From the figure 4.3.2, it clearly displays the workflow when the user executes the function to compute the average value from the files.

Chapter 5

System Implementation

5.1 Hardware Setup

Desktop:

- Used to develop and debug Fast Text Processor with Spyder.

Description	Specifications
Model	Dell Inspiron 3480
Processor	Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
Operating System	Windows 10
Installed RAM	8.00 GB (7.88 GB usable)
System type	64-bit operating system, x64-based processor

Table 5.1 Specification of desktop

5.2 Software Setup

1. Spyder

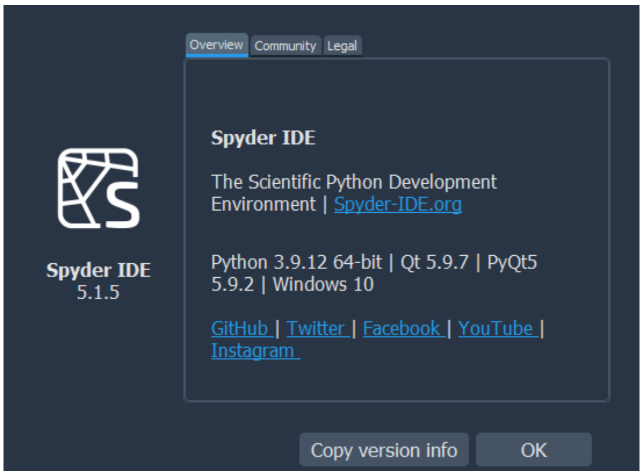
- Develop the Fast Text Processor using Python in Spyder

- Spyder version: 5.1.5 None

- Python version: 3.9.12 64-bit

- Qt version: 5.9.7

- PyQt5 version: 5.9.2



Spyder IDE

The Scientific Python Development Environment | Spyder-IDE.org

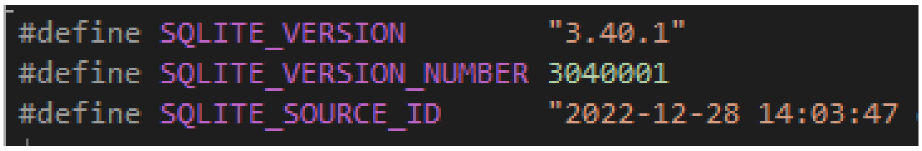
Python 3.9.12 64-bit | Qt 5.9.7 | PyQt5 5.9.2 | Windows 10

GitHub | Twitter | Facebook | YouTube | Instagram

Figure 5.2.1 Setup of Spyder

2. SQLite

- Used to insert raw data into SQLite, then retrieve data and execute mean computations.
- Time taken of the operations are measured.
- Version: 3.40.1

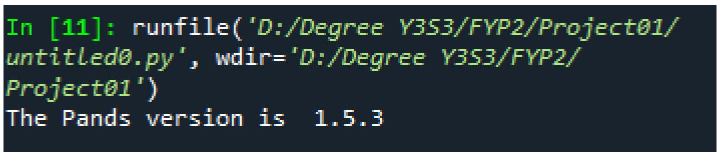


```
#define SQLITE_VERSION        "3.40.1"
#define SQLITE_VERSION_NUMBER 3040001
#define SQLITE_SOURCE_ID      "2022-12-28 14:03:47
```

Figure 5.2.2 Setup of SQLite

3. Pandas

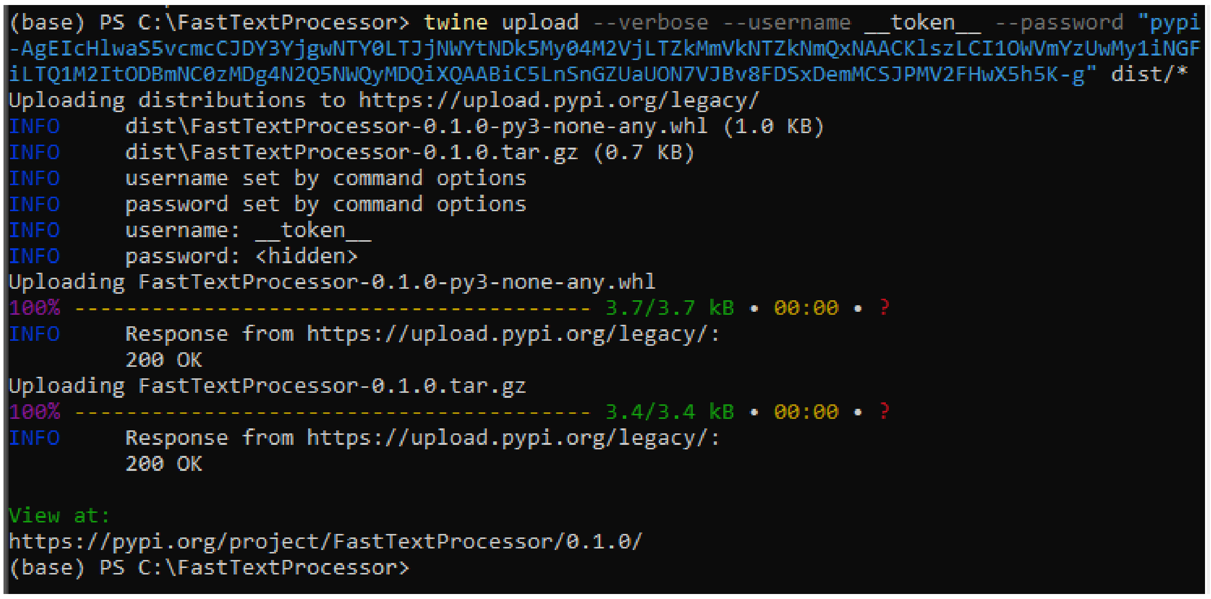
- Used to load raw data into the Pandas DataFrame and perform data query and calculation.
- Time taken of the operations are measured.
- Version: 1.5.3



```
In [11]: runfile('D:/Degree Y3S3/FYP2/Project01/
untitled0.py', wdir='D:/Degree Y3S3/FYP2/
Project01')
The Pands version is  1.5.3
```

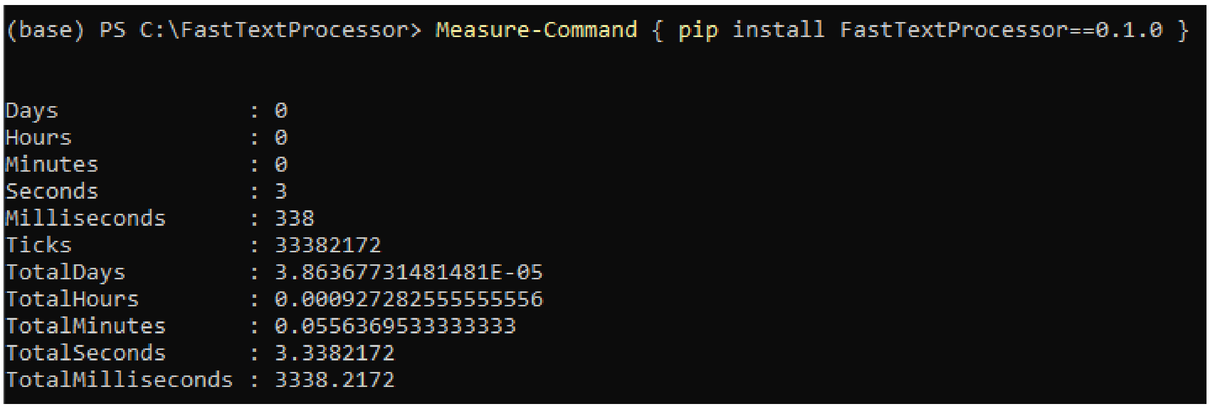
Figure 5.2.3 Setup of Pandas

5.3 System Operation (with Screenshot)



```
(base) PS C:\FastTextProcessor> twine upload --verbose --username __token__ --password "pypi
-AgEIcHlwaS5vcmcCJDY3YjgwNTY0LTJjNWYtNDk5My04M2VjLTZkMmVkNTZkNmQxNAACKlszLCI1OWVmYzUwMy1iNGF
iLTQ1M2ItODBmNC0zMDg4N2Q5NWQyMDQiXQAABiC5LnSnGZUaUON7VJBv8FDSxDemMCSJPMV2FHwX5h5K-g" dist/*
Uploading distributions to https://upload.pypi.org/legacy/
INFO     dist\FastTextProcessor-0.1.0-py3-none-any.whl (1.0 KB)
INFO     dist\FastTextProcessor-0.1.0.tar.gz (0.7 KB)
INFO     username set by command options
INFO     password set by command options
INFO     username: __token__
INFO     password: <hidden>
Uploading FastTextProcessor-0.1.0-py3-none-any.whl
100% ---------------------------------------- 3.7/3.7 kB • 00:00 • ?
INFO     Response from https://upload.pypi.org/legacy/:
         200 OK
Uploading FastTextProcessor-0.1.0.tar.gz
100% ---------------------------------------- 3.4/3.4 kB • 00:00 • ?
INFO     Response from https://upload.pypi.org/legacy/:
         200 OK

View at:
https://pypi.org/project/FastTextProcessor/0.1.0/
(base) PS C:\FastTextProcessor>
```

Figure 5.3.1.1 Successfully upload FastTextProcessor to PyPI



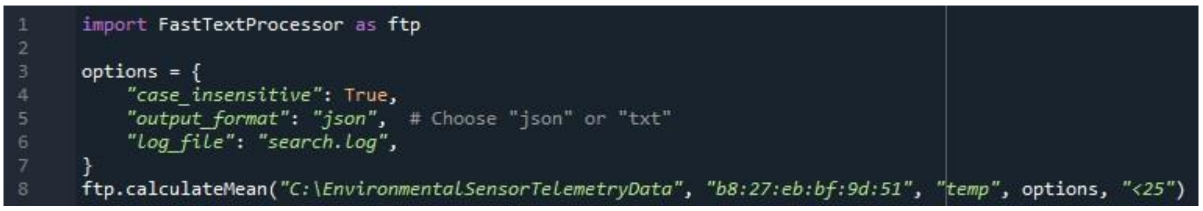
```
(base) PS C:\FastTextProcessor> Measure-Command { pip install FastTextProcessor==0.1.0 }


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 3
Milliseconds      : 338
Ticks             : 33382172
TotalDays         : 3.86367731481481E-05
TotalHours        : 0.000927282555555556
TotalMinutes      : 0.0556369533333333
TotalSeconds      : 3.3382172
TotalMilliseconds : 3338.2172
```

Figure 5.3.1.2 Time taken for installation of FastTextProcessor

The python package of FastTextProcessor was uploaded to Python Package Index by using twine as shown in Figure *5.3.2*. Other than FastTextProcessor.py, the uploaded package also included_init_.py, setup.py and README.md. Next, Figure *5.3.2* shows that the packages can be installed successfully.

5.3.1 FastTextProcessor Operations

1. Calculate mean value for sensor "b8:27:eb:bf:9d:51" where its temperature is less than 25.



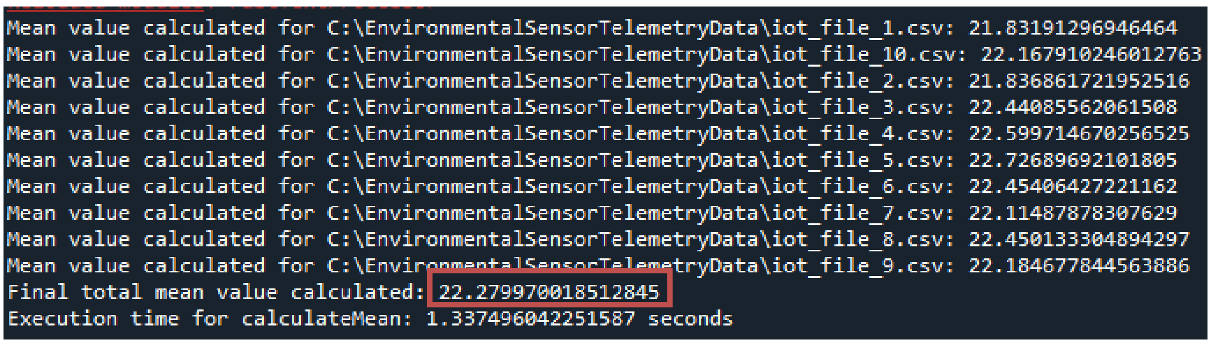
```
import FastTextProcessor as ftp

options = {
    "case_insensitive": True,
    "output_format": "json",   # Choose "json" or "txt"
    "log_file": "search.log",
}
ftp.calculateMean("C:\EnvironmentalSensorTelemetryData", "b8:27:eb:bf:9d:51", "temp", options, "<25")
```

Figure 5.3.1.3: Only 7 lines of code are required to get the mean value by FastTextProcessor

The Figure *5.3.1.3* above illustrates the simplicity and efficiency of the library for calculating the mean value that the user can achieve this with just a few lines of code. The code begins by defining a dictionary named "options" with specific settings that affect the behavior of the function. These settings include case sensitivity, output format and log file. The "case_insensitive" setting is set to True. This means that any text comparisons performed within the function will treat text as case insensitive. For instance, "temp" and "TEMP" would be considered equivalent. Furthermore, the "output_format" is specified as "json." This indicates that the results of the function will be formatted in JSON (JavaScript Object Notation). By using this library, the user can choose to either saved the output format as JSON format or text file by specifying the output_format as “json” or “txt”. In addition, the "log_file" parameter is set to "search.log." This defines the name of the log file where the function will record log messages and information about its operation. Moreover, the code then provides an example of how to use the calculateMean function with these settings. The function is called with the following arguments:

i. **File Path**: The first argument is the file path "C:\EnvironmentalSensorTelemetryData." This specifies the directory where the function will search for data files to process.
ii. **Search Keyword**: The second argument is "b8:27:eb:bf:9d:51." It represents a unique identifier for a sensor device. The user can also search for other variables other than sensor. However, it's important to note that the user can customize this search keyword to search for data as needed. This flexibility allows the user to analyze data from different sources by specifying the relevant keyword.
iii. **Data Attribute**: The third argument is "temp," which indicates that user want to calculate the mean value for the "temperature" data attribute within the files.
iv. **Options** Dictionary: The fourth argument is the "options" dictionary that defined earlier, allowing user to control the function's behavior according to the specified settings.
v. **Condition**: Finally, the fifth argument is "<25," implying that user want to calculate the mean value only for temperature readings that are less than 25. User can also search for the mean value without condition.

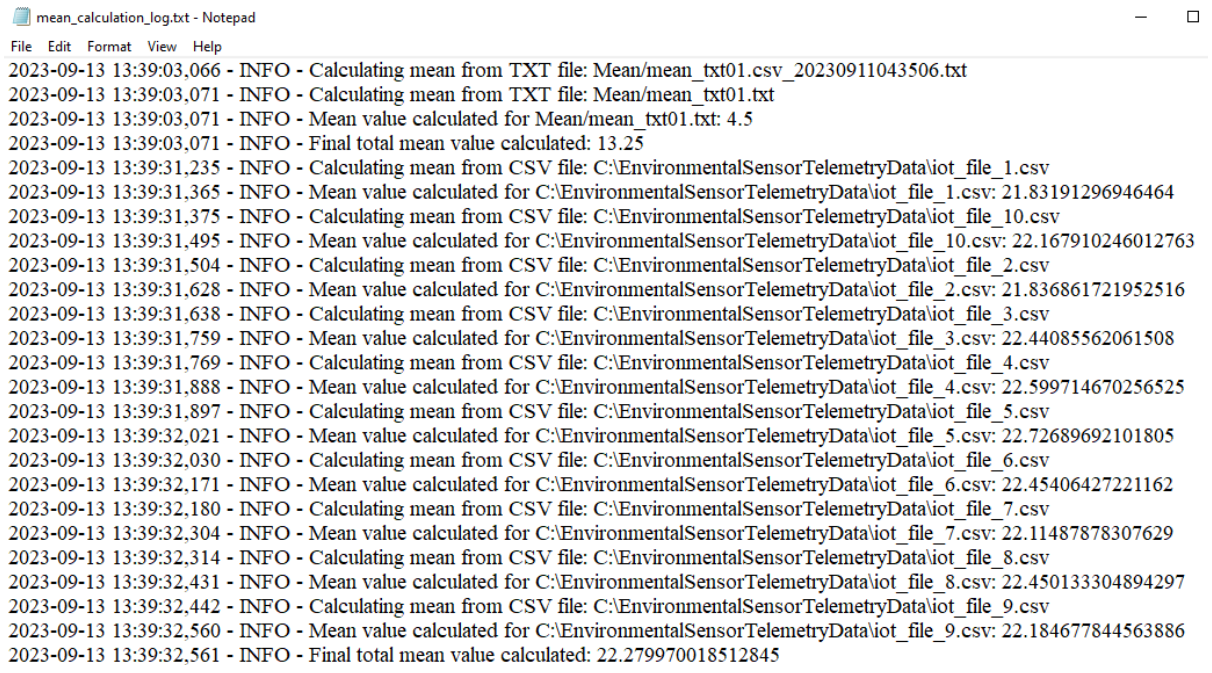
In summary, this code snippet demonstrates how to configure and utilize the FastTextProcessor library's calculateMean function to calculate mean values for specific sensor data while applying user-defined settings. It provides flexibility in handling sensor data and extracting meaningful insights from it.



```
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_1.csv: 21.83191296946464
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_10.csv: 22.167910246012763
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_2.csv: 21.836861721952516
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_3.csv: 22.44085562061508
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_4.csv: 22.599714670256525
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_5.csv: 22.72689692101805
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_6.csv: 22.45406427221162
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_7.csv: 22.11487878307629
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_8.csv: 22.450133304894297
Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_9.csv: 22.184677844563886
Final total mean value calculated: 22.279970018512845
Execution time for calculateMean: 1.337496042251587 seconds
```

Figure 5.3.1.4 Results of calculateMean function printed at console.

As depicted in Figure 5.3.1.4, the code calculates the mean value for each processed file, and these individual mean values are displayed in the console. Following this, the code calculates the final total mean value by summing up all the individual mean values and dividing this sum by the number of files processed. Remarkably, the entire calculation process is remarkably efficient, taking only 1.3375 seconds to complete.

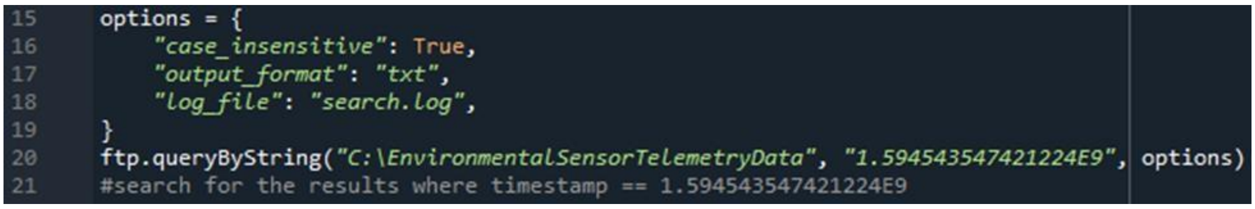


```
2023-09-13 13:39:03,066 - INFO - Calculating mean from TXT file: Mean/mean_txt01.csv_20230911043506.txt
2023-09-13 13:39:03,071 - INFO - Calculating mean from TXT file: Mean/mean_txt01.txt
2023-09-13 13:39:03,071 - INFO - Mean value calculated for Mean/mean_txt01.txt: 4.5
2023-09-13 13:39:03,071 - INFO - Final total mean value calculated: 13.25
2023-09-13 13:39:31,235 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_1.csv
2023-09-13 13:39:31,365 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_1.csv: 21.83191296946464
2023-09-13 13:39:31,375 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_10.csv
2023-09-13 13:39:31,495 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_10.csv: 22.167910246012763
2023-09-13 13:39:31,504 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_2.csv
2023-09-13 13:39:31,628 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_2.csv: 21.836861721952516
2023-09-13 13:39:31,638 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_3.csv
2023-09-13 13:39:31,759 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_3.csv: 22.44085562061508
2023-09-13 13:39:31,769 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_4.csv
2023-09-13 13:39:31,888 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_4.csv: 22.599714670256525
2023-09-13 13:39:31,897 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_5.csv
2023-09-13 13:39:32,021 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_5.csv: 22.72689692101805
2023-09-13 13:39:32,030 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_6.csv
2023-09-13 13:39:32,171 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_6.csv: 22.45406427221162
2023-09-13 13:39:32,180 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_7.csv
2023-09-13 13:39:32,304 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_7.csv: 22.11487878307629
2023-09-13 13:39:32,314 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_8.csv
2023-09-13 13:39:32,431 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_8.csv: 22.450133304894297
2023-09-13 13:39:32,442 - INFO - Calculating mean from CSV file: C:\EnvironmentalSensorTelemetryData\iot_file_9.csv
2023-09-13 13:39:32,560 - INFO - Mean value calculated for C:\EnvironmentalSensorTelemetryData\iot_file_9.csv: 22.184677844563886
2023-09-13 13:39:32,561 - INFO - Final total mean value calculated: 22.279970018512845
```

Figure 5.3.1.5 The "mean_calculation_log" records the output generated after running the provided code.

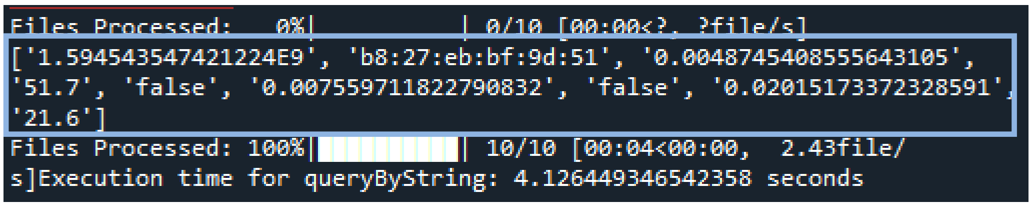
The log file serves as a convenient tool for users to review previous program output, and it meticulously records essential details such as date and time as illustrated in Figure 5.3.1.5. Additionally, the log file maintains a historical record of any errors that occurred during program execution. This record of errors can greatly assist engineers in identifying issues and streamlining the problem-solving process, ultimately making troubleshooting more efficient and effective.

2. Query for the results where timestamp equal to 1.594543547421224E9



```
options = {
    "case_insensitive": True,
    "output_format": "txt",
    "log_file": "search.log",
}
ftp.queryByString("C:\EnvironmentalSensorTelemetryData", "1.594543547421224E9", options)
#search for the results where timestamp == 1.594543547421224E9
```

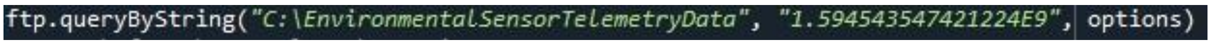
Figure 5.3.1.6: Only 5 lines of code are needed to execute a string query using FastTextProcessor.



```
Files Processed:   0%|          | 0/10 [00:00<?, ?file/s]
['1.594543547421224E9', 'b8:27:eb:bf:9d:51', '0.0048745408555643105',
'51.7', 'false', '0.007559711822790832', 'false', '0.02015173372328591',
'21.6']
Files Processed: 100%|██████████| 10/10 [00:04<00:00,  2.43file/
s]Execution time for queryByString: 4.126449346542358 seconds
```

Figure 5.3.1.7: The results generated by the queryByString function are displayed, accompanied by a progress bar to visualize the processing progress.

In today's data-driven world, the ability to efficiently extract valuable insights from vast datasets is paramount. FastTextProcessor, a versatile Python library, emerges as a powerful tool for simplifying data queries. Figure 5.3.1.6 reveals that with only 5 lines of code can enable users to extract valuable information from the environmental sensor telemetry data via FastTextProcessor. The dictionary "options" is same as previous function which allowing users to tailor their query precisely to their needs. The code snippet proceeds to execute a query using the queryByString function:



```
ftp.queryByString("C:\EnvironmentalSensorTelemetryData", "1.594543547421224E9", options)
```

Figure 5.3.1.8 Code snippet of queryByString

Let's break down the query:

i. **Query Location** (C:\EnvironmentalSensorTelemetryData): The function is directed to search for the specified query string within a designated directory. In this case, it's "C:\EnvironmentalSensorTelemetryData," where the environmental sensor telemetry data is stored. Users have the flexibility to replace this directory path with their desired file directory. FastTextProcessor will then filter through all the files in the provided directory, performing the query operation. (Note: The query operation is applicable to JSON, CSV, and text files within the directory.)

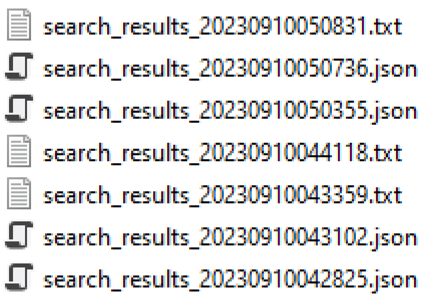
ii. **Query String** ("1.594543547421224E9"): The query string is the specific pattern or value that users are searching for within the dataset. It could represent a timestamp, sensor reading, or any other data point of interest.

iii. **Options Dictionary**: The previously defined "options" dictionary is passed as an argument to the queryByString function, allowing users to customize the query behavior according to their preferences.

Additionally, Figure 5.3.1 shows that users are presented with a progress bar that provides valuable insights into the search process upon invoking the queryByString function. This progress bar not only indicates the percentage of files processed but also offers a visual representation of the search's progress. In the output snippet provided, user can observe the progress bar's updates as it iterates through a total of ten files within the designated directory. The list of data elements enclosed within square brackets is the result of the query operation. Each element in the list represents a data point that matches the search criteria defined by the user. In this particular output, it displayed a series of values, which likely pertain to sensor telemetry data. These values include unique identifiers, boolean indicators, and numerical readings.

Next, one key observation is that the queryByString function doesn't merely identify matching files but also extracts relevant data points from those files. This means that users receive a concise and informative summary of the data that meets their query criteria. As the progress bar reaches completion, essential information will be provided regarding the execution time of the queryByString function. In the given example, the query operation took approximately 4.26 seconds to complete. This insight into execution time is valuable for assessing the efficiency of the search, particularly when dealing with larger datasets.

As demonstrated in Figure 5.3.1.9, the search results will be conveniently saved in both JSON and text file formats, as specified by the user-defined options dictionary. This feature is designed for user convenience, allowing easy reference to the results whenever needed. To ensure uniqueness, the saved files are named with a timestamp. In summary, the output of the queryByString function in FastTextProcessor offers a clear and concise summary of the query results. It provides users with the ability to explore and extract specific data points of interest within a directory of files efficiently.



search_results_20230910050831.txt
search_results_20230910050736.json
search_results_20230910050355.json
search_results_20230910044118.txt
search_results_20230910043359.txt
search_results_20230910043102.json
search_results_20230910042825.json

Figure 5.3.1.9 The search_results_timestamp file, can be saved in both JSON and text file formats based on the options dictionary defined.

The provided code snippet exemplifies the simplicity and power of FastTextProcessor for data querying and mean value calculation tasks, which achieved the objectives of this project. By configuring options, specifying a query location, and defining the query string, users can effortlessly sift through extensive datasets to extract pertinent information. Moreover, the generated log file enhances the query process by offering valuable insights into its execution. FastTextProcessor's user-friendly approach to data querying, along with its ability to calculate mean values, streamlines the process and empowers users to harness the full potential of their data.

5.4 Implementation Issues and Project Challenges

Several challenges arose during implementation, including difficulties in calculating mean values from various file types within the directory. Some file extensions posed issues, requiring the function to adapt and accurately process different file formats. Additionally, issues with result files were identified, as they initially contained only the last row of results due to overwriting. These challenges were resolved with guidance from the supervisor.

Another primary challenge was handling diverse data formats, such as CSV, JSON, or plain text, each with unique parsing requirements. Ensuring seamless processing while maintaining data integrity was critical. The project should be able to handle more data formats like parquet.

Next, performance was a key challenge, particularly when processing extensive and complex datasets. Efficient algorithms and data structures were essential to prevent slowdown with large data volumes. Besides, implementing parallel processing might increase the performance.

Data quality and consistency also posed challenges, as real-world data often contains missing or erroneous values and lacks standardized structures. Addressing these issues was crucial for meaningful analysis.

Chapter 6
System Evaluation and Discussion

6.1 System Testing and Performance Metrics

In Chapter 6, system testing of the data query tool, "FastTextProcessor" will be conducted. This tool provides essential functionalities for searching and processing data from various file formats, including JSON, CSV, and text files. In this section, the primary functionalities and performance of FastTextProcessor will be explored and evaluated. FastTextProcessor offers the following main functionalities:

1. **Search and Match**: The tool allows users to search for specific patterns or keywords within JSON, CSV, and text files, providing matching results.
2. **Mean Calculation**: It can calculate mean values from CSV and JSON data based on user-specified conditions, such as specific columns and device criteria.

To assess the system's performance using diverse test cases and performance metrics, several key aspects including query execution time, and accuracy of results are focused on:

1. **Query Execution Time**: The amount of time it takes for the program to process a query or calculation and return results. This metric is crucial for assessing the toolset's conveniency. Shorter execution times are generally desirable as they indicate faster query processing.
2. **Accuracy of Results:** It measures the correctness and precision of the information provided by the toolset. It assesses whether the toolset produces the expected and accurate results. High accuracy is paramount, especially in data analysis and query tools. Accurate results ensure that decisions and insights drawn from the data are reliable and trustworthy.
3. **Coding Lines:** It measures the toolset's simplicity and conciseness in comparison to other databases or query tools. A lower number of coding lines can indicate a more straightforward and efficient implementation, making it easier for developers to work with and maintain the codebase. It also reduces the likelihood of errors and bugs, contributing to overall system reliability and ease of maintenance.

In addition to evaluating FastTextProcessor's efficiency in terms of data querying and mean value calculations, it's essential to assess its performance comprehensively from various angles. The results are summarized in Table 6.2.2.1 below, shedding light on its overall performance.

Function	**Execution Time**	**Lines of Code**	**Accuracy**	**User-Friendliness**
queryByString	4.1264 seconds	5 lines	Accurate	High
calculateMean	1.3375 seconds	7 lines	Accurate	High
Overall Efficiency	High	High	High	High

Table 6.1.1 FastTextProcessor System Testing and Performance Metrics

The data accuracy is verified by the outputs of SQLite and Pandas where the outputs for same query is the same. For example, all the blue boxes on the print screens indicates the same results for queryByString, while red colour boxes given same output which is 22.28 seconds for calculateMean function.

6.2 Testing Setup and Result

Testing Setup

In this section, the testing results and comparison between FastTextProcessor, SQLite, and Pandas will be conducted. The constant testing setup is used while measuring the time taken for calculationMean and queryByString functions on different platforms. This FYP would like to acknowledge and give credit to the owners of the dataset used. The dataset, "Environmental Sensor Telemetry Data", was obtained from Kaggle Data Repository (https://www.kaggle.com/datasets/garystafford/environmental-sensor-data-132k), and this project extend the gratitude to the data contributors for making it publicly available for education purposes. The dataset is divided equally into 10 csv files which each are around 6,000 kB. Additionally, all test cases were running on the Spyder version 5.1.5 and using Python 3.9.

Testing results

To ensure the robustness and reliability of FastTextProcessor, a comprehensive system testing phase was conducted. This phase encompassed various aspects of the tool's functionality, including data querying and mean value calculations. The performance metrics and results presented in the table above provide a valuable glimpse into FastTextProcessor's efficiency and user-friendliness. However, it's equally important to delve into specific test cases to understand how FastTextProcessor handles real-world scenarios. Thus, Table 6.1.2 below will show the specified test cases and the testing details of FastTextProcessor.

Input	Expected Output	Results
calculateMean("Mean/", "sensor03", "temp", options)	Calculated mean without condition and printed the output at console. Saved the printed results in separate file in user specified file type.	PASS
	Complete mean results for each files and the total mean value are recorded in mean_calculation_log file	PASS

calculateMean("Mean/", "sensor03", "temp", options, "<25")	Calculated mean with condition. Saved the printed results in separate file in user specified file type.	PASS
	Complete mean results for each files and the total mean value are recorded in mean_calculation_log file	PASS
calculateMean("Mean/", "sensor03", "temp", options, "=25", "20230911083000")	Calculated mean with condition and timestamp. Saved the printed results in separate file in user specified file type.	PASS
	Complete mean results for each files and the total mean value are recorded in mean_calculation_log file	PASS
queryByString("Mean/", "sensor03", options)	Printed the query results accurately	PASS
	Complete results are saved in the search_result_timestamp file.	PASS
	Log file is updated, and the details are recorded.	PASS
Place different file types in file directory and execute functions	Results should be include all files extensions like CSV, JSON, and text file.	PASS
Set case-insensitive equal to true	Displayed results with case-insensitive	PASS
Set case-insensitive equal to false	Displayed results without case-insensitive	PASS
Set output format equal to “json”	Saved the output results in JSON file	PASS
Set output format equal to “txt”	Saved the output results in text file	PASS
Set the “log_file” to search.log	Saved log_file in search log	PASS

Table 6.1.2 Verification Test for FastTextProcessor.

6.3 Comparison results between FastTextProcessor, SQLite, and Pandas

Now, the performance between different methods ill be compared from the aspects of time taken to perform tasks and lines of codes required to complete each functions. The tools like SQLite and Pandas will be used as they are common tools that will be used by engineers when it comes to query large datasets.

i. Comparison of Lines of Code by FastTextProcessor, SQLite, and Pandas

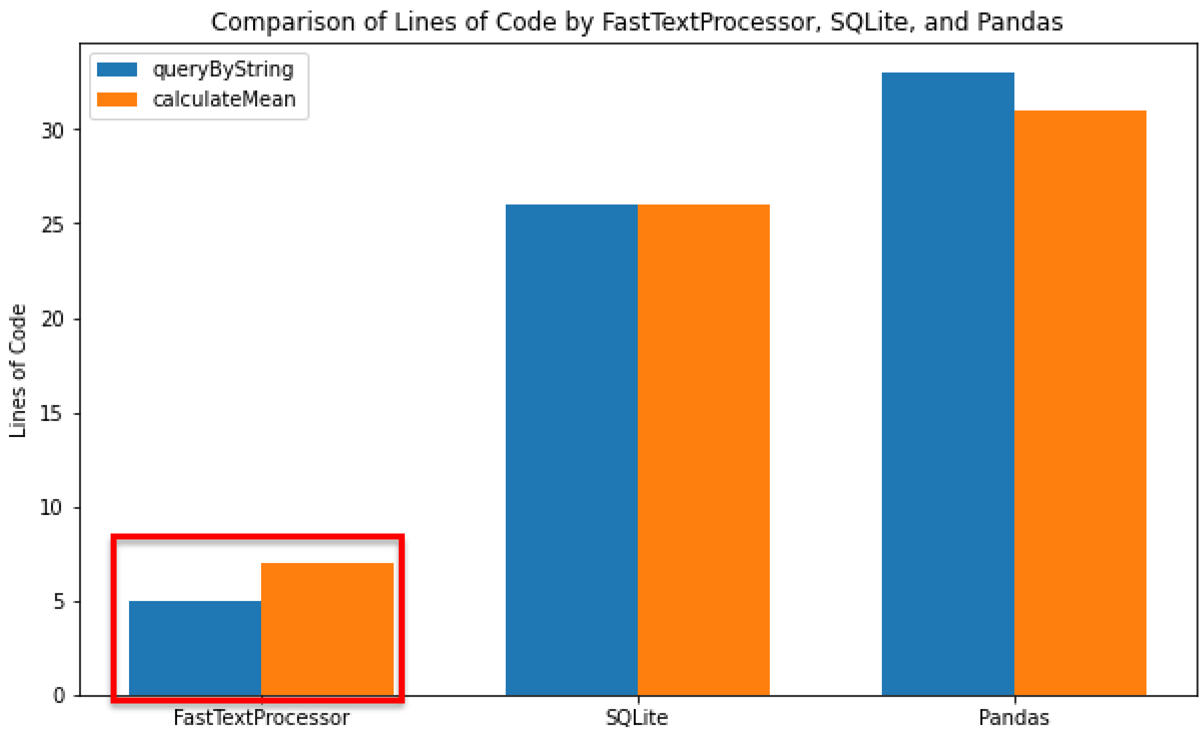


Figure 6.2.2.1 Graph of Comparison of Lines of Code by FastTextProcessor, SQLite, and Pandas

In Figure 6.2.2.1, the graph visually illustrates that FastTextProcessor excels in providing a streamlined and efficient approach to text querying and mean value calculations. It achieves these tasks with a minimal number of lines of code, highlighting its simplicity and user-friendliness. In contrast, SQLite and Pandas, renowned for their robust data processing capabilities, necessitate a considerably larger codebase to accomplish similar operations. While SQLite and Pandas are versatile choices for complex data tasks, FastTextProcessor emerges as the preferred option for users seeking a straightforward and efficient solution for text-based queries and mean value calculations. Further details regarding lines of code for each tool and comprehensive testing results are presented in the following section.

ii. Comparison of execution time between FastTextProcessor, SQLite, and Pandas

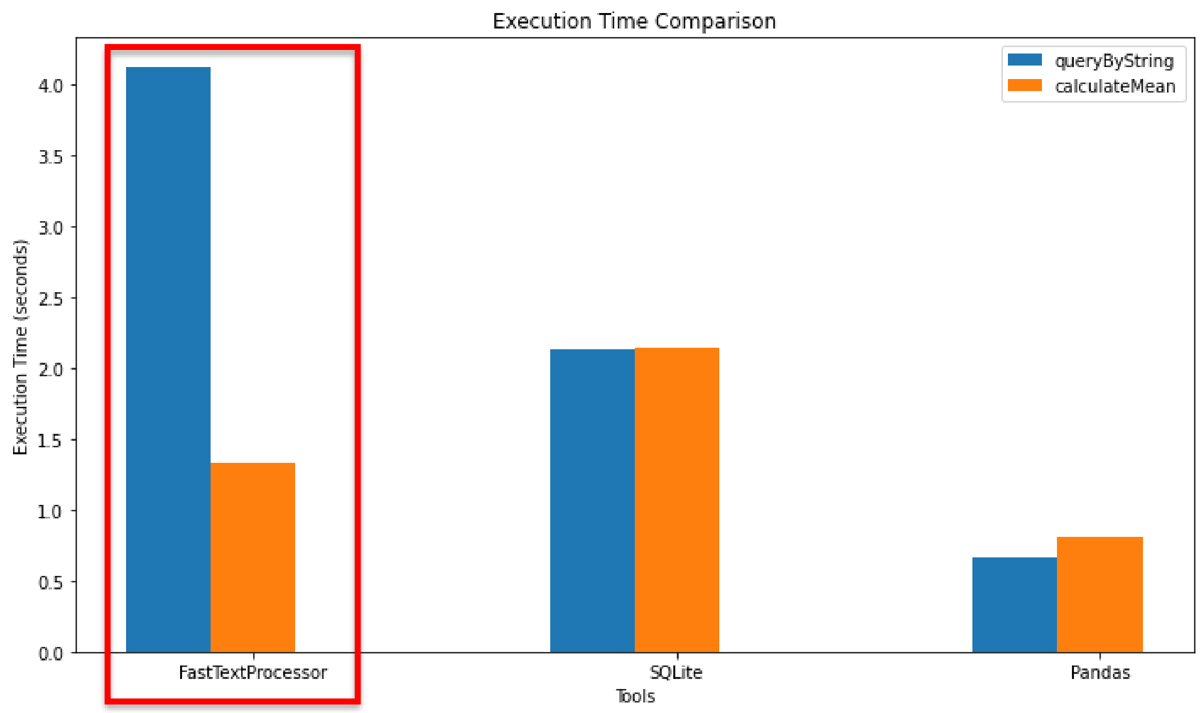


Figure 6.2.2.2 Graph of Comparison of Execution time comparison

As depicted in Figure 6.2.2.2, the execution times of key operations across different data processing tools: FastTextProcessor, SQLite, and Pandas. While FastTextProcessor exhibits slightly longer execution times compared to SQLite and Pandas for certain operations, it is essential to delve deeper into the context to truly appreciate its advantages. FastTextProcessor may have recorded the highest execution times for both the "queryByString" and "calculateMean" operations, but these times remain within the range of mere seconds— 4.1264 seconds and 1.3375 seconds, respectively. Importantly, these times are well below the 10 seconds, demonstrating that FastTextProcessor excels in processing speed, ensuring quick results even for moderately large datasets.

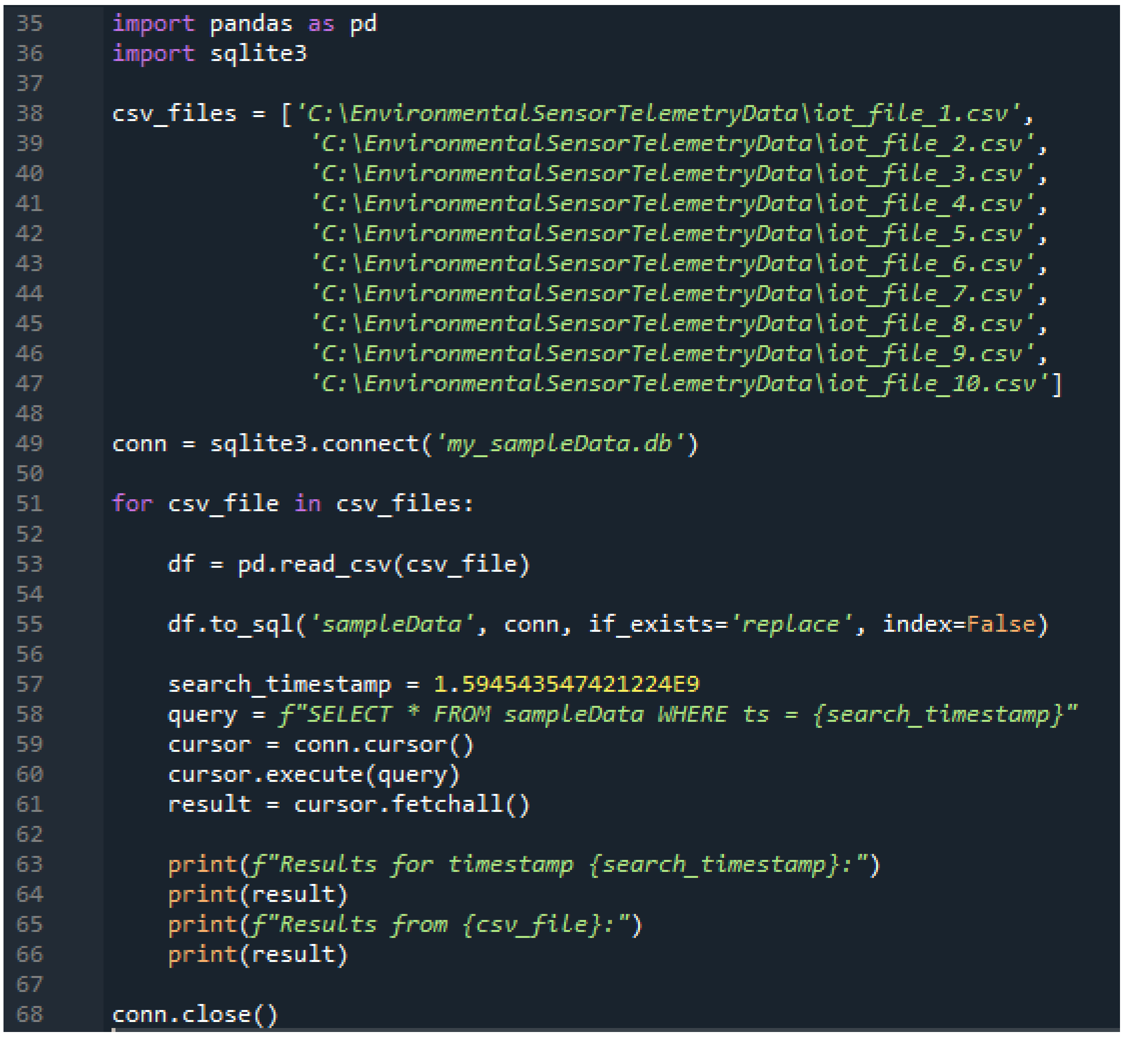
However, the true essence of FastTextProcessor's superiority becomes evident when considering the lines of code required for these operations. While FastTextProcessor accomplishes these tasks with a mere 5 lines of code for "queryByString" and 7 lines for "calculateMean," alternative solutions such as Python or SQLite demand substantially more code—33 lines for "queryByString" and 26 lines for "calculateMean" in SQLite's case. The significance of this discrepancy cannot be overstated. Writing and debugging extensive lines of code takes time and effort, which can often outweigh the benefits of slightly faster execution times. In a real-world scenario, where time is a precious commodity, the efficiency gained from

FastTextProcessor's simplicity translates into significant advantages. Besides, the minimalistic syntax of FastTextProcessor enables users to quickly grasp and implement their data processing tasks without the need for extensive training or specialized knowledge.

The testing results and process of SQLite and Pandas will be displayed in following section:

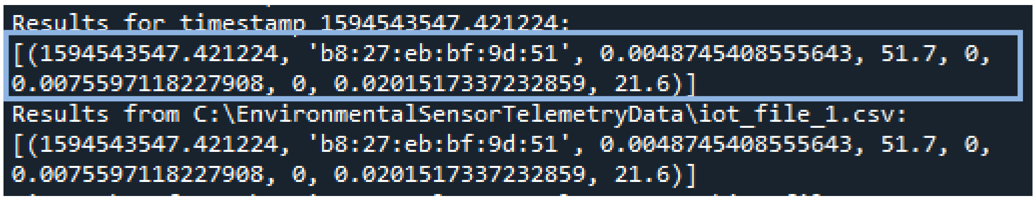
i. SQLite

The testing process begins with the evaluation of the query-by-string function, followed by the mean calculation testing. Thus, the testing results of query for the results where timestamp equal to 1.594543547421224E9 will be provided as follow:



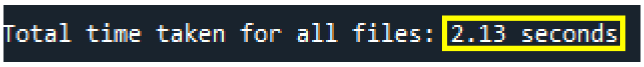
```python
import pandas as pd
import sqlite3

csv_files = ['C:\EnvironmentalSensorTelemetryData\iot_file_1.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_2.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_3.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_4.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_5.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_6.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_7.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_8.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_9.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_10.csv']

conn = sqlite3.connect('my_sampleData.db')

for csv_file in csv_files:

    df = pd.read_csv(csv_file)

    df.to_sql('sampleData', conn, if_exists='replace', index=False)

    search_timestamp = 1.594543547421224E9
    query = f"SELECT * FROM sampleData WHERE ts = {search_timestamp}"
    cursor = conn.cursor()
    cursor.execute(query)
    result = cursor.fetchall()

    print(f"Results for timestamp {search_timestamp}:")
    print(result)
    print(f"Results from {csv_file}:")
    print(result)

conn.close()
```

Figure 6.2.2.3 26 lines of code needed for SQLite to perform string query.



```
Results for timestamp 1594543547.421224:
[(1594543547.421224, 'b8:27:eb:bf:9d:51', 0.0048745408555643, 51.7, 0,
0.0075597118227908, 0, 0.0201517337232859, 21.6)]
Results from C:\EnvironmentalSensorTelemetryData\iot_file_1.csv:
[(1594543547.421224, 'b8:27:eb:bf:9d:51', 0.0048745408555643, 51.7, 0,
0.0075597118227908, 0, 0.0201517337232859, 21.6)]
```

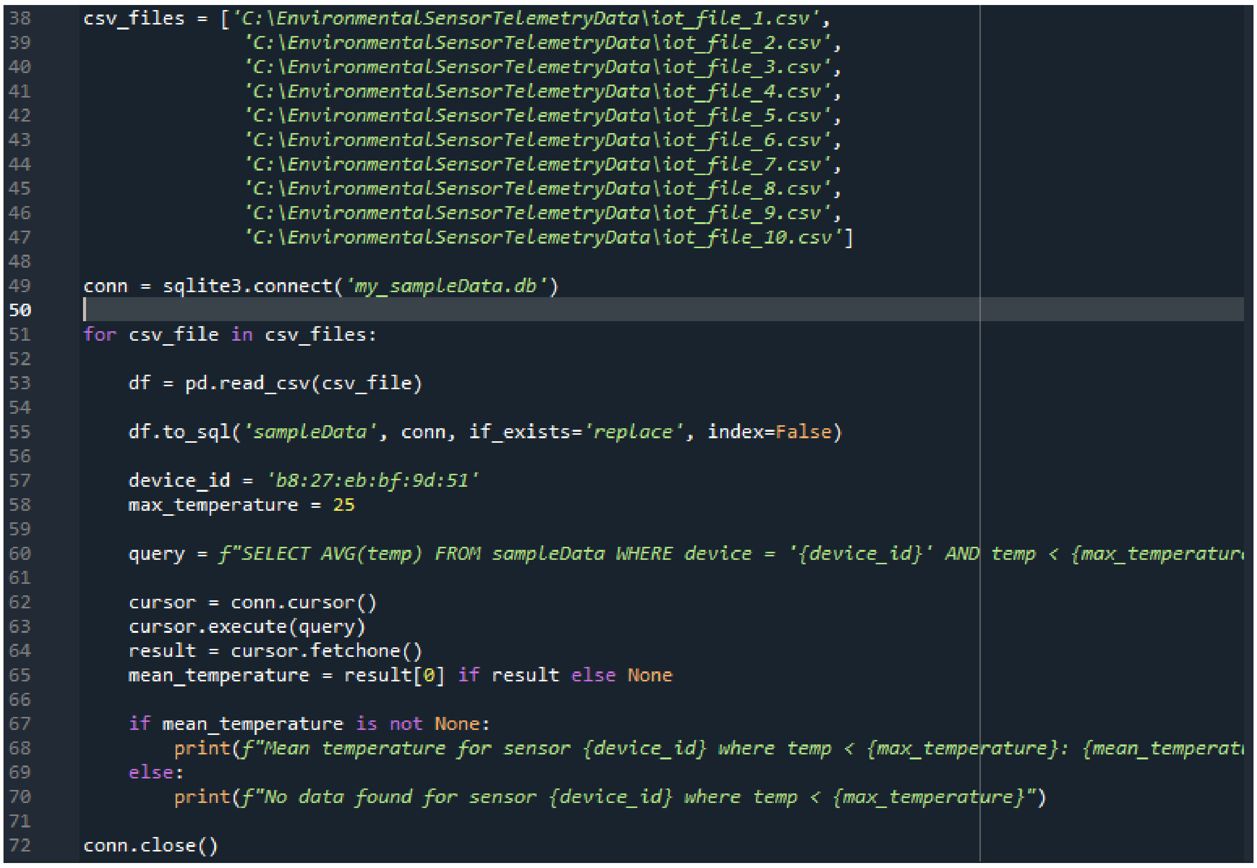
Figure 6.2.2.4 The printed query results through the use of SQLite



```
Total time taken for all files: 2.13 seconds
```

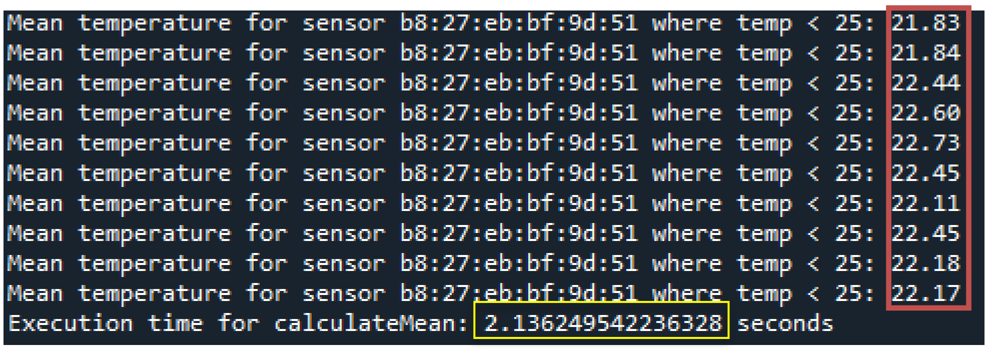
Figure 6.2.2.5 Execution queries with SQLite takes approximately 2.13 seconds

Calculate mean value for sensor “b8:27:eb:bf:9d:51” where its temperature is less than 25 on SQLite:



```python
csv_files = ['C:\EnvironmentalSensorTelemetryData\iot_file_1.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_2.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_3.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_4.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_5.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_6.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_7.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_8.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_9.csv',
             'C:\EnvironmentalSensorTelemetryData\iot_file_10.csv']

conn = sqlite3.connect('my_sampleData.db')

for csv_file in csv_files:

    df = pd.read_csv(csv_file)

    df.to_sql('sampleData', conn, if_exists='replace', index=False)

    device_id = 'b8:27:eb:bf:9d:51'
    max_temperature = 25

    query = f"SELECT AVG(temp) FROM sampleData WHERE device = '{device_id}' AND temp < {max_temperatur

    cursor = conn.cursor()
    cursor.execute(query)
    result = cursor.fetchone()
    mean_temperature = result[0] if result else None

    if mean_temperature is not None:
        print(f"Mean temperature for sensor {device_id} where temp < {max_temperature}: {mean_temperat
    else:
        print(f"No data found for sensor {device_id} where temp < {max_temperature}")

conn.close()
```

Figure 6.2.2.6 26 lines of code needed for SQLite to perform mean calculation.



```
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 21.83
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 21.84
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.44
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.60
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.73
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.45
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.11
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.45
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.18
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.17
Execution time for calculateMean: 2.136249542236328 seconds
```

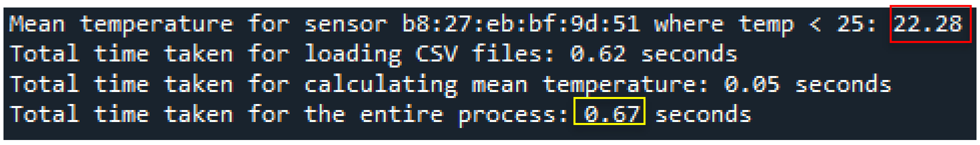
/10

Figure 6.2.2.7 Execution of mean calculation with SQLite takes approximately 2.14 seconds

The average value calculated by SQLite is 22.28 seconds, matching the results obtained with FastTextProcessor and Pandas, confirming the accuracy of the data processing.

ii. Python

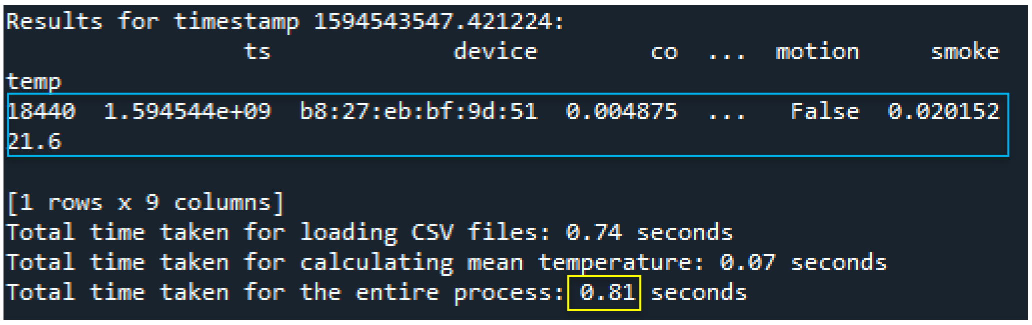
Calculate mean value for sensor “b8:27:eb:bf:9d:51” where its temperature is less than 25 using Python:



```
Mean temperature for sensor b8:27:eb:bf:9d:51 where temp < 25: 22.28
Total time taken for loading CSV files: 0.62 seconds
Total time taken for calculating mean temperature: 0.05 seconds
Total time taken for the entire process: 0.67 seconds
```

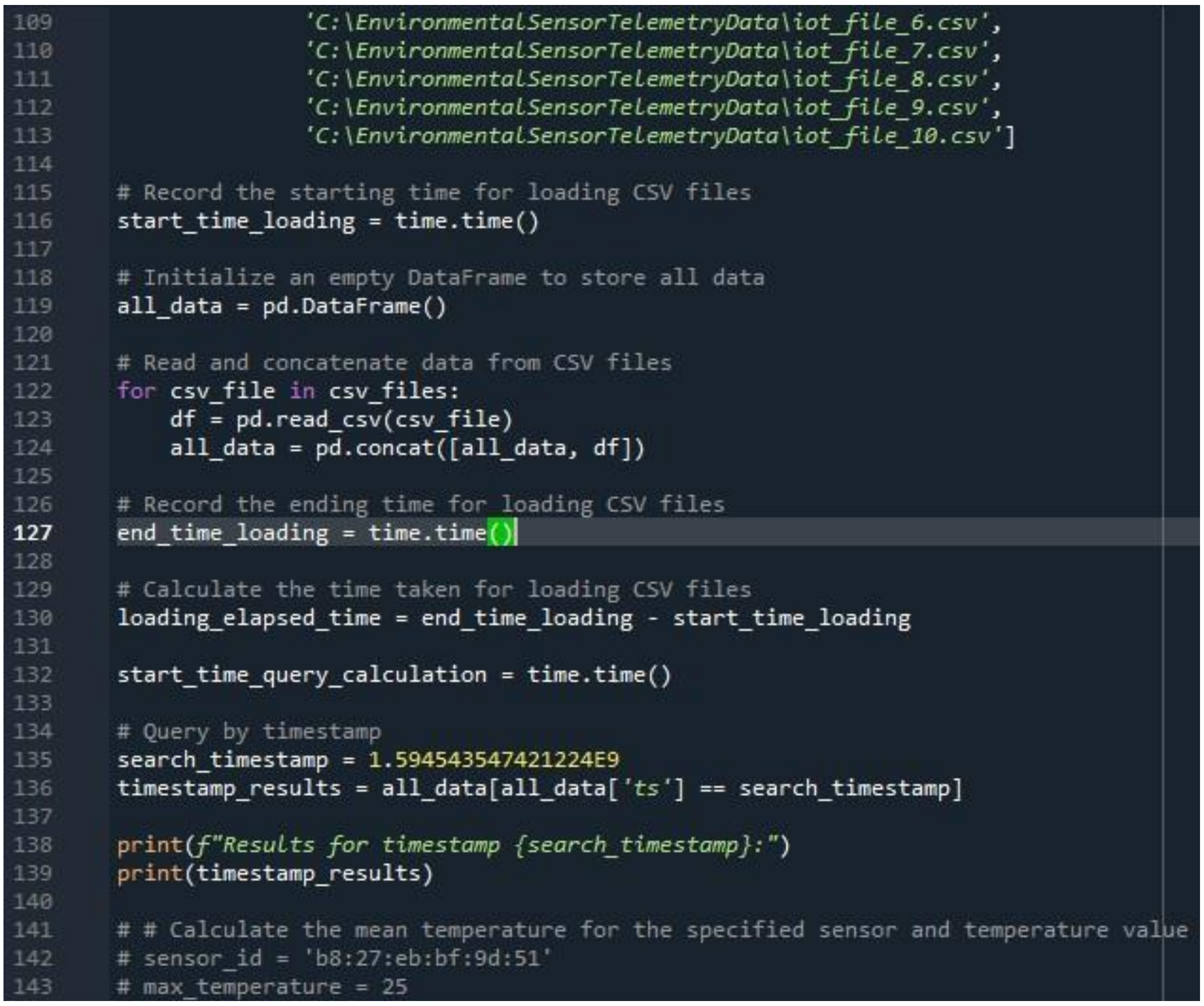
Figure 6.2.2.8 Time taken to perform mean calculation using Pandas is 0.67 seconds

Query for the results where timestamp equal to 1.594543547421224E9 using Python:



```
Results for timestamp 1594543547.421224:
                 ts             device        co  ...  motion     smoke
temp
18440  1.594544e+09  b8:27:eb:bf:9d:51  0.004875  ...   False  0.020152
21.6

[1 rows x 9 columns]
Total time taken for loading CSV files: 0.74 seconds
Total time taken for calculating mean temperature: 0.07 seconds
Total time taken for the entire process: 0.81 seconds
```

Figure 6.2.2.9 Time taken to perform query using Pandas is 0.81 seconds



```python
                'C:\EnvironmentalSensorTelemetryData\iot_file_6.csv',
                'C:\EnvironmentalSensorTelemetryData\iot_file_7.csv',
                'C:\EnvironmentalSensorTelemetryData\iot_file_8.csv',
                'C:\EnvironmentalSensorTelemetryData\iot_file_9.csv',
                'C:\EnvironmentalSensorTelemetryData\iot_file_10.csv']

# Record the starting time for loading CSV files
start_time_loading = time.time()

# Initialize an empty DataFrame to store all data
all_data = pd.DataFrame()

# Read and concatenate data from CSV files
for csv_file in csv_files:
    df = pd.read_csv(csv_file)
    all_data = pd.concat([all_data, df])

# Record the ending time for loading CSV files
end_time_loading = time.time()

# Calculate the time taken for loading CSV files
loading_elapsed_time = end_time_loading - start_time_loading

start_time_query_calculation = time.time()

# Query by timestamp
search_timestamp = 1.594543547421224E9
timestamp_results = all_data[all_data['ts'] == search_timestamp]

print(f"Results for timestamp {search_timestamp}:")
print(timestamp_results)

# # Calculate the mean temperature for the specified sensor and temperature value
# sensor_id = 'b8:27:eb:bf:9d:51'
# max_temperature = 25
```

Figure 6.2.2.10 Code snippet of Python on performing tasks of mean calculation(31 lines) and string query (33 lines).

6.4 Objectives Evaluation

The development of Fast Text Processor has achieved the 2 main objectives:

1. Successfully implemented a preprocessing program for data analytics so that data does not need to be populated before it can be queried like existing NoSQL database.
2. Successfully developed a program that allows data to be run directly by data analytics software without formatting the data.

Moreover, the additional functions like logging and reporting are also built in the program to handle errors and for convenience. The code needs to be capable of gracefully handling unexpected errors, such as missing files or invalid data, and provide informative logging to aid in debugging and monitoring.

6.5 Concluding Remark

The performance of Fast Text Processor is the best among other data tools such as SQLite and Pandas. It took lowest number of lines for both queryByString and calculateMean functions compared to other technologies. Besides, Fast Text Processor has passed through all the verification cases.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

This FYP has successfully addressed the initial problem statements and fulfilled its objectives. The development of the Fast Text Processor in Python, using the Spyder IDE, simplifies data retrieval from diverse datasets, reducing time and effort compared to conventional database methods. The program's adaptability to various data formats enhances its versatility. Furthermore, the program exhibits high data accuracy while requiring the lowest number of lines of code and the shortest possible coding time compared to other tools. Hence, it represents an innovative and successful program.

7.2 Recommendation

Fast Text Processor serves as a temporary tool designed to cater to users' specific needs, particularly those who require immediate data extraction before forwarding it for further processing. This tool excels when dealing with datasets that aren't frequently accessed or manipulated, making it an efficient choice for scenarios where a full-fledged database setup would be excessive.

Imagine a situation where a person needs to investigate potential server breaches. In such a scenario, the individual can swiftly download the server's log files, and rather than going through the time-consuming and complex process of setting up a traditional database, they can leverage Fast Text Processor. This tool simplifies the task of querying and analyzing the log data, enabling them to quickly identify and address any unusual access patterns or potential security threats. Fast Text Processor's flexibility and ease of use streamline the entire data extraction and analysis process, ultimately saving valuable time and effort for the user.

REFERENCES

[1]C. Taylor, "Structured vs Unstructured Data 101: Top Guide | Datamation", Datamation, 2021. [Online]. Available: https://www.datamation.com/big-data/structured-vs-unstructured-data/#:~:text=Unstructured%20data%20makes%20up%2080,on%20the%20business%20intelligence%20table. [Accessed: 19- Aug- 2022].

[2]D. Reinsel, J. Gantz and J. Rydning, "The Digitization of the World From Edge to Core", Seagate.com, 2018. [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. [Accessed: 19- Aug- 2022].

[3]S. García, S. Ramírez-Gallego, J. Luengo, J. Benítez and F. Herrera, "Big data preprocessing: methods and prospects", Springer, 2016. [Online]. Available: https://doi.org/10.1186/s41044-016-0014-0. [Accessed: 19- Aug- 2022].

[4] C. Fan, M. Chen, X. Wang, J. Wang, and B. Huang, "A review on data preprocessing techniques toward efficient and reliable knowledge discovery from building operational data," Frontiers, 01-Jan-1AD. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fenrg.2021.652801/full#B66. [Accessed: 20-Aug-2022].

[5] S. Kanoje, V. Powar and D. Mukhopadhyay, "Using MongoDB for social networking website deciphering the pros and cons," 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), 2015, pp. 1-3, doi: 10.1109/ICIIECS.2015.7192924.

[6]I. Luković, S. Ristić and P. Mogin, "A Methodology of A Database Schema Design Using The Subschemas", 2003. [Online]. Available: https://www.researchgate.net/profile/Ivan-Lukovic/publication/256661455_A_Methodology_of_a_Database_Schema_Design_Using_The_Subschemas/links/0912f50615408c43e8000000/A-Methodology-of-a-Database-Schema-Design-Using-The-Subschemas.pdf. [Accessed: 25- Aug- 2022].

[7]A. Nayak, A. Poriya and D. Poojary, "Type of NOSQL Databases and its Comparison with Relational Databases", Citeseerx.ist.psu.edu, 2013. [Online]. Available:

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.3372&rep=rep1&type=pdf. [Accessed: 25- Aug- 2022].

[8]"What is Apache Spark - Azure HDInsight", Docs.microsoft.com, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/hdinsight/spark/apache-spark-overview. [Accessed: 29- Aug- 2022]

[9]S. Salloum, R. Dautov, X. Chen, P. Peng and J. Huang, "Big data analytics on Apache Spark", SpringerLink, 2016. [Online]. Available: https://link.springer.com/article/10.1007/s41060-016-0027-9#citeas. [Accessed: 29- Aug- 2022]

[10]M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker and I. Stoica, "Spark: cluster computing with working sets", Guide Proceedings, 2010. [Online]. Available: https://dl.acm.org/doi/10.5555/1863103.1863113. [Accessed: 29- Aug- 2022]

[11]S. Dahiya, "Advantages of PySpark", Programsbuzz.com, 2021. [Online]. Available: https://www.programsbuzz.com/article/advantages-pyspark. [Accessed: 29- Aug- 2022]

[12] "Four Important Advantages of Apache Spark", Linkedin.com, 2021. [Online]. Available: https://www.linkedin.com/pulse/four-important-advantages-apache-spark-elegant-microweb/. [Accessed: 29- Aug- 2022]

[13] "Limitations of Apache Spark-Ways To Overcome Spark Limitations", TechVidvan. [Online]. Available: https://techvidvan.com/tutorials/limitations-of-apache-spark/. [Accessed: 30- Aug- 2022]

[14]S. Landset, T. Khoshgoftaar, A. Richter and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem", 2015. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-015-0032-1#citeas. [Accessed: 30- Aug- 2022]

[15]H. Chahal and P. Gulia, "Comprehensive Study of Open-Source Big Data Mining Tools", 2016. [Online]. Available: https://www.researchgate.net/publication/322628934_Comprehensive_Study_of_Open-Source_Big_Data_Mining_Tools. [Accessed: 30- Aug- 2022]

[16]V. Kalra and R. Aggarwal, "Importance of Text Data Preprocessing & Implementation in RapidMiner", 2018. [Online]. Available: https://www.researchgate.net/profile/Rashmi-Agrawal-3/publication/322782057_Importance_of_Text_Data_Preprocessing_Implementation_in_RapidMiner/links/5b59a5420f7e9bc79a65eb85/Importance-of-Text-Data-Preprocessing-Implementation-in-RapidMiner.pdf. [Accessed: 30- Aug- 2022]

[17]E. G. Kulkarni and R. B. Kulkarn, WEKA Powerful Tool in Data Mining , 2016. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/52803329/Paper1-with-cover-page-v2.pdf?Expires=1662005152&amp;Signature=c52TQjjVhOdmDNqde7KUKHfihhaRghm7afKvfBtIUuDqAd5IQmlpFI1s4HApso~6MAs8Dr~4kj7e7l2fo4ibaG1Ndz87AX9kgblPmuHhn7i8ooK2KvQzyh2GUdTVsjmnH0TgMlc2f3QRQTotIyO7GLqyR31n4NVqcVE9MSueSQCKwC8izdrkw3-0iByZ80QGDC~3Lf8OlylNPixVhtwoyVO8CCJ6e5o789hDHH2ij2CXcIFu~pH8VLYvKGGhUe8o7Zere9FWUYcbZHuyG8X1guU--CaOpz-e726J1er2ieDt8CCsfig8XTJB1wHZebhXi2VZKQyidLd0jS9rdg~7hA_&amp;Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. [Accessed: 01-Sep-2022].

[18]"Weka 3 - Data Mining with Open Source Machine Learning Software in Java", Cs.waikato.ac.nz, 2022. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/. [Accessed: 01- Sep- 2022]

[19]S. Singhal and M. Jena, "A Study on WEKA Tool for Data Preprocessing, Classification and Clustering", International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 2, no. 6, pp. 1-4, 2013.

[20]K. Brush and E. Burns, “What is data visualization and why is it important?,” SearchBusinessAnalytics, 20-Feb-2020. [Online]. Available:

https://www.techtarget.com/searchbusinessanalytics/definition/data-visualization. [Accessed: 01-Sep-2022].

[21] “ETL process overview - ETL database,” Stitch. [Online]. Available: https://www.stitchdata.com/etldatabase/etl-process/. [Accessed: 02-Sep-2022].

[22] "What are database schemas? 5 minute guide with examples", Educative: Interactive Courses for Software Developers, 2022. [Online]. Available: https://www.educative.io/blog/what-are-database-schemas-examples. [Accessed: 02- Sep-2022]

[23] A. Dennis, B. H. Wixom, and D. Tegarden, Systems Analysis &amp; Design : An Object-Oriented Approach With UML ., 6th ed. United States of America: John Wiley &amp; Sons, Inc, 2021.

[24] C. Chien, “What is rapid application development (RAD)?,” What is Rapid Application Development (RAD)?, 04-Feb-2020. [Online]. Available: https://codebots.com/app-development/what-is-rapid-application-development-rad. [Accessed: 03-Sep-2022].

[25] M. Makai. Why Use Python? -Full Stack Python, 2017. URL:https://www.fullstackpython.com/why-use-python.html [Accessed: 03- Sep- 2022]

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 2**
Student Name & ID: Loh Yi Ling (19ACB06235)	
Supervisor: Dr Ts Dr Ooi Boon Yaik	
Project Title: loT Database for non structured data type	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

During this period, I focused on reviewing and studying the progress made in my FYP 1. By doing so, I was able to identify the areas that required further development in FYP 2. The following tasks have been accomplished:

FYP 1 Evaluation:
I thoroughly assessed the outcomes and achievements of my FYP 1. This evaluation allowed me to recognize the strengths and weaknesses of the previous phase, providing valuable insights for improvements in FYP 2.

Identifying FYP 2 Goals:
Based on the evaluation of FYP 1, I successfully determined the objectives and specific tasks to be addressed in FYP 2. These goals are aimed at building upon the foundation laid in the previous phase and ensuring overall project progression.

Addressing Weaknesses:
I dedicated effort to improving the weaknesses identified in FYP 1. By devising appropriate strategies and solutions, I aim to rectify the areas that did not meet expectations in the previous stage, ensuring a more robust and successful FYP 2.

Enhancing Strengths:
Additionally, I worked towards enhancing the strengths exhibited in FYP 1. By capitalizing on the successful aspects and refining them further, I aim to maximize the impact of FYP 2 and deliver an outstanding final project.

2. WORK TO BE DONE

In the upcoming period, the following tasks will be undertaken for my Final Year Project:

Learn Spyder:
I will dedicate time to learn and familiarize myself with the Spyder integrated development environment (IDE) to effectively use it for my project.

Python Library Implementation:

I will start working on the implementation of the Python library, which will involve developing functions for specific calculations and generating valuable outputs.

3. PROBLEMS ENCOUNTERED

While progressing through the tasks mentioned above, I anticipate encountering some challenges. Two potential problems that may arise are as follows:

Integration with Spyder:
Learning a new development environment like Spyder might present challenges in integrating my existing code or adapting it to this new IDE. I may need to resolve compatibility issues or update certain code snippets to make them work seamlessly within Spyder.

Function Accuracy and Edge Cases:
During the implementation of the Python library, I may face difficulties in ensuring the accuracy of the functions. So, robust testing and debugging will be essential to identify and address any issues that arise.

As I encounter these challenges, I will diligently seek solutions through research, consultation with my supervisor, and experimentation to ensure the successful completion of my Final Year Project.

4. SELF EVALUATION OF THE PROGRESS

Task Breakdown:
I will create a detailed breakdown of the tasks to be undertaken in FYP 2, ensuring a structured and organized approach to project execution.

Timeline Creation:
I will establish a comprehensive timeline for FYP 2, outlining milestones and deadlines to maintain steady progress throughout the project.

Alex

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 4**
Student Name & ID: Loh Yi Ling (19ACB06235)	
Supervisor: Dr Ts Dr Ooi Boon Yaik	
Project Title: loT Database for non structured data type	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Tool Change:
Based on valuable guidance from my supervisor, I decided to transition from using Jupyter Notebook to Spyder as my primary development environment for the project.

Learning Spyder:
I dedicated time to learn how to use Spyder effectively by utilizing various online resources, including YouTube tutorials. This enabled me to familiarize myself with Spyder's features and functionalities.

Python Library Development:
I commenced the development of the Python library, a significant component of my project. The purpose of this library is to perform specific calculations on the raw data and provide valuable outputs to the engineer before sending the data to the cloud.

Implementation of Functions:
Within the Python library, I successfully implemented the following functions:

Minimum Value Calculation: Developed a function that calculates the minimum value within selected files.
Maximum Value Calculation: Implemented a function to calculate the maximum value within selected files.
Mean Value Calculation: Created a function to calculate the mean value within the files.

2. WORK TO BE DONE

Integration Testing:
In this phase, I will verify the accuracy of the Python library's calculations through rigorous integration testing with different datasets. Additionally, I plan to implement more valuable functions to enhance the capabilities of the library.

Meeting with Supervisor:
Schedule a meeting with my supervisor to present the progress made and seek feedback on the implementation so far.

3. PROBLEMS ENCOUNTERED

During the development of the coding for my Final Year Project, I encountered a significant challenge related to user input of file addresses. As I was implementing the functions in the Python library, I realized that allowing users to input individual file paths could be cumbersome and less user-friendly, especially when dealing with multiple files. Instead, it would be more convenient if users could directly input the folder path containing the relevant files.

Proposed Solution:

To address this issue, I plan to discuss this matter with my supervisor during our meeting next week. I will propose a solution where users can input the folder path containing the necessary files, and the Python library will automatically process all the files within that folder. This approach will simplify the user experience, making it easier and more efficient for users to analyze multiple files at once.

4. SELF EVALUATION OF THE PROGRESS

Areas for Improvement:

Task Breakdown and Time Allocation:
While I accomplished significant milestones, I acknowledge the need for a more meticulous breakdown of tasks to ensure efficient time allocation. By breaking down tasks into logical and evenly distributed pieces, I can better manage my time and prevent potential time constraints.

Reinforcing Time Management:
Improving time management is crucial to ensure that I have ample time for my FYP. I need to allocate sufficient time for research, development, testing, and documentation, ensuring a well-rounded and comprehensive project.

Prioritization of FYP Activities:
I should prioritize FYP activities based on their significance and deadlines. By identifying critical tasks and allocating more time to complex or time-consuming aspects, I can optimize the project's progress.

Plan for Improvement:

To address these areas for improvement, I have devised the following plan for the upcoming week:

Refined Task Breakdown:
I will meticulously break down tasks into manageable segments, ensuring logical distribution of efforts and time across the week. This will provide a clearer picture of my progress and aid in meeting weekly goals.

Time Management Strategies:
I will implement effective time management strategies, such as setting specific time slots for various project activities, setting reminders, and adhering to a structured daily schedule.

Weekly Goal Setting:
Each week, I will set clear and achievable goals, considering the project's overall timeline. Regularly reassessing and adjusting these goals will help maintain focus and progress.

Alex

Supervisor’s signature

[illegible]

Student’s signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 6**
Student Name & ID: Loh Yi Ling (19ACB06235)	
Supervisor: Dr Ts Dr Ooi Boon Yaik	
Project Title: loT Database for non structured data type	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Coding Improvement:
After fruitful discussions with my supervisor, I made the decision to enhance the coding by allowing users to enter a folder path. This modification enables the processing of all files within the designated folder. For example, the minimum value among the files will be calculated and output to the user, providing a more streamlined and user-friendly approach.

Proposed Additional Functions:
To further aid users and enhance the functionality of my project, I started proposing more functions. These additional functions will provide users with added convenience.

Guidance from Dr. Ooi:
Dr. Ooi, my mentor, provided invaluable guidance on writing the coding for my FYP. Additionally, Dr. Ooi shared some insightful examples of IoT datasets, which will serve as valuable references in applying the acquired knowledge to my own project.

2. WORK TO BE DONE

Complete Coding Improvement:
I will finalize the coding enhancement to ensure a seamless user experience when processing files from the designated folder path.

Implement Additional Functions:
I will work on implementing the proposed additional functions to enhance the project's capabilities and provide users with more options for data analysis.

3. PROBLEMS ENCOUNTERED

During the implementation of the coding improvement, I encountered a few challenges in efficiently processing and managing multiple files from the user-provided folder path. However, with perseverance and experimentation, I was able to address these challenges and achieve the desired outcome.

4. SELF EVALUATION OF THE PROGRESS

Accomplishments:

Coding Enhancement and User Input:
I successfully implemented the coding improvement, allowing users to enter a folder path for processing multiple files. This enhancement enhances the project's usability and efficiency.

Additional Function Proposals:
I initiated the proposal of additional functions to augment the project's capabilities, providing users with more convenient and versatile data analysis options.

Areas for Improvement:

Speeding Up Project Progress:
Recognizing the urgency of completing my FYP on time, I acknowledge the need to increase my pace of work. I will strive to maintain a consistent and focused approach to meet project milestones promptly.

Extending Time Allocation for FYP:
Given the complexity and scope of my FYP, I believe the time assigned for the project should be extended to ensure adequate progress and thorough exploration of the subject matter.

Deepening Knowledge through Reading:
I realize the significance of deepening my understanding of relevant concepts and methodologies. To achieve this, I commit to dedicating more time to reading and research to bolster my knowledge base.

Alex

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 8**
Student Name & ID: Loh Yi Ling (19ACB06235)	
Supervisor: Dr Ts Dr Ooi Boon Yaik	
Project Title: loT Database for non structured data type	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Successfully developed 2 functions which are calculate mean and query by string.
2. WORK TO BE DONE Able to query from multiple data format such as JSON, CSV, and text files.
3. PROBLEMS ENCOUNTERED Inaccurate results of mean value
4. SELF EVALUATION OF THE PROGRESS N/A

[illegible]
Supervisor's signature

[illegible]
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 10**
Student Name & ID: Loh Yi Ling (19ACB06235)	
Supervisor: Dr Ts Dr Ooi Boon Yaik	
Project Title: loT Database for non structured data type	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Achieved objectives.
2. WORK TO BE DONE Comparison between FastTextProcessor and other tools FYP Report
3. PROBLEMS ENCOUNTERED Measuring the time taken of each technologies
4. SELF EVALUATION OF THE PROGRESS N/A

[illegible]
Supervisor's signature

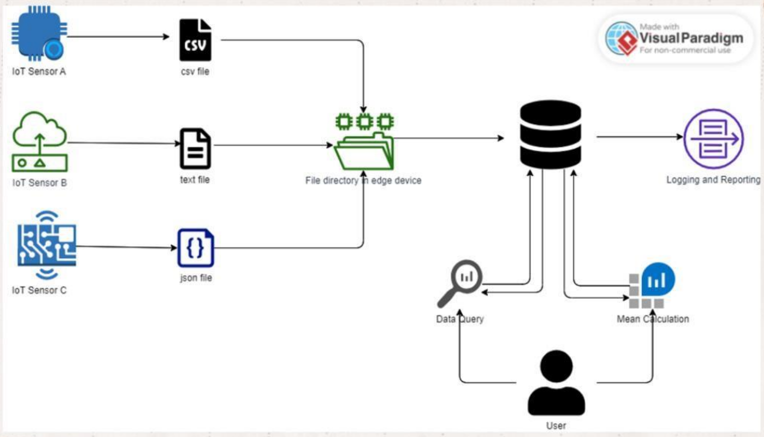
[illegible]
Student's signature

POSTER

Loh Yi Ling

IOT DATABASE FOR NON-STRUCTURED DATA TYPE

SYSTEM ARCHITECTURE DIAGRAM



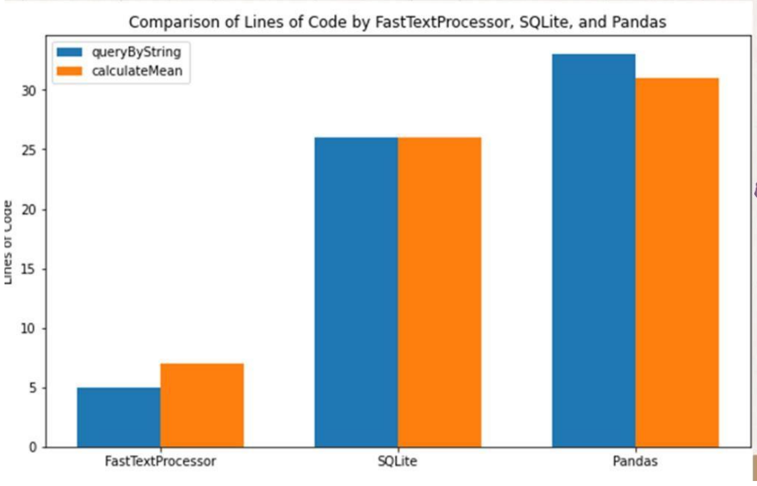
OBJECTIVES

FastTextProcessor, a Python library designed to simplify two fundamental data processing tasks: querying by string and calculating mean values.

The objectives of this project be summarized as below:

1. Implement a preprocessing program for data analytics so that data does not need to be populated before it can be queried like existing NoSQL database.
2. Develop a program that allows data to be run directly by data analytics software without formatting the data.

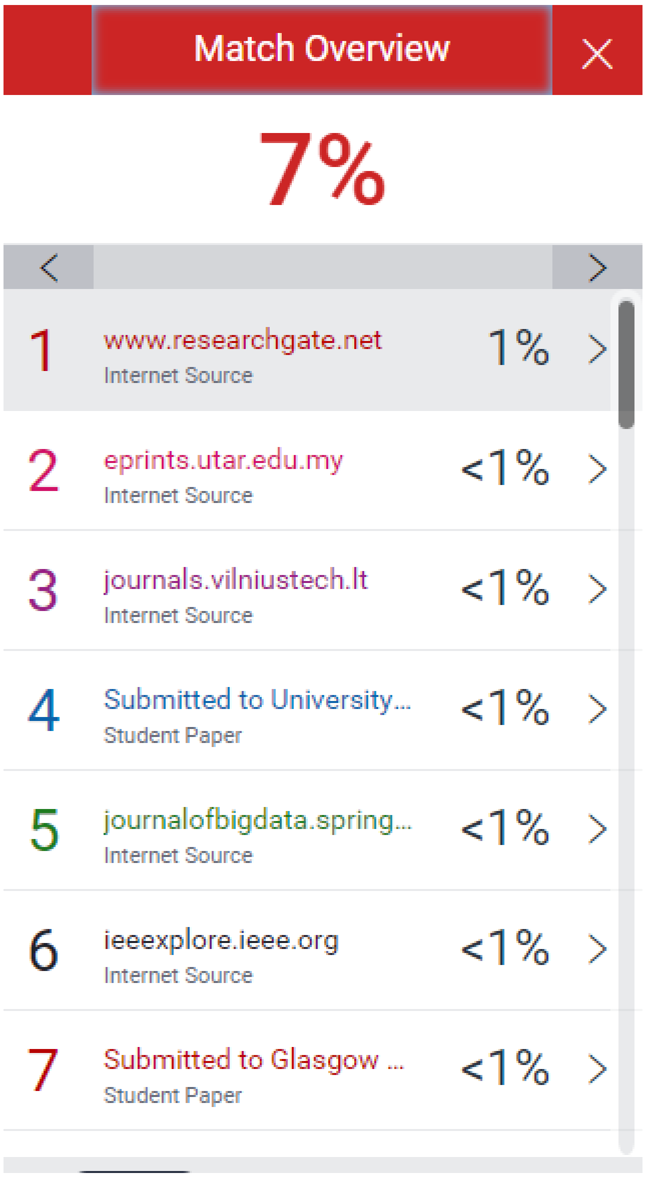
RESULTS:



FICT

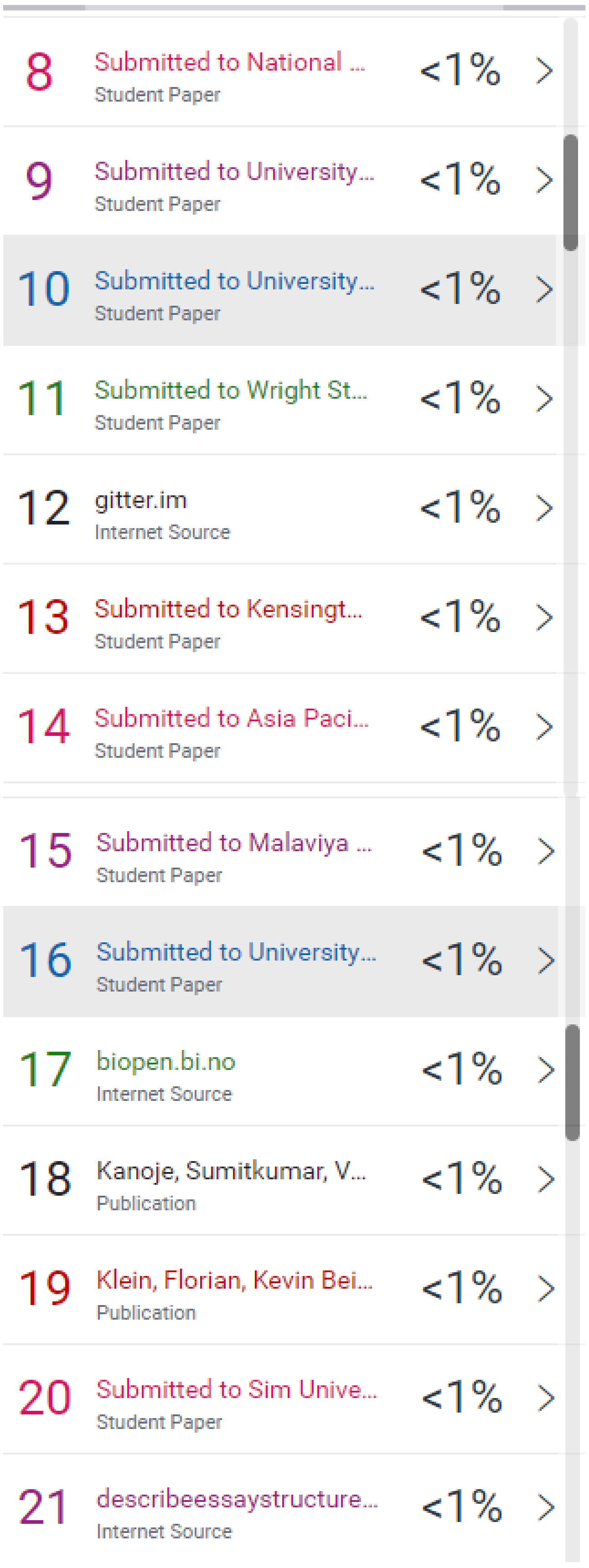
UTAR UNIVERSITI TUNKU ABDUL RAHMAN

PLAGIARISM CHECK RESULT


Match Overview

7%

1	www.researchgate.net Internet Source	1%	>
2	eprints.utar.edu.my Internet Source	<1%	>
3	journals.vilniustech.lt Internet Source	<1%	>
4	Submitted to University... Student Paper	<1%	>
5	journalofbigdata.spring... Internet Source	<1%	>
6	ieeexplore.ieee.org Internet Source	<1%	>
7	Submitted to Glasgow ... Student Paper	<1%	>



8	Submitted to National ...	Student Paper	<1%
9	Submitted to University...	Student Paper	<1%
10	Submitted to University...	Student Paper	<1%
11	Submitted to Wright St...	Student Paper	<1%
12	gitter.im	Internet Source	<1%
13	Submitted to Kensingt...	Student Paper	<1%
14	Submitted to Asia Paci...	Student Paper	<1%
15	Submitted to Malaviya ...	Student Paper	<1%
16	Submitted to University...	Student Paper	<1%
17	biopen.bi.no	Internet Source	<1%
18	Kanoje, Sumitkumar, V...	Publication	<1%
19	Klein, Florian, Kevin Bei...	Publication	<1%
20	Submitted to Sim Unive...	Student Paper	<1%
21	describeessaystructure...	Internet Source	<1%

Universiti Tunku Abdul Rahman			
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.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	LOH YI LING
ID Number(s)	19ACB06235
Programme / Course	Bachelor of Information Systems (Honours) Information Systems Engineering
Title of Final Year Project	IoT DATABASE FOR NON-STRUCTURED DATA TYPE

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 7 % **Similarity by source** Internet Sources: 4 % Publications: 3 % Student Papers: 4 %	
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: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 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.

[illegible]

Signature of Supervisor

Signature of Co-Supervisor

Name: Dr. Ooi Boon	Name:
Date: 15/9/202	Date:

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	19ACB06235
Student Name	LOH YI LING
Supervisor Name	TS DR OOI BOON YAIK

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	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 (not 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 (not 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.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]

(Signature of Student)
Date: 14/9/2023