ONLINE CAB TRIP ANALYSIS USING MACHINE LEARNING ALGORITHM

A Mini Project Report

Submitted In Partial Fulfilment of The Requirement For The Award of The Degree

Bachelor of Engineering

In

COMPUTER SCIENCE AND ENGINERING - DATA SCIENCE

Submitted by

NAME: SYED SAAD QUADRI	**1MJ20CD037**
NAME: SUFIAN BAIG	**1MJ20CD035**
NAME: VISHAL CHOUHAN	**1MJ20CD043**

Under the guidance of

REKHA P

Assistant Professor

Department of Information Science

MVJ College of Engineering, Bengaluru
(An Autonomous Institute)
JUNE 2023

MVJ COLLEGE OF ENGINEERING, BENGALURU - 560067

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DATA SCIENCE

CERTIFICATE

It is Certified that **The Mini Project** work Titled ***'Online Cab Trip Analysis'*** is carried out by **SYED SAAD QUADRI (1MJ20CD037), SUFIAN BAIG (1MJ20CD035),** and **VISHAL CHOUHAN (1MJ20CD043)** who are Confide Students of MVJ College of Engineering, Bengaluru, in partial fulfilment for the award of Degree of **Bachelor of Engineering in Computer Science & Engineering (DATA SCIENCE)** of the Visvesvaraya Technological University, Belagavi during the year 2022-2023. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the Mini Project Report deposited in the departmental library. The Mini Project report has been approved as it satisfies the Academic Requirements in respect of Mini Project work prescribed by the institution for the said Degree.

Signature of Guide **Signature of Head of the Department** **Signature of Principal**

External Viva

Name of Examiners	**Signature with Date**
1	
2	

MVJ COLLEGE OF ENGINEERING, BENGALURU-560067

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING - DATA SCIENCE

DECLARATION

We, Syed Saad Quadri, Sufian Baig, Vishal Chouhan. Students of **Sixth** Semester B.E. Department of COMPUTER SCIENCE AND ENGINEERING- DATA SCIENCE, MVJ College of Engineering, Bengaluru, hereby declare that the Mini Project titled ***'Online Cab Trip Analysis'*** has been carried out by us and submitted in partial fulfilment for the award of Degree of **Bachelor of Engineering** in **COMPUTER SCIENCE AND ENGINEERING- DATA SCIENCE** during the year 2022-2023.

Further we declare that the content of the dissertation has not been submitted previously by anybody for the award of any Degree or Diploma to any other University.

We also declare that any Intellectual Property Rights generated out of this project carried out at MVJCE will be the property of MVJ College of Engineering, Bengaluru and we will be one of the authors of the same.

Place: Bengaluru

Date:

Name	Signature
1. **SYED SAAD QUADRI (**1MJ20CD037**)**	
2. **SUFIAN BAIG (**1MJ20CD035**)**	
3. **VISHAL CHOUHAN (**1MJ20CD043**)**	

ACKNOWLEDGEMENT

We are indebted to our guide Professor **Rekha P**, Department of Information Science and Engineering, MVJ College of Engineering for her wholehearted support, suggestions and invaluable advice throughout our project work and also for the help in the preparation of this report.

Our sincere thanks to **Mr Kumar R**, Associate Professor and Head, Department of Computer Science And Engineering, MVJCE for his support and encouragement.

We express sincere gratitude to our beloved Principal, **Dr. M. Brindha,** for all her support towards this project work.

Lastly, we take this opportunity to thank our **Family** members and **Friends** who provided all the backup support throughout the project work.

ABSTRACT

Online cab trip analysis refers to the process of analysing data and information related to online cab services, such as Uber or Ola, to gain insights into various aspects of their operations. This analysis involves examining factors such as demand patterns, route optimization, driver performance, pricing strategies, customer segmentation, and service improvement. By leveraging data analytics techniques, online cab companies can make data-driven decisions to optimize resource allocation, enhance customer experience, improve operational efficiency, and identify growth opportunities. The analysis helps in predicting demand, optimizing routes, evaluating driver performance, detecting fraud, personalizing services, and expanding into new markets. Overall, online cab trip analysis enables companies to leverage data to make informed decisions and improve their services in a competitive and rapidly evolving industry.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

The advent of online cab services has transformed the way people travel, providing a convenient and accessible mode of transportation in many urban areas. With the increasing popularity and widespread adoption of these services, there is a growing need to analyze the data generated by online cab trips. Online cab trip analysis refers to the systematic study of trip records, customer feedback, driver information, and other contextual factors associated with online cab services. By analyzing this wealth of data, valuable insights can be extracted, leading to improved service efficiency, customer satisfaction, and overall operational effectiveness.

The analysis of online cab trips offers numerous benefits to both service providers and users. For service providers, understanding the spatiotemporal patterns of cab trips is crucial for optimizing fleet management and resource allocation. By analyzing trip origins, destinations, and durations, service providers can identify peak hours, popular routes, and areas with high demand. This knowledge enables them to strategically position their fleet and reduce passenger wait times, ultimately improving service efficiency and customer experience.

Moreover, analyzing customer feedback plays a vital role in identifying areas for improvement. By leveraging sentiment analysis techniques on feedback data, service providers can gain insights into customer experiences, preferences, and pain points. Understanding customer sentiments enables them to address specific issues promptly and enhance overall customer satisfaction levels. Additionally, this analysis helps service providers tailor their services to better meet customer expectations, fostering loyalty and retention.

Another crucial aspect of online cab trip analysis involves examining the behavior and performance of cab drivers. By analyzing driver-related data, such as ratings, trip histories, and performance metrics, service providers can gain insights into driver efficiency, reliability, and customer satisfaction. Identifying top-performing drivers and understanding the factors contributing to their success can guide driver selection, training programs, and incentivization strategies. This analysis not only enhances driver performance but also improves passenger safety and satisfaction.

The increasing availability of large datasets and advancements in data analytics techniques, including machine learning and predictive modeling, have greatly facilitated the analysis of online cab trips. These techniques allow for the identification of complex patterns, prediction of demand, and optimization of operational processes. The insights gained from online cab trip analysis can inform data-driven decision-making, enabling service providers to refine their strategies, enhance service quality, and ultimately gain a competitive edge in the market.

In conclusion, online cab trip analysis is a critical area of research and practice, driven by the need to understand and improve the efficiency and quality of online cab services. By harnessing the power of data analytics, service providers can gain valuable insights into trip patterns, customer satisfaction, and driver performance. This knowledge can inform strategic decision-making, leading to enhanced service efficiency, improved customer experiences, and a more sustainable transportation ecosystem. As online cab services continue to evolve, the analysis of trip data will remain an essential tool for unlocking new opportunities and driving innovation in the transportation industry.

CHAPTER 2

Theory And Concepts

2.1 Collection of Data

The data collection for the online cab trip analysis project includes gathering various information such as trip details (trip ID, pick-up location, drop-off location, trip duration, distance traveled, fare amount, and time of the trip), user information (user ID, age, gender, and previous trip history), weather data (temperature, precipitation, weather conditions), time and date information (day of the week, time of day, and date), driver information (driver ID, experience level, ratings, feedback), traffic data (average speed, congestion levels), and other factors like special events or public holidays. With this data, the theoretical framework for analysis involves several key components. These include demand analysis to identify patterns in cab bookings based on time, day, and weather, supply analysis to assess driver availability, trip duration and distance analysis to understand the factors influencing trip length, fare analysis to examine pricing dynamics, customer satisfaction analysis to evaluate user ratings and feedback, driver performance analysis to assess driver ratings and feedback, and efficiency analysis to evaluate the overall effectiveness of the cab service. By analyzing these factors, the project aims to gain insights into the dynamics of online cab trips and identify areas for improvement and optimization.

2.2 Data Pre-Processing

The data processing for the online cab trip analysis project involves several steps to transform and analyze the collected data. Initially, the raw data is cleaned by removing any duplicates, outliers, or missing values. Then, data normalization techniques may be applied to ensure consistency and comparability across different variables. Next, feature engineering techniques can be employed to derive additional meaningful features from the existing data, such as calculating average trip duration or creating time-based features. Once the data is cleaned and preprocessed, statistical and exploratory data analysis techniques can be utilized to gain insights into the dataset. This may involve calculating descriptive statistics, creating visualizations, and identifying correlations between variables. Machine learning algorithms can also be employed to build predictive models or uncover hidden patterns in the data. Clustering algorithms may be used to group similar trips based on different criteria. Additionally, techniques like regression analysis or time series analysis can be applied to forecast demand or predict future trip characteristics. Finally, the results and insights obtained from the data processing steps can be presented in the form of reports, dashboards, or visualizations to facilitate decision-making and drive improvements in the online cab service.

2.3 <u>Cleaning Of Missing Data</u>

Cleaning missing data is a crucial step in data pre processing to ensure the accuracy and reliability of your analysis. When handling missing data in your dataset, it is important to identify the missing values and understand the reasons behind their absence. Missing values can occur due to various factors, such as data entry errors, technical issues, or genuine missing information.

There are several approaches to handle missing data. One option is to delete the corresponding rows or columns containing missing values. However, this approach should be used with caution as it may lead to a loss of valuable information. Another approach is imputation, which involves filling in the missing values with estimated or inferred values. Common imputation techniques include mean, median, or mode imputation, where missing values are replaced with the respective attribute's mean, median, or mode. Regression imputation utilizes regression models to predict missing values based on other attributes. Multiple imputation generates multiple plausible imputations by modeling the missing values using statistical techniques.

When imputing missing values, it is essential to consider the potential bias introduced into the dataset. Imputed values may not perfectly represent the true values, which can impact subsequent analysis. Documenting the imputation process and conducting sensitivity analyses can help assess the influence of imputation on the results.
Evaluate the impact of missing data on your analysis and consider conducting separate analyses with and without imputed values to understand their effect. Transparent documentation of the data cleaning process, including the imputation method used, the

percentage of missing values, and any assumptions made, ensures transparency and reproducibility.

It is important to select the appropriate missing data handling technique based on your dataset's characteristics and analysis objectives. Carefully considering and addressing missing data ensures the reliability and validity of your findings.

Messaging (or MSNG) is a Python library that provides a convenient and efficient way to implement messaging systems in Python applications. It simplifies the process of sending and receiving messages between different components or services within an application or across distributed systems. The MSNG library offers various messaging patterns and protocols, such as publish-subscribe, request-reply, and message queues, enabling seamless communication and coordination between different parts of an application. With its user-friendly API and robust functionality, the MSNG library empowers developers to build scalable and reliable messaging solutions in Python.

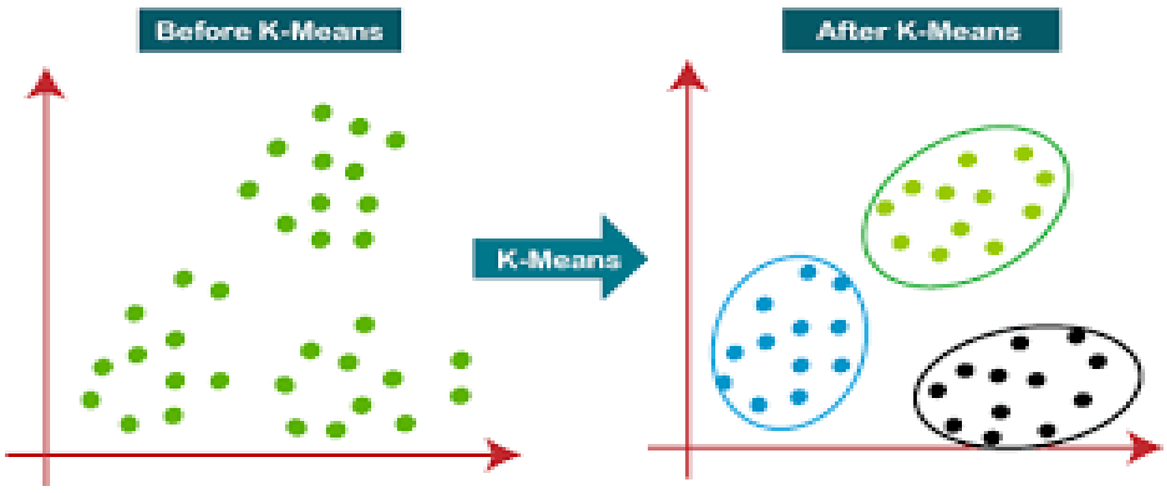
2.4 Clustering Methodology

The K-means clustering algorithm is widely used in data analysis and machine learning for grouping similar data points together. It is particularly useful in online cab trip analysis to identify patterns in trip origins and destinations based on their geographical proximity.

The algorithm begins by randomly selecting a predefined number of centroids, usually denoted as K. These centroids serve as the initial cluster centers. Each data point, in this case, a trip origin or destination, is then assigned to the nearest centroid based on its spatial distance. The Euclidean distance or other distance metrics are commonly used for this purpose.

After assigning data points to the centroids, the algorithm updates the positions of the centroids by recalculating them as the mean of all data points assigned to each centroid. This process is iteratively repeated, with data points being reassigned and centroids updated, until convergence criteria are met. The convergence criteria can be based on a maximum number of iterations or when the centroids no longer change significantly.

The final result of the K-means clustering algorithm is a set of K clusters, each represented by a centroid. Each data point is assigned to its respective cluster based on the nearest centroid. The algorithm aims to minimize the within-cluster variance or the sum of squared distances between data points and their assigned centroids.



In online cab trip analysis, K-means clustering can be used to identify clusters of trips with similar start or end locations. By grouping trips based on their geographical proximity, valuable insights can be obtained. This information can assist in optimizing fleet distribution, identifying popular routes, or analyzing trip patterns based on different geographical regions.

However, it's important to note that the effectiveness of the K-means algorithm depends on appropriate initialization of centroids and the choice of the optimal number of clusters (K). The initialization of centroids can impact the convergence and quality of the resulting clusters. Additionally, selecting the right number of clusters requires careful consideration and validation based on the specific dataset and objectives of the online cab trip analysis.

Overall, the K-means clustering algorithm offers a powerful and widely used approach for analyzing online cab trip data. By leveraging its ability to group similar trips based on geographical proximity, service providers can gain valuable insights to enhance operational efficiency, optimize resource allocation, and improve the overall quality of their services.

CHAPTER 3

System Requirements and Specifications

3.1 Hardware Requirements

- **Processor:** Pentium IV or above
- **RAM:** 2GB or above
- **Hard Disk:** 500GB or above
- **Input Device:** keyboard, mouse
- **Output Device:** Monitor

4.1 Software Requirements

- **Operating System:** Windows 7 or above
- **Coding language:** Python Programming
- **Visual Studio Code**

4.2 Functional Requirements

- **Data Collection:** Develop a system or mechanism to collect and store relevant data, including trip information (pick-up location, drop-off location, trip duration, distance travelled, fare amount), user information (user ID, age, gender), weather data, time and date information, driver information, traffic data, and any additional factors deemed necessary.
- **Data Pre processing:** Implement data pre processing techniques to clean and prepare the collected data for analysis, including handling duplicates, outliers, missing values, and normalizing the data if required.
- **Demand Analysis:** Build algorithms or models to analyse the demand for cab trips based on factors such as time of day, day of the week, weather conditions, and other variables. This analysis should provide insights into peak demand periods, factors influencing booking frequency, and trends over time.

- **Supply Analysis:** Develop mechanisms to analyze the availability of cabs and drivers in different areas and at different times. This analysis should consider factors such as driver availability, response time, and geographic coverage.
- **Trip Duration and Distance Analysis:** Create algorithms or models to analyze trip durations and distances, considering factors such as time of day, day of the week, traffic conditions, and other relevant variables. This analysis should provide insights into the factors affecting trip length and help identify pattern
- **Reporting and Visualization:** Design a reporting and visualization system to present the analyzed data and insights in a clear and user-friendly manner. This system should include charts, graphs, dashboards, and other visual representations that facilitate understanding and decision-
making.

4.3 Non-Functional Requirements

- **Performance:** The system should be able to handle large volumes of data efficiently and provide timely analysis and insights. It should be designed to perform computations and data processing tasks within acceptable time frames.
- **Scalability:** The system should be scalable to accommodate potential growth in data volume and user demand. It should be able to handle increasing amounts of data and users without significant degradation in performance.
- **Reliability:** The system should be reliable and available for data analysis consistently. It should minimize the occurrence of system failures, errors, and downtimes. Backup and recovery mechanisms should be in place to ensure data integrity and availability.
- **Security:** The system should ensure the security and privacy of the collected data. It should implement appropriate measures to protect data against unauthorized access, breaches, and data leaks. User authentication and access control mechanisms should be employed to restrict data access to authorized personnel.

CHAPTER 4

System Design

4.1 System Architecture

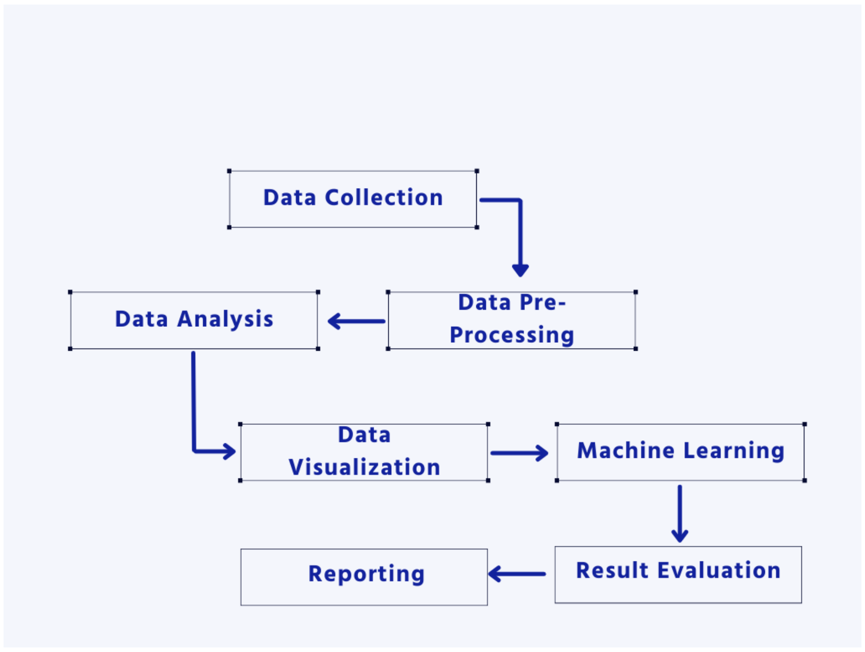
he system architecture for the online cab trip analysis project consists of a data collection layer to gather trip information, user data, and other relevant data. The collected data is preprocessed to clean, normalize, and engineer features. The processed data is then analyzed using statistical analysis and machine learning techniques to extract insights on demand, supply, trip duration, fare, customer satisfaction, and driver performance. The results are visualized through charts, graphs, and interactive dashboards, and reports are generated for stakeholders. The architecture includes infrastructure, integration, and security layers to ensure scalability, data integration, and data protection.

- **Data Collection Layer:** This layer involves mechanisms to collect and store the required data for analysis. It may include APIs or data ingestion processes to retrieve trip information, user data, weather data, and other relevant data from various sources. The collected data can be stored in a database or a data lake for further processing.
- **Data Preprocessing Layer:** In this layer, the collected data undergoes preprocessing steps to clean and prepare it for analysis. This includes removing duplicates, handling missing values, normalizing data, and performing feature engineering techniques. Data preprocessing can be done using tools like Python libraries (e.g., Pandas) or data integration platforms.
- **Data Analysis Layer:** This layer involves performing various analyses on the preprocessed data. Different analysis techniques, such as statistical analysis, machine learning algorithms, or clustering methods, can be applied to extract meaningful insights and patterns from the data. This layer may include modules for demand analysis, supply analysis, trip duration analysis, fare analysis, customer satisfaction analysis, driver performance analysis, and efficiency analysis.
- **Data Visualization Layer**: After the data analysis, this layer focuses on creating visualizations, reports, and dashboards to present the analyzed data and insights in a user-friendly and informative manner. Visualization tools such as Tableau,

Power BI, or Python libraries like Matplotlib and Seaborn can be utilized to create interactive charts, graphs, and visual representations.

- **Reporting and Presentation Layer:** This layer involves generating reports, presentations, or interactive dashboards to communicate the findings and outcomes of the analysis. These reports can be accessed by stakeholders such as management, operations teams, or decision-makers to gain insights and make informed decisions.
- **Infrastructure Layer:** The infrastructure layer encompasses the underlying technology infrastructure required to support the system. This includes servers, cloud platforms (such as Amazon Web Services or Microsoft Azure), databases, and storage systems to store and process the data. The infrastructure should be scalable, secure, and reliable to handle the volume of data and ensure system performance.
- **Integration Layer:** This layer involves integrating the analysis system with other external systems or APIs to access additional data sources, such as traffic data, public events, or external analytics tools. This integration enables a more comprehensive analysis by incorporating relevant external data.
- **Security and Privacy:** Throughout the system architecture, security measures should be implemented to protect the collected data and ensure user privacy. This includes user authentication, data encryption, access controls, and compliance with data protection regulations.

4.2 Flowchart



The flowchart outlines the key steps involved in your online cab trip analysis project. It begins with data collection, where relevant information such as trip details, user information, weather data, and more are collected and stored. The next step is data pre processing, which involves cleaning the collected data and preparing it for analysis by removing duplicates, outliers, and missing values. Feature engineering techniques may also be applied to derive additional meaningful features. The flow then moves to data analysis, where various aspects such as demand analysis, supply analysis, trip duration and distance analysis, fare analysis, customer satisfaction analysis, driver performance analysis, and efficiency analysis are conducted. These analyses help extract insights and patterns from the data. The subsequent step is data visualization, where charts, graphs, and visualizations are created to present the obtained insights in a clear and concise manner. Machine learning techniques can be applied for predictive modeling or pattern discovery. The results and insights obtained from the analysis are then evaluated, and reports, dashboards, or presentations are prepared to communicate the project outcomes. The flowchart concludes with the end of the project, signifying the completion of the analysis phase.

CHAPTER 5

IMPLEMENTATION

The phase of the project known as implementation is when the hypothetical setup is converted into a functional framework. To produce the required end result, implementation must be an exact translation of the design document into a suitable programming language. Choosing the wrong programming language to implement or using an inappropriate programming technique frequently results in the product being damaged. It is preferable for the coding phase to be directly related to the design phase in the sense that implementation should ideally be carried out in an object-oriented manner if the design is done so. The implementation involves:

- Careful planning.
- Investigation of the current system and the constraints on implementation.
- Training of the team in the newly developed system.

At this point, the design or design modifications are presented and operationalized in a particular circumstance. Any software implementation is always preceded by crucial choices on the platform to be utilized, the language to be used, etc. Numerous considerations, including the actual environment in which the system operates, the desired speed, security concerns, and other implementation-specific features, frequently have an impact on these choices. Before this project was put into action, three key implementation decisions were taken. They are as follows:

- Analysis of the hardware and software components.
- Selection of the programming language for development of the application.
- Coding guidelines to be followed.

The modules used to design the chatbot are as follows:

- Streamlit
- Pandas
- Matplotlib
- Missingno
- Seaborn

Python is employed as a programming language, while Streamlit is used to connect the frontend and backend components. The Data Analysis front end was created using

Streamlit and its back end analysis was created, trained, developed, and tested using the Kmeans Algorithm. Users can upload their CSV file using file_uploader and the applications generates comprehensive reports or interactive dashboards that summarize the analysis findings and make them accessible to users.

5.1 Streamlit

Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers. Data scientists or machine learning engineers are not web developers and they're not interested in spending weeks learning to use these frameworks to build web apps. Instead, they want a tool that is easier to learn and to use, as long as it can display data and collect needed parameters for modeling. Streamlit allows you to create a stunning-looking application with only a few lines of code.

The best thing about Streamlit is that you don't even need to know the basics of web development to get started or to create your first web application. So if you're somebody who's into data science and you want to deploy your models easily, quickly, and with only a few lines of code, Streamlit is a good fit.

One of the important aspects of making an application successful is to deliver it with an effective and intuitive user interface. Many of the modern data-heavy apps face the challenge of building an effective user interface quickly, without taking complicated steps. Streamlit is a promising open-source Python library, which enables developers to build attractive user interfaces in no time.

Streamlit is the easiest way especially for people with no front-end knowledge to put their code into a web application:

- No front-end (html, js, css) experience or knowledge is required.
- You don't need to spend days or months to create a web app, you can create a really beautiful machine learning or data science app in only a few hours or even minutes.
- It is compatible with the majority of Python libraries (e.g. pandas, matplotlib, seaborn, plotly, Keras, PyTorch, SymPy(latex)).

Installing Streamlit

It is highly recommended to use a virtual environment before installing RASA. The command in anaconda for creating the virtual environment is:

```
conda create -n myenv python=3.8
conda activate myenv
```

Now, we will install streamlit in our environment. We will install it using pip install.

```
pip install streamlit
```

Streamlit primarily focuses on providing a framework for creating interactive web applications using Python. Once you have a Streamlit script (Python file), you can run it using the streamlit run command followed by the filename. For example:

```
Streamlit run myapp.py
```

The file directory of a Streamlit application typically consists of the following files and directories:

1. Python script: This is the main Python file where you write your Streamlit application code. It usually has a .py extension, such as app.py. This is where you define the layout, widgets, and functionality of your Streamlit app.

2. Requirements file: This file lists the Python dependencies required by your Streamlit application. It is usually named requirements.txt or environment.yml (for conda). You can specify the required packages and their versions in this file.

3. Data files: If your Streamlit app uses any data files, such as CSV, JSON, or other formats, you may include them in the file directory.

4. Static assets: If your Streamlit app includes static assets like images, CSS stylesheets, or JavaScript files, you can place them in a directory named static within the file directory. You can reference these assets in your app's code using relative paths.

5. Other files and directories: Depending on your specific application, you may have additional files or directories. For example, if you are using machine learning models, you might have a directory for storing those models. If you are using a

database, you might have a configuration file for connecting to the database.

Streamlit File Uploader Function

The st.file_uploader function in Streamlit is a valuable tool that empowers users to upload files directly within a Streamlit application. This interactive widget enables users to select and upload files from their local machines to the Streamlit app's runtime environment. The st.file_uploader function takes parameters such as a label to describe the purpose of the file uploader and the option to specify the allowed file types. By default, it allows uploading a single file, but it can be configured to accept multiple files. When a file is uploaded, the function returns a file-like object that can be utilized for further processing, such as reading and analyzing the uploaded file's contents. Streamlit's st.file_uploader simplifies the process of incorporating file upload functionality into applications, facilitating seamless user interaction and enabling data exploration or analysis workflows. The **syntax** of st.file_uploader is as follows:

```
file = st.file_uploader(label, type=None, accept_multiple_files=False, key=None)
```

The important parameters of st.file_uploader are:

label: The label or display text for the file uploader widget. It is typically a string that describes the purpose or instructions for file upload.

type: An optional parameter that restricts the file types that can be uploaded. It can take a string or list of strings specifying the allowed file extensions, such as '.csv', ['.jpg', '.png'], or ['.txt', '.pdf', '.docx']. If not provided, all file types can be uploaded.

accept_multiple_files: A boolean parameter that determines whether multiple files can be uploaded at once. By default, it is set to False, meaning only a single file can be selected for upload. If set to True, multiple files can be selected using the file dialog.

key: An optional parameter that allows you to uniquely identify the file uploader widget. It can be useful when interacting with other widgets or managing the state of the application.

The st.file_uploader function returns a file-like object or None if no file is uploaded. You can use the returned file object to perform further operations, such as reading and processing the uploaded file data.

Here is an example usage of st.file_uploader:

```
import streamlit as st

uploaded_file = st.file_uploader("Upload a CSV file")
if uploaded_file is not None:
    data = pd.read_csv(uploaded_file)
    st.write(data)
```

5.2 Pandas DataFrame fillna() Method

The fillna() method in pandas is a powerful tool for handling missing or NaN (Not a Number) values in a DataFrame or Series. Missing data is a common issue when working with real-world datasets, and fillna() allows us to replace these missing values with specified values or strategies.

One way to use fillna() is to replace missing values with a specific scalar value. For example, we can fill all the missing values in a column with zero using df['column_name'].fillna(0, inplace=True). This approach is helpful when we want a consistent value across the dataset.

Another useful option is to fill missing values with statistics such as the mean or median of the column. For instance, df['column_name'].fillna(df['column_name'].mean(), inplace=True) replaces NaN values with the mean value of that column. This strategy provides a more meaningful estimate based on the available data.

The fillna() method also offers methods like 'ffill' and 'bfill' to propagate previous or next values respectively. By setting method='ffill', missing values are filled with the preceding non-null value. Conversely, method='bfill' fills NaN values with the subsequent non-null value. These methods are useful when dealing with time-series data or sequentially ordered data.

Additionally, the fillna() method allows limiting the number of consecutive NaN values

filled using the limit parameter. For example,
df['column_name'].fillna(df['column_name'].mean(), limit=2, inplace=True)
limits the filling of consecutive NaN values to a maximum of two.

By understanding and utilizing the fillna() method effectively, we can handle missing data appropriately, ensuring the integrity and reliability of our analysis and models.

Syntax

```
dataframe.fillna(value, method, axis, inplace, limit, downcast)
```

The important parameters of the fillna() method are:

- value: The value to replace the missing or NaN values with. It can be a scalar value, a dictionary, a Series, or a DataFrame.
- method: Specifies a filling method. It can take values like 'pad', 'ffill', 'bfill', 'backfill', or 'nearest'. These methods propagate the previous or next value to fill the NaN values accordingly.
- axis: Specifies the axis along which the fillna() operation is performed. By default, it fills values column-wise (axis=0). To fill values row-wise, you can set axis=1.
- inplace: If set to True, the DataFrame or Series is modified in place, and the original data is changed. By default, it is set to False, which returns a new DataFrame or Series with the filled values.
- limit: Limits the number of consecutive NaN values filled. For example, if you set limit=2, it will fill a maximum of two consecutive NaN values.
- downcast: Specifies a type-casting strategy for the resulting data. It can take values like 'infer', 'integer', 'signed', 'unsigned', etc.

Example:

```
import pandas as pd
# Fill missing values with a scalar value
df['column_name'].fillna(0, inplace=True)

# Fill missing values with the mean of the column
df['column_name'].fillna(df['column_name'].mean(), inplace=True)

# Fill missing values with the previous value
df.fillna(method='ffill', inplace=True)

# Fill missing values with the next value
df.fillna(method='bfill', inplace=True)

# Fill missing values with the mean of the column, limit to 2 consecutive values
df['column_name'].fillna(df['column_name'].mean(), limit=2, inplace=True)
```

5.3 Missingno

Missingno is a Python library that provides a convenient way to visualize and analyze missing data in a dataset. It offers a range of useful tools for understanding the patterns and extent of missing values in your data. The library is built on top of Matplotlib and Seaborn, making it easy to integrate into your data analysis workflow. With Missingno, you can generate informative visualizations such as heatmaps and bar charts to identify missing data patterns and assess the completeness of your dataset. These visualizations allow you to quickly grasp the distribution of missing values across variables and gain insights into potential data quality issues. Additionally, Missingno provides functions to handle missing data, such as dropping missing rows or filling missing values with different strategies. Overall, Missingno is a valuable tool for data exploratory analysis, enabling you to effectively understand and handle missing data in your datasets.

The **syntax** for using Missingno library in Python is as follows:

```
import missingno as msno

# Visualize missing data pattern with heatmap
msno.heatmap(df)

# Visualize missing data pattern with bar chart
msno.bar(df)
```

Here's an
example of how to use Missingno library to visualize missing data:

```
import pandas as pd
import missingno as msno

# Create a sample DataFrame with missing values
data = {
   'A': [1, 2, None, 4, 5],
   'B': [None, 2, 3, 4, None],
   'C': [1, 2, 3, None, None]
}
df = pd.DataFrame(data)
```

5.4 Matplotlib

Matplotlib is a popular data visualization library in Python that provides a wide range of tools for creating static, animated, and interactive visualizations. It is widely used for generating high-quality plots, charts, histograms, scatter plots, and other types of visualizations to explore and communicate data effectively.

Key **features** and **components** of Matplotlib include:

Figure and Axes: Matplotlib follows a hierarchical structure where a Figure object represents the entire figure or window, and within the figure, one or more Axes objects are created. Axes objects represent the coordinate system in which data is plotted.

Plotting Functions: Matplotlib provides a variety of functions for creating different types of plots, such as line plots, bar plots, scatter plots, histogram plots, pie charts, and more. These functions
allow you to customize various aspects of the plot, including colors, markers, line styles, labels, and annotations.
Customization Options: Matplotlib offers extensive customization options to control the appearance of plots. You can modify properties such as colors, line styles, fonts, axes limits, tick marks, grid lines, legends, and titles to match your specific requirements. Matplotlib provides a vast range of options for fine-tuning the visual aesthetics of your plots.

Subplots: Matplotlib allows you to create multiple subplots within a single figure. This is useful when you want to display multiple plots side by side or in a grid layout, making it easier to compare and analyze different aspects of the data.

Matplotlib Interfaces: Matplotlib provides two main interfaces for creating plots - a functional interface and an object-oriented interface. The functional interface allows you to create plots using functions like plot(), scatter(), and hist(). The object-oriented interface provides more control and flexibility, where you directly create Figure and Axes objects and call methods on them to create and customize the plots.

Integration with NumPy and Pandas: Matplotlib seamlessly integrates with other popular libraries such as NumPy and Pandas. It can directly plot data stored in NumPy arrays or Pandas data structures, making it easy to visualize and analyze numerical data.

Overall, Matplotlib is a powerful and versatile library for data visualization in Python, providing a wide range of plotting capabilities and customization options. It is extensively used in various fields, including scientific research, data analysis, machine learning, and data visualization tasks.
Here's a brief overview of the syntax and an example of creating a simple line plot using Matplotlib:

```
import matplotlib.pyplot as plt

# Create data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a figure and axes
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y)
# Customize the plot
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Line Plot')

# Show the plot
plt.show()
```

First, import the matplotlib.pyplot module, commonly aliased as plt, which provides the plotting functions and methods.

Create the data for the x-axis and y-axis. In this example, x represents the x-axis values and y represents the corresponding y-axis values.

Create a figure and axes using the subplots() function. This will create a blank canvas (figure) and an axes object (ax) on which you'll plot your data.

Use the plot() method of the axes object (ax) to plot the data. Pass the x and y values as arguments to the plot() function. This will create a line plot connecting the points defined by the (x, y) pairs.

Customize the plot by setting labels for the x-axis and y-axis using the set_xlabel() and set_ylabel() methods, respectively. Set the title of the plot using the set_title() method. Finally, use plt.show() to display the plot.

5.5 Seaborn

Seaborn is a powerful data visualization library in Python that is built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn offers a wide range of visualizations and statistical tools to explore and analyze data effectively.

With Seaborn, you can create various types of plots such as scatter plots, line plots, bar plots, histograms, box plots, heatmaps, and many more. It offers a rich set of options to customize the aesthetics of your plots, including color palettes, themes, and styles.

Seaborn simplifies the process of creating complex visualizations by providing easy-to-use functions and a consistent API. It integrates well with Pandas, making it convenient to work with DataFrame objects. Seaborn also has built-in support for handling categorical data, creating multi-plot grids, and visualizing statistical relationships.

One of the key strengths of Seaborn is its ability to create visually appealing and informative statistical visualizations with just a few lines of code. It automates many aspects of plot creation, such as handling missing data, adding informative labels, and displaying statistical summaries.
Overall, Seaborn is a valuable tool for data exploration and presentation. Whether you are

a data scientist, analyst, or researcher, Seaborn can help you visualize and communicate your data effectively, making it an essential library in the Python data visualization ecosystem.

This is a complete **Example** that demonstrates the syntax and usage of Seaborn:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load sample data from Seaborn
tips = sns.load_dataset('tips')
# Set the plot style
sns.set_style('whitegrid')

# Create a scatter plot
sns.scatterplot(x='total_bill', y='tip', data=tips)
```

Pandas

Pandas is a powerful and popular open-source library in Python for data manipulation and analysis. It provides data structures and functions to efficiently work with structured data, such as tables and time series data. Pandas introduces two primary data structures: Series and DataFrame. A Series is a one-dimensional labeled array capable of holding any data type, while a DataFrame is a two-dimensional tabular data structure consisting of rows and columns, similar to a spreadsheet or SQL table.

Pandas offers a wide range of functionalities for data cleaning, preprocessing, transformation, and analysis. It provides convenient methods for handling missing data, merging and joining datasets, reshaping data, applying mathematical operations, filtering, sorting, and grouping data, as well as handling date and time data. Pandas also integrates well with other popular data analysis libraries in Python, such as NumPy, Matplotlib, and scikit-learn.

With its intuitive and expressive syntax, Pandas simplifies many common data manipulation tasks, allowing users to efficiently explore, manipulate, and analyze large datasets. Whether it's data cleaning, data wrangling, or data analysis, Pandas is a go-to library for many data scientists and analysts due to its versatility, performance, and ease

Syntax and Example

```
import pandas as pd

# Creating a DataFrame from a dictionary
data = {'Name': ['John', 'Jane', 'Mike', 'Emily'],
        'Age': [25, 30, 35, 28],
        'City': ['New York', 'London', 'Paris', 'Sydney']}

df = pd.DataFrame(data)

# Displaying the DataFrame
print(df)
```

5.6 Methods For Clustering Efficiency

Confusion Matrix

A confusion matrix is a widely used tool in machine learning and classification tasks that provides a comprehensive evaluation of the performance of a predictive model. It is a square matrix that presents a summary of the predicted and actual classifications made by the model. The matrix is organized into rows and columns, with each row representing the instances in a predicted class, and each column representing the instances in an actual class.

The confusion matrix provides valuable insights into the model's performance by displaying the counts of true positives, true negatives, false positives, and false negatives. These counts allow us to calculate various evaluation metrics, such as accuracy, precision, recall, and F1 score, which provide a more detailed understanding of the model's behavior.The true positives (TP) represent the cases where the model correctly predicted a positive class, while true negatives (TN) represent the cases where the model correctly predicted a negative class. On the other hand, false positives (FP) occur when the model incorrectly predicted a positive class, and false negatives (FN) occur when the model incorrectly predicted a negative class.

By analyzing the values in the confusion matrix, we can assess the model's ability to accurately classify instances from different classes. It helps us identify any patterns of misclassification and understand the specific types of errors made by the model. This information can be used to fine-tune the model, adjust decision thresholds, or focus on specific classes that require improvement.

Overall, the confusion matrix is a powerful tool for evaluating the performance of a predictive model, enabling us to gain valuable insights into its strengths and weaknesses. It serves as a foundation for making informed decisions about model improvement, feature engineering, or adjusting classification thresholds to optimize the model's performance in real-world applications.

Silhouette Analysis

Silhouette analysis is a technique used to assess the quality and consistency of clusters formed by a clustering algorithm. It provides a measure of how well each data point fits within its assigned cluster and helps to determine the optimal number of clusters in unsupervised learning tasks. The analysis is based on calculating silhouette coefficients for each data point.

The silhouette coefficient measures the compactness and separation of a data point with respect to its own cluster compared to other neighboring clusters. A high silhouette coefficient close to 1 indicates that the data point is well-matched to its own cluster and well-separated from neighboring clusters. On the other hand, a low silhouette coefficient close to -1 suggests that the data point may have been assigned to the wrong cluster, and the clusters may be overlapping or poorly separated.

By calculating silhouette coefficients for different numbers of clusters, we can identify the number of clusters that results in the highest average silhouette coefficient. The optimal number of clusters corresponds to the point where the silhouette score is maximized, indicating the best balance between intra-cluster cohesion and inter-cluster separation.

Silhouette analysis provides a valuable visualization tool for understanding the structure and validity of clustering results. It helps in determining the appropriate number of clusters and can guide the selection of the optimal clustering algorithm. Additionally, it enables comparisons between different clustering approaches and facilitates the interpretation and evaluation of clustering outcomes.

In summary, silhouette analysis plays a crucial role in assessing the quality of clustering results by measuring the coherence and separation of data points within and between clusters. It aids in identifying the optimal number of clusters and enhances our understanding of the underlying patterns and structures in unlabeled datasets.

Elbow Method

The elbow method is a popular technique used to determine the optimal number of clusters in unsupervised machine learning tasks, particularly in clustering algorithms such as K-means. The method derives its name from the shape of the graph generated during the analysis, which resembles an elbow.

The elbow method works by evaluating the sum of squared distances (also known as the within-cluster sum of squares) for different values of k, the number of clusters. For each value of k, the algorithm calculates the total sum of squared distances between each data point and its assigned cluster centroid. This value represents the overall compactness or cohesion of the clusters.

During the analysis, the sum of squared distances is plotted against the number of clusters. Initially, as the number of clusters increases, the sum of squared distances tends to decrease rapidly. This is because adding more clusters allows for better fitting of individual data points.

However, at a certain point, adding more clusters does not lead to a significant decrease in the sum of squared distances. This is where the "elbow" occurs in the graph. The elbow point represents the optimal number of clusters, as it signifies the trade-off between minimizing the sum of squared distances and avoiding excessive complexity or overfitting.The elbow method provides a useful visual tool for selecting the optimal number of clusters in unsupervised learning tasks. It helps to strike a balance between capturing meaningful patterns in the data and avoiding unnecessary complexity. By leveraging the elbow method, researchers and practitioners can make informed decisions regarding the number of clusters to use in clustering algorithms, enhancing the interpretability and effectiveness of the clustering results.

5.7 Algorithm

Here is the procedure to follow based on which proposed Data Analysis works.

Input : data: The dataset to be clustered

n_clusters: The desired number of clusters

Output : centroids: The final centroid positions

labels: The assigned labels for each data point

Pseudo code :

```
1. Randomly initialize n_clusters centroids from the dataset.
2. Initialize an empty list labels to store the assigned labels for each data point.
3. Set iteration to 0.
4. Repeat the following steps until convergence or a maximum number of iterations:
      o Increment iteration by 1.
      o Assign each data point to the nearest centroid by calculating the Euclidean
        distance.
            ▪ For each data point x in the dataset:
            ▪ Compute the Euclidean distance between x and each centroid.
            ▪ Assign x to the cluster with the closest centroid.
            ▪ Add the assigned label to the labels list.
      o Update the centroid positions by calculating the mean of all data points
        assigned to each centroid.
            ▪ For each cluster c from 1 to n_clusters:
                  • Compute the mean of all data points assigned to cluster c.
                  • Update the centroid position of cluster c with the computed
                    mean.
      o Check for convergence by comparing the new centroid positions with the
        previous positions. If they are the same, exit the loop.
      o If the maximum number of iterations is reached, exit the loop.
5. Return the final centroid positions (centroids) and the assigned labels for each data
   point (labels)
```

CHAPTER 6

Software Testing

Testing is done to look for mistakes. Testing is the process of looking for any flaws or weaknesses in a piece of work. It offers a means of examining the operation of parts, subassemblies, assemblies, and/or a finished product. It is the process of testing software to make sure that it satisfies user expectations and meets requirements without failing in an unacceptable way. Software testing is an examination done to tell users about the quality of the software service or product being tested. In order for the application domain to realize and comprehend the dangers of software deployment, software testing also gives an unbiased, independent view of the software. The practice of running a computer program with the goal of detecting software bugs is just one example of a test technique (errors or other defects). Software testing is also known as the procedure for validating and confirming that a piece of software, be it an application, program, or other thing:

- Works during execution in the majority or all conditions.
- Meets the technical standards that determined how it was designed and developed.
- Works as planned.
- Is possible to execute with the same qualities.
- Works during execution in the majority or all conditions.

This chapter provides a summary of each testing technique used to produce a system free of bugs. The product can be tested for quality using a variety of methods at various project development stages. Different test types exist. Every test type responds to a certain testing requirement.

6.1 Levels of Testing

UNIT TESTING

Every module was tested separately. Unit testing concentrates verification work on the module, the smallest unit of software design. The developed programme is made up of various modules. Each module carries out a distinct task. The precise control flow for each module was confirmed via this type of testing. Important control pathways are checked to find faults inside the module's border using thorough design considerations as a reference.

INTEGRATION TESTING

Software components that have been merged are tested in integration tests to see if they genuinely operate as a single program. Testing is event-driven and focuses more on the fundamental result of screens or fields. Unit testing of the individual components and integration testing both indicate that the combination of the components is accurate and reliable. Integration testing is especially designed to highlight issues that result from combining components. Two varieties of integration testing exist:

FUNCTIONAL TESTING

Functional tests offer methodical proof that the functions being tested are available in accordance with the technical and business requirements, system documentation, and user manuals. Focus of functional testing is on the following areas:

Valid Input: Recognized valid input classes need to be accepted.

Invalid Input: Defined categories of invalid input need to be rejected.

Functions: It is necessary to use the listed functions.

Output: Specific classes of application outputs need to be tested.

Functional tests are organised and prepared with a focus on requirements, important functions, or unique test cases. Additionally, testing must take into account systematic coverage of data fields, established procedures, and subsequent processes as well as business process flows. Additional tests are found, and the usefulness of the existing tests is assessed, before functional testing is finished.

SYSTEM TESTING

System testing makes ensuring that the integrated software system as a whole complies with specifications. In order to provide known and predictable outcomes, it tests a setup. The configuration oriented system integration test is an illustration of system testing. System testing is based on process flows and descriptions, with a focus on pre-driven integration points and links.

BLACK BOX TESTING

Testing software in a "black box" is doing so without having any knowledge of the inner workings, architecture, or language of the module being tested. Black box tests, like the majority of other types of tests, must be created from a clear source document, such as a specification or requirements document. It is a type of testing where the software being tested is handled like a black box. It is impossible to "look" inside. Without taking into account how the software functions, the test generates inputs and responds to outputs.

WHITE BOX TESTING

White box testing is a type of testing where the software tester is familiar with the inner workings, structure, and language of the software, or at the at least, knows what it is intended to do. It has a goal. It is employed to test regions that are inaccessible from a black box level.

ACCEPTANCE TESTING

Before the system is approved for operational usage, this is the last stage of testing. Instead of using synthetic data, the system is tested with data provided by the system procedure. Acceptance by users. Any project's testing phase is crucial and necessitates the end user's active involvement. Additionally, it makes sure the system satisfies the functional specifications.

6.2 Test Cases

Test Case 1:

Test Case#	TC02
Test Name	Wrong dataset entry
Test Description	To test whether the program is running after entering wrong dataset
Input	Incorrect dataset is entered
Expected Output	Error message should me shown
Actual Output	Error message was shown
Test Result	Success

Test Case 2:

Test Case#	TC01
Test Name	Dataset Entry
Test Description	To test If the program runs without Dataset
Input	No dataset is entered
Expected Output	Error message should be shown
Actual Output	Error Message was shown
Test Result	Success

Test Case 3:

Test Case#	TC03
Test Name	Correct dataset entry
Test Description	To test whether the program is running after entering correct dataset
Input	Correct dataset is entered
Expected Output	Program runs with correct analysis
Actual Output	Program runs with correct analysis
Test Result	Success

CHAPTER 7

Conclusion

The Online Cab Trip Analysis project aimed to explore and analyze a dataset containing attributes such as start date, end date, start location, stop location, miles traveled, and purpose of cab trips. Through a comprehensive analysis of the dataset, several key findings were discovered, providing valuable insights into the cab trip patterns and characteristics.

Firstly, it was observed that business cabs were not only used more frequently but also travelled longer distances compared to other categories. This suggests a higher demand for business-related travel and potential opportunities for targeting this segment.

The analysis revealed a seasonal pattern in cab usage, with round trips being more prevalent in December. This finding suggests that December could be a profitable month for cab services by potentially raising fares to meet the increased demand.

Moreover, the analysis highlighted that the majority of cab rides were within a distance of 35 miles, taking approximately 30 minutes. Understanding these average trip durations and distances can help in optimizing cab operations, estimating travel times, and managing resources effectively.

Additionally, the analysis indicated that certain cities had higher cab traffic compared to others, with just a few cities accounting for a significant portion of the rides. This insight could aid in resource allocation and focusing marketing efforts on areas with higher demand.

The study also examined the purpose of cab trips, revealing common reasons such as meetings, meals, and customer visits. Understanding the distribution of trips by purpose can assist in tailoring services and promotions to specific customer needs.

Overall, the Online Cab Trip Analysis project provided valuable insights into cab trip patterns, customer preferences, and areas of opportunity. The findings can guide decision-making processes for cab service providers, enabling them to optimize operations, enhance customer experiences, and maximize profitability.

References

[1] Alemu, E., Girma, D., & Sisay, A. (2019). Analysis of the Online Cab Hailing Service Using K-Means Clustering Algorithm. International Journal of Computer Science and Information Security, 17(8), 33-41.

[2] Chatterjee, S., Banerjee, A., & Misra, S. (2018). Analysis of the Impact of Ratings on Cab Booking Services. In 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE.

[3] Goyal, M., & Singh, R. (2019). Analysis of Demand and Supply in Ride-Sharing Applications. In 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-5). IEEE.

[4] Kumar, R., Sachdeva, R., & Sohi, B. S. (2020). Analysis of Travel Time in Online Cab Services Using Machine Learning Techniques. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE.

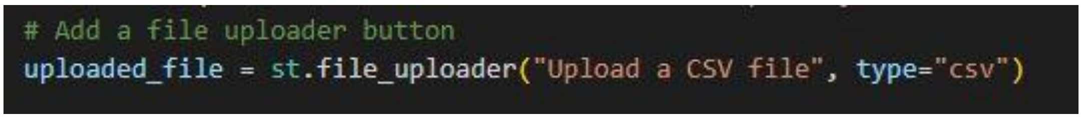
[5] Rani, N., Gupta, D., & Kumar, S. (2020). Predictive Analytics for Cab Service Optimization using Machine Learning. In 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence) (pp. 1-5). IEEE.

[6] https://github.com/ssad5678/WebAppTest

[7] https://www.geeksforgeeks.org

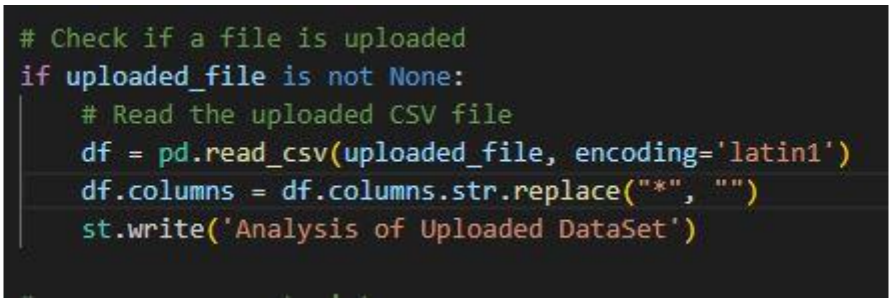
Appendix A: Code Implementation

A1 Uploading of data set



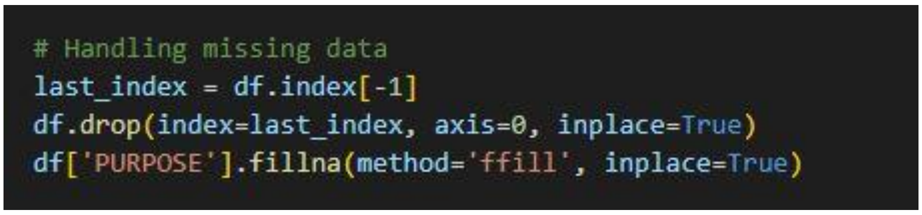
```
# Add a file uploader button
uploaded_file = st.file_uploader("Upload a CSV file", type="csv")
```

A2 Checking if Dataset is Uploaded



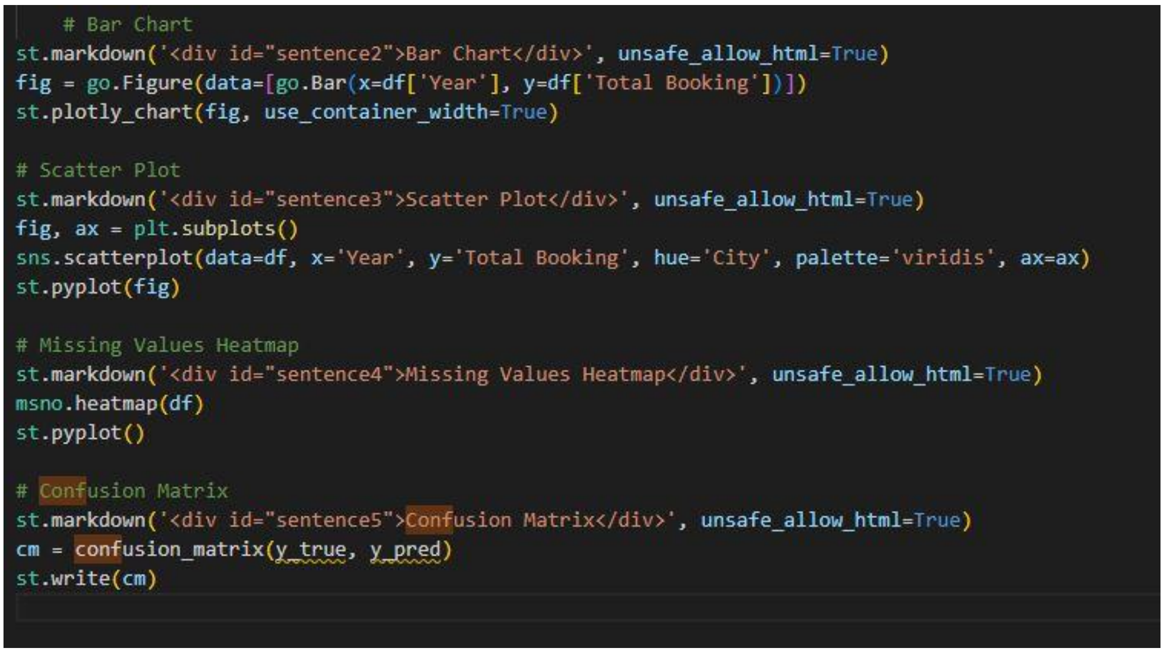
```
# Check if a file is uploaded
if uploaded_file is not None:
    # Read the uploaded CSV file
    df = pd.read_csv(uploaded_file, encoding='latin1')
    df.columns = df.columns.str.replace("*", "")
    st.write('Analysis of Uploaded DataSet')
```

A3 Cleaning of Missing Data



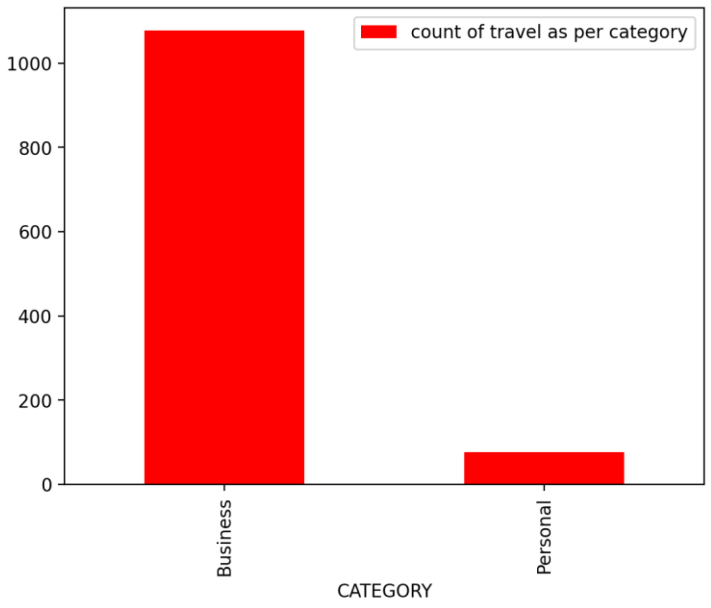
```
# Handling missing data
last_index = df.index[-1]
df.drop(index=last_index, axis=0, inplace=True)
df['PURPOSE'].fillna(method='ffill', inplace=True)
```

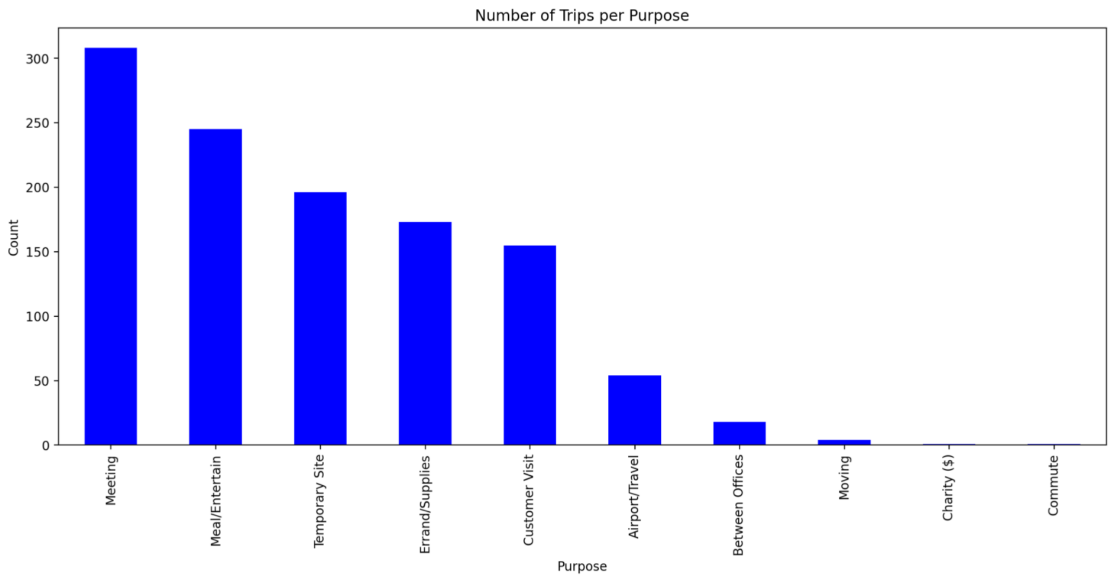
A4 Data Visualization

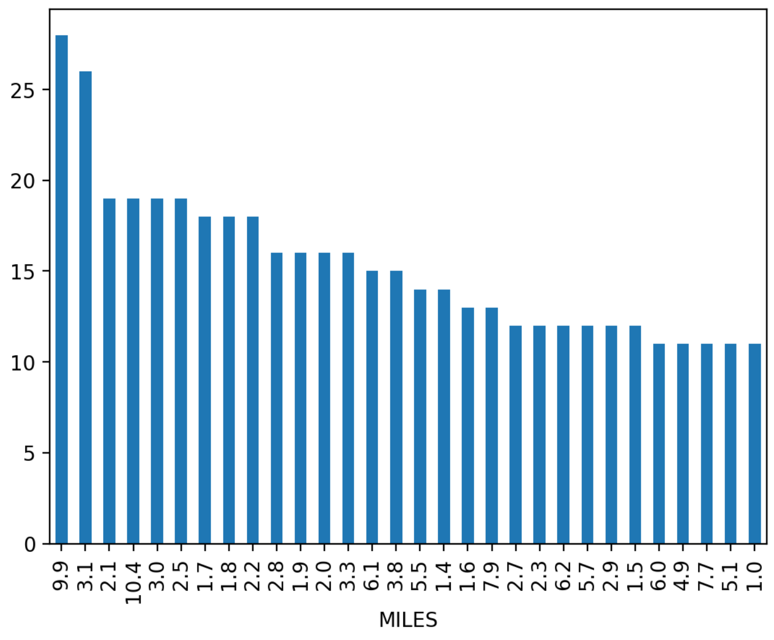


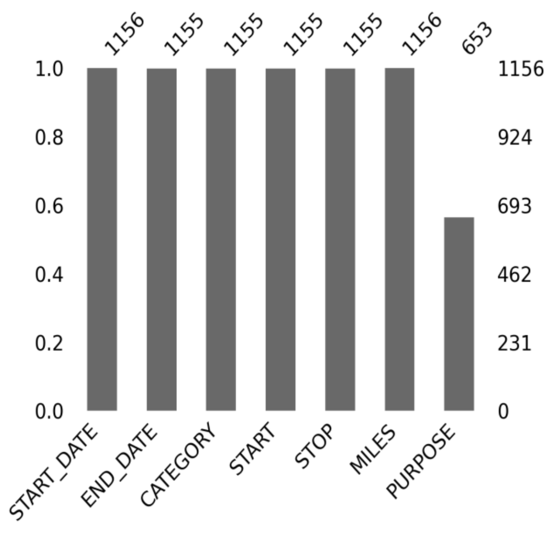
```
    # Bar Chart
st.markdown('<div id="sentence2">Bar Chart</div>', unsafe_allow_html=True)
fig = go.Figure(data=[go.Bar(x=df['Year'], y=df['Total Booking'])])
st.plotly_chart(fig, use_container_width=True)

# Scatter Plot
st.markdown('<div id="sentence3">Scatter Plot</div>', unsafe_allow_html=True)
fig, ax = plt.subplots()
sns.scatterplot(data=df, x='Year', y='Total Booking', hue='City', palette='viridis', ax=ax)
st.pyplot(fig)

# Missing Values Heatmap
st.markdown('<div id="sentence4">Missing Values Heatmap</div>', unsafe_allow_html=True)
msno.heatmap(df)
st.pyplot()

# Confusion Matrix
st.markdown('<div id="sentence5">Confusion Matrix</div>', unsafe_allow_html=True)
cm = confusion_matrix(y_true, y_pred)
st.write(cm)
```

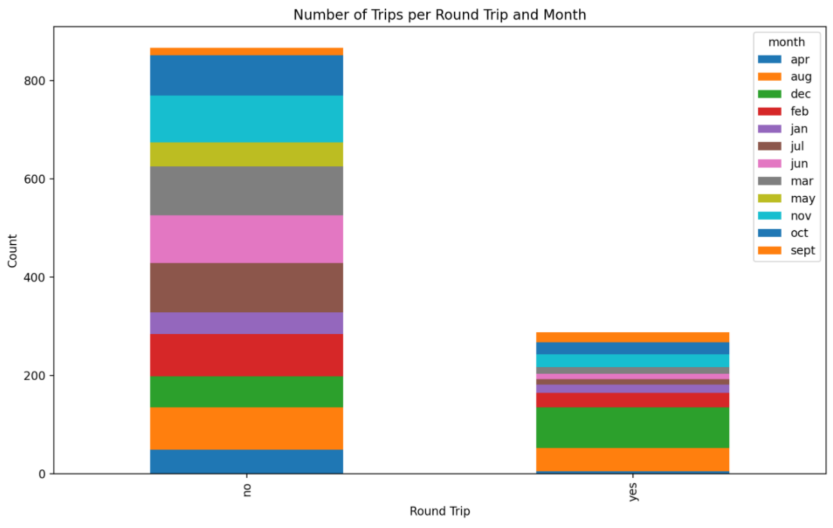
Appendix B: Output

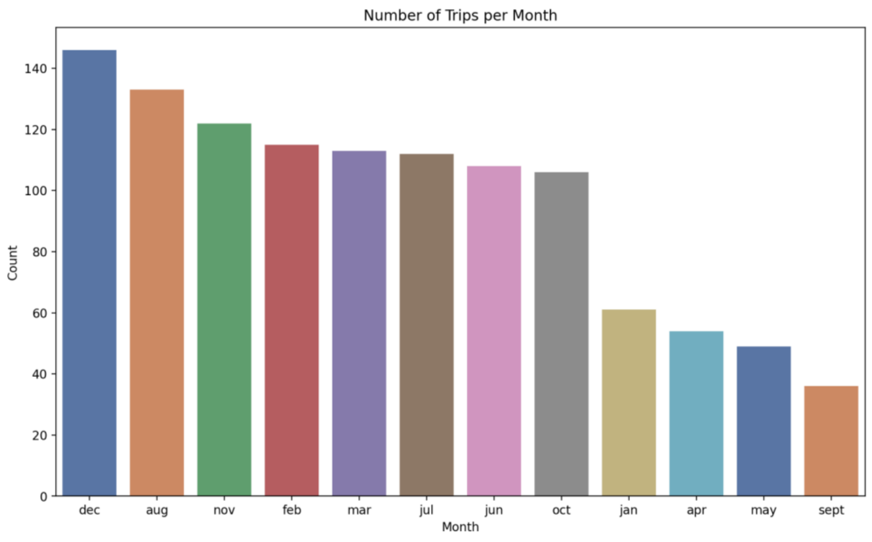


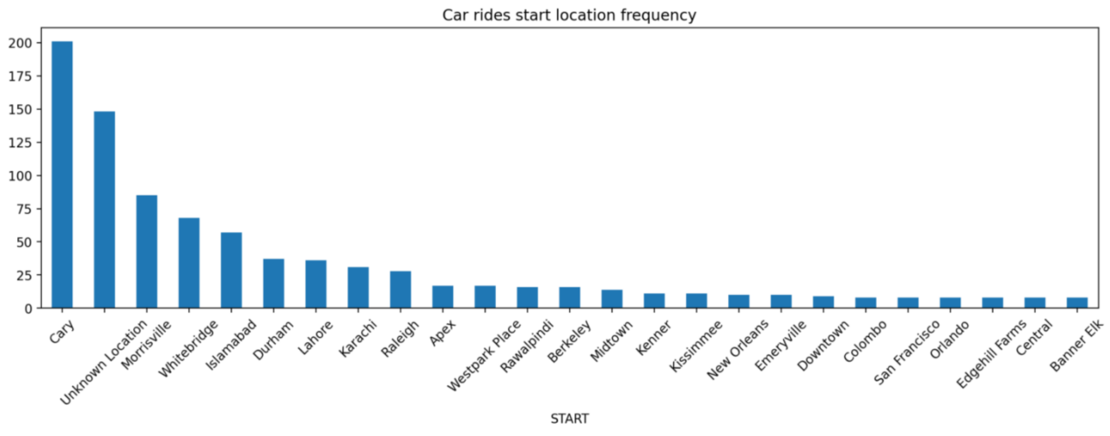


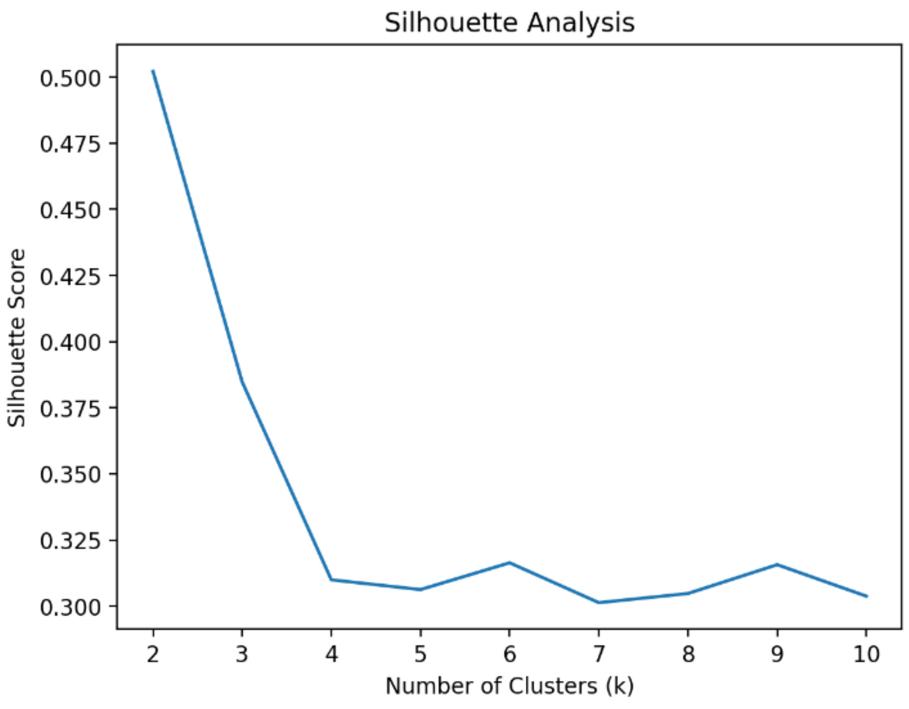

25
20
15
10
5
0
9.9
3.1
2.1
10.4
3.0
2.5
1.7
1.8
2.2
2.8
1.9
2.0
3.3
6.1
3.8
5.5
1.4
1.6
7.9
2.7
2.3
6.2
5.7
2.9
1.5
6.0
4.9
7.7
5.1
1.0
MILES

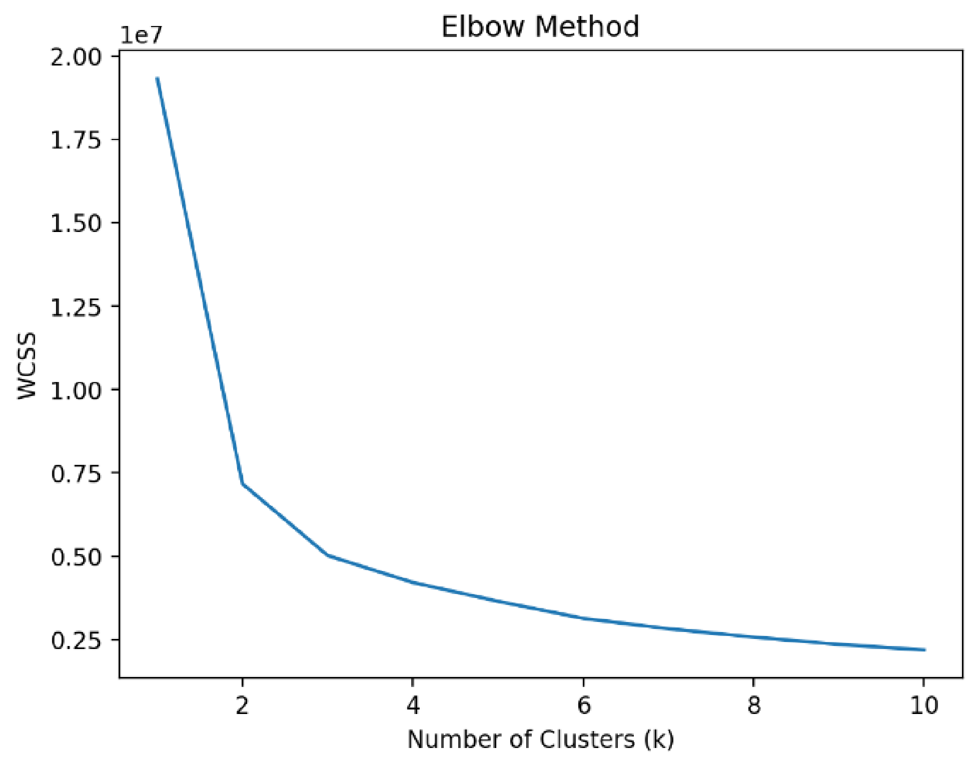

1156
1155
1155
1155
1155
1156
653
1.0
0.8
0.6
0.4
0.2
0.0
1156
924
693
462
231
0
START_DATE
END_DATE
CATEGORY
START
STOP
MILES
PURPOSE

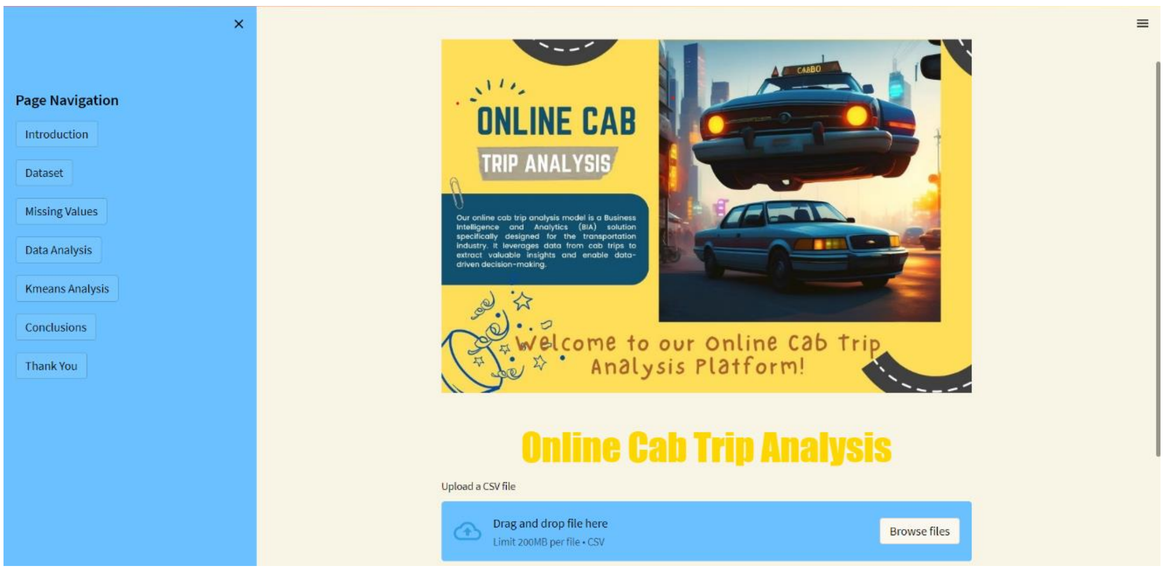

Number of Trips per Round Trip and Month
month
apr
aug
dec
feb
jan
jul
jun
mar
may
nov
oct
sept
Count
800
600
400
200
0
no
yes
Round Trip


Number of Trips per Month
Count
Month
dec
aug
nov
feb
mar
jul
jun
oct
jan
apr
may
sept


Car rides start location frequency
START
Cary
Unknown Location
Morrisville
Whitebridge
Islamabad
Durham
Lahore
Karachi
Raleigh
Apex
Westpark Place
Rawalpindi
Berkeley
Midtown
Kenner
Kissimmee
New Orleans
Emeryville
Downtown
Colombo
San Francisco
Orlando
Edgehill Farms
Central
Banner Elk


Silhouette Analysis
Silhouette Score
Number of Clusters (k)





Our online cab trip analysis model is a Business Intelligence and Analytics (BIA) solution specifically designed for the transportation industry. It leverages data from cab trips to extract valuable insights and enable data-driven decision-making.