MATH6185 OPERATIONAL RESEARCH AND DATA SCIENCE CASE STUDY 1

Predicting London Bike Usage Using Machine Learning

Student:
Vyom Khanna
vk1n23@soton.ac.uk
28965736
Msc Data and Decision Analytics
School of Mathematical Sciences

Supervisor 1:
Professor Christine Currie
christine.currie@soton.ac.uk

Supervisor 2:
Professor Xiang Song
xiang.song@port.ac.uk

Contents

Visualisation

1 Introduction

Urban mobility changed in the last couple of years due to the schemes of bike sharing by cities around the world for battling both traffic congestion and problems related to the environment. A vibrant city, London brought forward its cycle hire scheme in 2010 following successful implementations in Paris and Brussels. This was done to help deal with the city's notorious traffic issues, cut down on pollution, and improve healthy living for the citizens and visitors.
The scheme proved to be an instant success and gained over one million trips in the first ten weeks following its launch.
However, success presents its problems, and the most noticeable one has been in dealing with the logistics of demand and supply of bikes. The critical challenge was to balance high-demand locations with the availability of bikes, ensuring that they did not end up in excess in areas with low demand. Solving these challenges called for a detailed understanding of user behaviour and effective predictive models to forecast bike usage patterns. This report discusses the application of various data science methodologies to improve the London bike-sharing management scheme. I am going to analyse user behaviour, perform multivariate and geospatial analysis, and use machine learning in our objective to optimize bike distribution and improve overall user satisfaction. I aim to contribute through this comprehensive study to the efficient and sustainable operation of urban bike-sharing schemes.

2 Data

For Question 1 to analyse urban mobility patterns, station performance, and cycling preferences among London's diverse population data is taken from the open-source TFL website https://cycling.data.tfl.gov.uk/
In Question 2 the aim is to develop a predictive London bike usage model using machine learning techniques and Python software. Data is taken from open-source https://www.visualcrossing.com/weather/weather-data-services/

3 Literature Review

3.1 Data Collection and Preprocessing

Effective prediction models rely on comprehensive, high-quality datasets. The primary data sources include trip records from Transport for London's open data platform, detailing trip duration, start and end locations, and timestamps. Additionally, weather information, which significantly influences bike usage patterns, is incorporated. Combining these datasets provides a robust foundation for predictive modelling (Gu, 2023).

3.2 Machine Learning Techniques

Various machine learning algorithms are employed to predict bike-sharing demand, ranging from traditional regression methods to advanced neural networks.
Regression Models: Multiple linear regression serves as a baseline technique for demand prediction, helping to understand the linear relationships between bike usage and factors like weather, day of the week, and holidays. (Gu, 2023) demonstrated its effectiveness in capturing these dependencies.
Neural Networks: Deep learning models, particularly Long Short-Term Memory (LSTM) networks, excel in capturing temporal dependencies in bike-sharing data. An LSTM model effectively modelled the time series nature of the data and improved prediction accuracy for hourly bike demand in London (MDPI, 2023).

3.3 Feature Engineering and Data Analysis

Feature engineering enhances machine learning model performance by selecting and creating relevant features.

1. Temporal Features: Variables such as the hour of the day, day of the week, and season impact bike usage patterns. Creating features reflecting these aspects helps capture periodic trends (ScienceDirect, 2023).
2. Weather Features: Conditions like temperature, precipitation, and wind speed are vital predictors of bike usage. Integrating historical weather data with trip data enhances model accuracy (VisualCrossing, 2023).
3. Station-Specific Features: Characteristics of bike stations, including their location and proximity to landmarks, influence demand patterns, aiding in understanding spatial dependencies (ScienceDirect, 2023).

3.4 Model Evaluation and Optimisation

Evaluating and optimising machine learning models is crucial for reliable predictions. Common evaluation metrics include Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared values. Techniques such as cross-validation and grid search for hyperparameter tuning are commonly employed.

(Feng, 2017) demonstrated that ensemble methods like random forest and GBDT, combined with thorough feature engineering, result in lower RMSE and higher R-squared values compared to traditional regression models.

4 Analysing Urban Mobility Patterns and Cycling Preferences in London's Diverse Population

4.1 Data Collection and Preparation

My student ID is 28965736, which ends with the digit 6. Consequently, I downloaded the data for the month of June for my analysis. The data for this study was sourced from Transport for London's (TfL) open data portal, which provides extensive records of bike hires across the city. The datasets used for Exploratory Data Analytics are:

- 372JourneyDataExtract29May2023-04Jun2023.csv
- 374JourneyDataExtract12Jun2023-18Jun2023.csv
- 373JourneyDataExtract05Jun2023-11Jun2023.csv
- 375JourneyDataExtract19Jun2023-30Jun2023.csv

4.2 Data Processing

The Python library Pandas was imported for data manipulation and analysis. Using ***pd.read_csv()***, all four CSV files were loaded into separate Dataframes. These DataFrames were then combined into a single DataFrame using ***pd.concat()*** with the parameter ***ignore_index=True***, ensuring that all rows are consolidated under their respective columns.

```
df1 = pd.read_csv('372JourneyDataExtract29May2023-04Jun2023.csv')
df2 = pd.read_csv('374JourneyDataExtract12Jun2023-18Jun2023.csv')
df3 = pd.read_csv('373JourneyDataExtract05Jun2023-11Jun2023.csv')
df4 = pd.read_csv('375JourneyDataExtract19Jun2023-30Jun2023.csv')
df = pd.concat([df1, df2, df3, df4], ignore_index=True)
```

4.3 Data Cleaning

The downloaded dataset included entries for May 29th, 30th, and 31st. However, since the focus of this analysis is the month of June, I filtered out these. To achieve this, I first converted the 'Start Date' column to a datetime format using ***pd.to_datetime()***. Then, I filtered the data to include only entries where the month was June by using ***dt.month == 6*** on the new start date column. Upon reviewing the filtered dataset, I found no blank columns or empty cells. However, I identified a few entries in the 'Total Duration (ms)' column where the duration was less than one minute or more than one day. These were deemed outliers and were removed from the dataset. 1 minute is equivalent to 60,000 milliseconds and 1 day is equivalent to 86,400,000 milliseconds. The resulting DataFrame, ***june_df***, is the dataset used for further analysis.

```
df['Start date'] =pd.to_datetime(df['Start date'], format='%Y-%m-%d %H:%M')
june_df =  df[df['Start date'].dt.month == 6] june_df = june_df[(june_df['Total duration
(ms)'] >= 60000) & (june_df['Total duration (ms)'] <= 8640000
```

4.4 Adding Column

I performed several feature engineering steps to enhance our dataset and facilitate more detailed analysis. These steps included adding new columns to the june_df and transforming existing data into more useful formats. Below are the detailed steps and the corresponding Python code used.

4.4.1 Adding a Column for Day of the Week

I am creating a new column, *day_of_the_week*, to indicate the day of the week for each trip. This was achieved using the ***dt.day_name()*** method on the Start date column.

```
june_df['day_of_the_week'] = june_df['Start date'].dt.day_name()
```

4.4.2 Formatting Start Date

A new column, *Start date temp*, was added to store the start date in YYYY-MM-DD format. This was done using the ***dt.date()*** method.

```
june_df['Start date temp'] = june_df['Start date'].dt.date
```

4.4.3 Time of Day Classification

To categorise trips into different times of the day, I defined a function, *time_of_day*, which classifies the hour of the trip start time into Morning, Afternoon, Evening, or Night. This function was then applied to the Start date column to create the *time_of_day* column.

```
def time_of_day(Time):
     if 6 <= Time < 12:
 return 'Morning'
     elif 12 <= Time < 18:
 return 'Afternoon'
     elif 18 <= Time < 23:
 return 'Evening'
     else:
 return 'Night'
june_df['time_of_day'] = june_df['Start date'].dt.hour.apply(time_of_day)
```

4.4.4 Creating a Route Column

I combined the *Start station* and *End station* columns to form a new column named Route, which indicates the journey path.

```
june_df['Route'] = june_df['Start station'] + ' - ' + june_df['End station']
```

4.4.5 Extracting Hour from Start Date

A new column, *hour*, was created to extract the hour component from the Start date column using ***dt.hour***.

```
june_df['hour'] = june_df['Start date'].dt.hour
```

4.4.6 Time Interval Buckets

I categorised trip durations into time intervals using the ***pd.cut()*** method. This involved creating a new column, Time Interval, which classifies the duration of each trip into predefined buckets.

```
ms = [0, 5*60*1000, 15*60*1000, 30*60*1000, 60*60*1000, 24*60*60*1000]
intervals = ['0-5 Min', '5-15 Min', '15-30 Min', '30-60 Min', '60 Min - 1 Day']
june_df['Time Interval'] = pd.cut(june_df['Total duration (ms)'], bins=ms, labels=intervals)
```

4.4.7 Week Number Classification

To facilitate weekly analysis, I added a new column, *Week_Number*, which classifies the trips based on the week of the month they occurred. This was done using a custom function, *Week_Number*, which assigns week numbers based on the trip date.

```
def Week_Number(date):
if date >= pd.Timestamp('2023-06-01') and date < pd.Timestamp('2023-06-08'):
    return 'Week 1'
elif date >= pd.Timestamp('2023-06-08') and date < pd.Timestamp('2023-06-15'):
    return 'Week 2'
elif date >= pd.Timestamp('2023-06-15') and date < pd.Timestamp('2023-06-22'):
    return 'Week 3'
else:
     return 'Week 4'
june_df['Week_Number'] = june_df['Start date'].apply(Week_Number)
```

4.4.8 Dropping Unwanted DataFrames

Finally, I removed any unwanted DataFrames from the workspace to clean up the environment.

```
del df, df1, df2, df3, df4
```

4.5 Insights and Visualizations

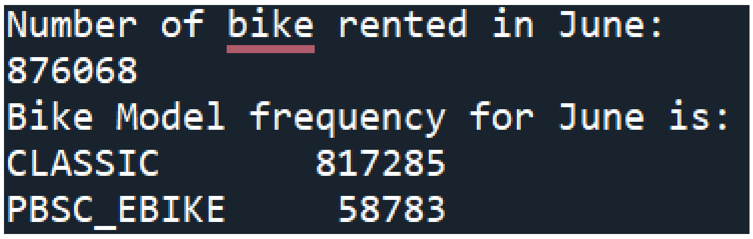
This section presents the insights and visualisations generated from the analysis of bike rental data. Various types of charts, including pie charts, bar charts, and scatter plots, were used to illustrate different aspects of the data.

4.5.1 Bike Model Frequency

To understand the distribution of bike models rented, I calculated the frequency of each bike model using ***value_counts()***.

```
rental_frequency = june_df['Bike model'].value_counts()
```

bike_frequency = june_df['Bike model'].value_counts().sum()



```
Number of bike rented in June:
876068
Bike Model frequency for June is:
CLASSIC       817285
PBSC_EBIKE     58783
```

Figure 1: Bike Model Frequency and Number of Bikes Rented in June Output

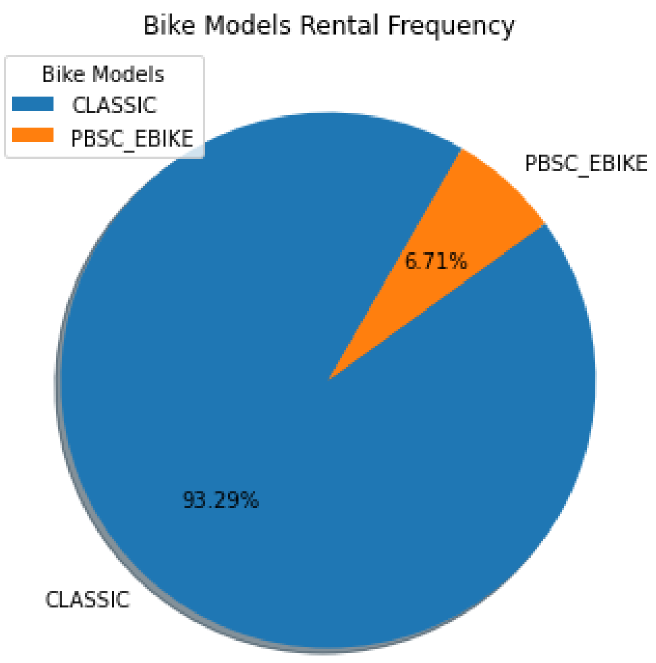
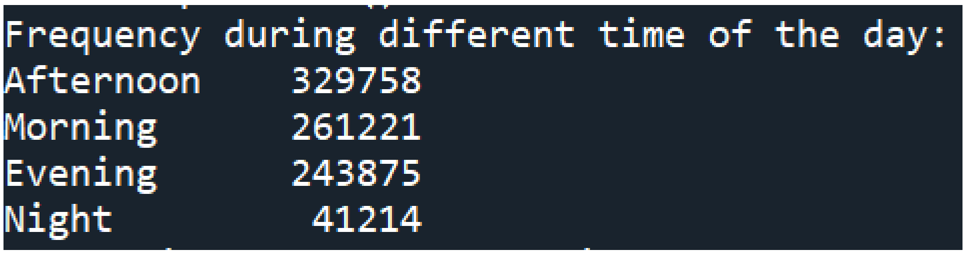


Figure 2: Bike Model Frequency Pie Chart

4.5.2 Time of the Day Frequency

I analysed the distribution of bike rentals across different times of the day using ***value_counts()***.

time_route = june_df['time_of_day'].value_counts()



```
Frequency during different time of the day:
Afternoon    329758
Morning      261221
Evening      243875
Night         41214
```

Figure 3: Bike model frequency during different times of the day in June Output

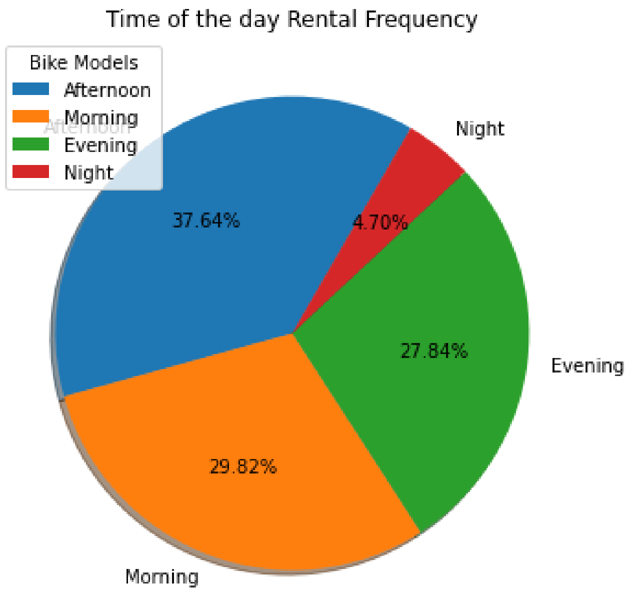


Figure 4: Bike model frequency during different times of the day in the June Pie Chart

4.5.3 Number of Trips Every Day

To understand the daily rental patterns, I calculated the number of trips per day using ***groupby()*** and ***size().***

trips_per_day = june_df.groupby('Start date temp').size()

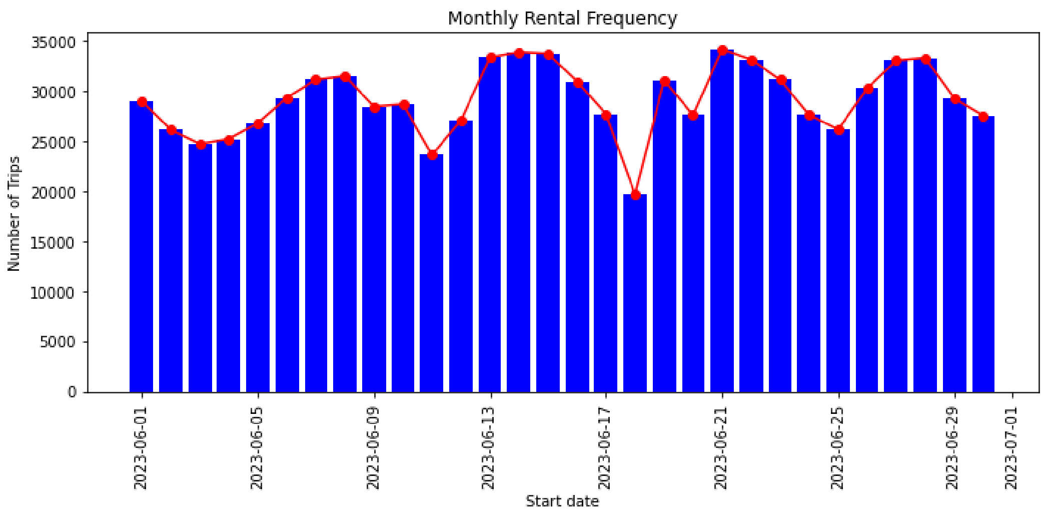
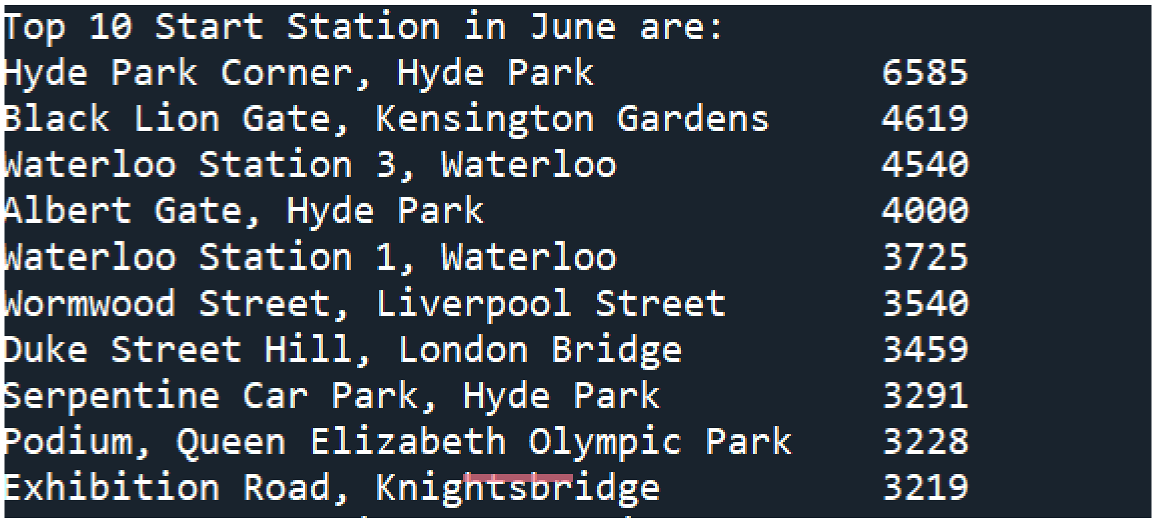


Figure 5: Daily Bike Rental Frequency for June 2023

4.5.4 Frequent Start Stations

I identified the top 10 start stations using ***value_counts()*** and ***head(10)***



```
Top 10 Start Station in June are:
Hyde Park Corner, Hyde Park                 6585
Black Lion Gate, Kensington Gardens         4619
Waterloo Station 3, Waterloo                4540
Albert Gate, Hyde Park                      4000
Waterloo Station 1, Waterloo                3725
Wormwood Street, Liverpool Street           3540
Duke Street Hill, London Bridge             3459
Serpentine Car Park, Hyde Park              3291
Podium, Queen Elizabeth Olympic Park        3228
Exhibition Road, Knightsbridge              3219
```

Figure 6: Top 10 Bike Rental Start Stations in June 2023 Values

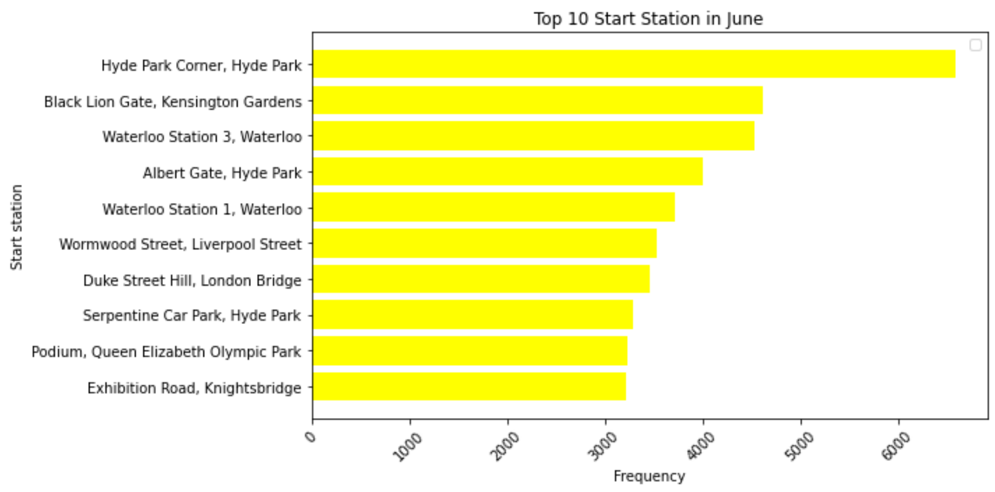
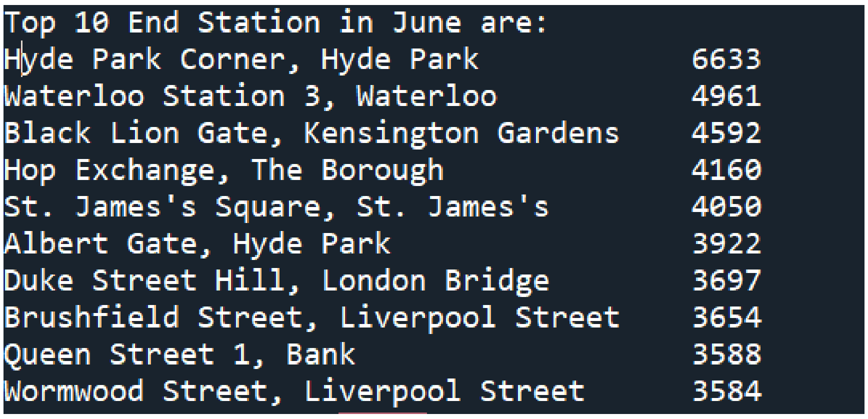


Figure 7: Top 10 Bike Rental Start Stations in June 2023 Horizontal bar graph

4.5.5 Frequent End Stations

I identified the top 10 end stations using ***value_counts()*** and ***head(10)***



```
Top 10 End Station in June are:
Hyde Park Corner, Hyde Park               6633
Waterloo Station 3, Waterloo              4961
Black Lion Gate, Kensington Gardens       4592
Hop Exchange, The Borough                 4160
St. James's Square, St. James's           4050
Albert Gate, Hyde Park                    3922
Duke Street Hill, London Bridge           3697
Brushfield Street, Liverpool Street       3654
Queen Street 1, Bank                      3588
Wormwood Street, Liverpool Street         3584
```

Figure 8: Top 10 Bike Rental End Stations in June 2023 Values

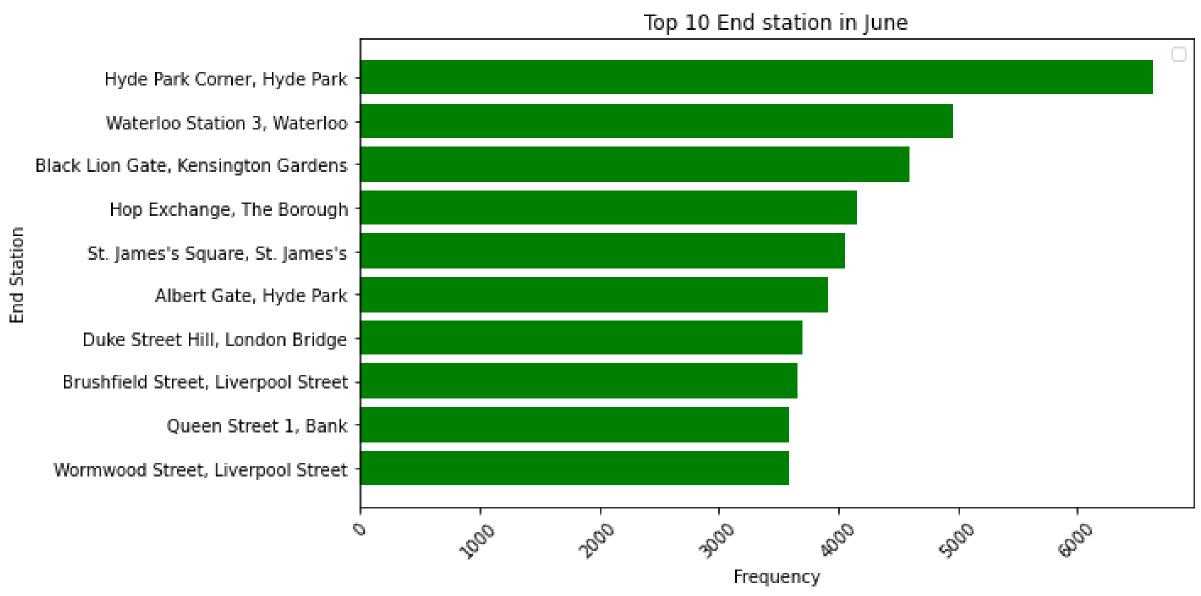


Figure 9: Top 10 Bike Rental End Stations in June 2023 Horizontal bar graph

4.5.6 Frequent Routes

To identify the most popular routes, I calculated the top 10 routes using ***value_counts()*** and ***head(10)***

I used the *textwrap* library to make the route names more readable and created a horizontal bar chart.

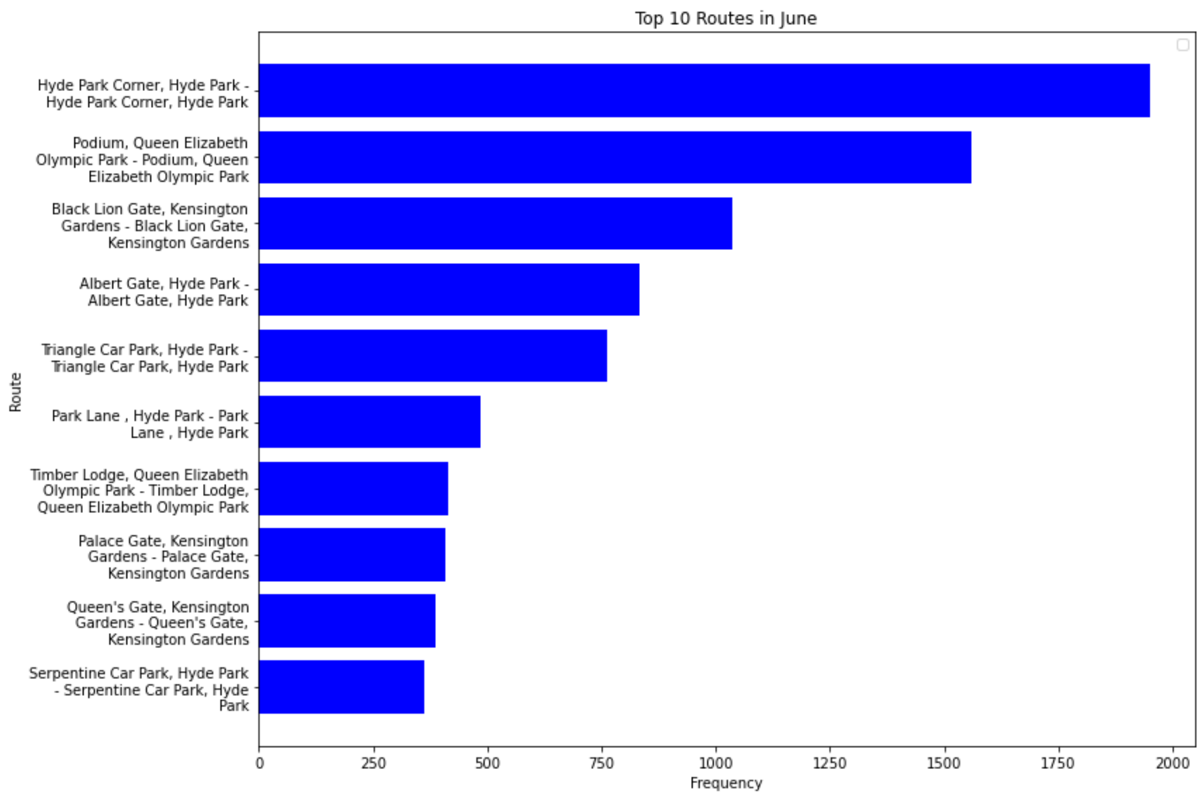


Figure 10: Top 10 Bike Routes in June 2023 Horizontal bar graph

4.5.7 Hourly Rentals

I analysed the distribution of rentals by hour for the entire month. A scatter plot was created to show the number of rentals per hour.

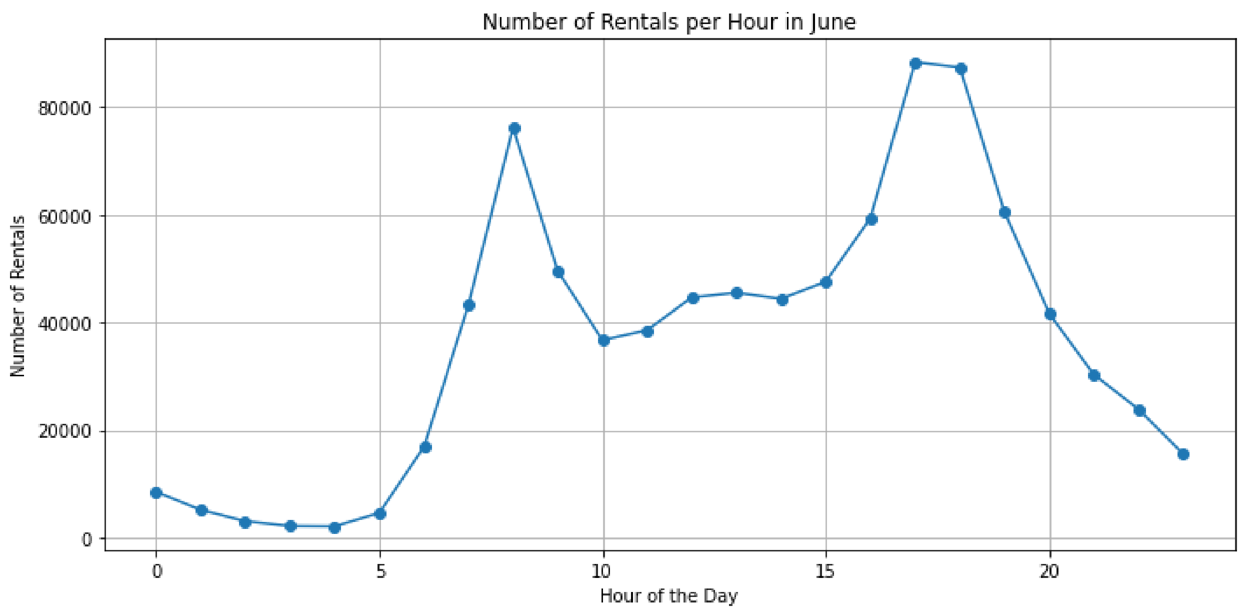


Figure 11: Hourly Bike Rental Frequency in June 2023

4.5.8 Frequent Bike Numbers

I identified the top 10 bike numbers used in June.

```
june_df['Bike number'] = june_df['Bike number'].astype(str)
frequent_bike = june_df['Bike number'].value_counts().head(10)
```

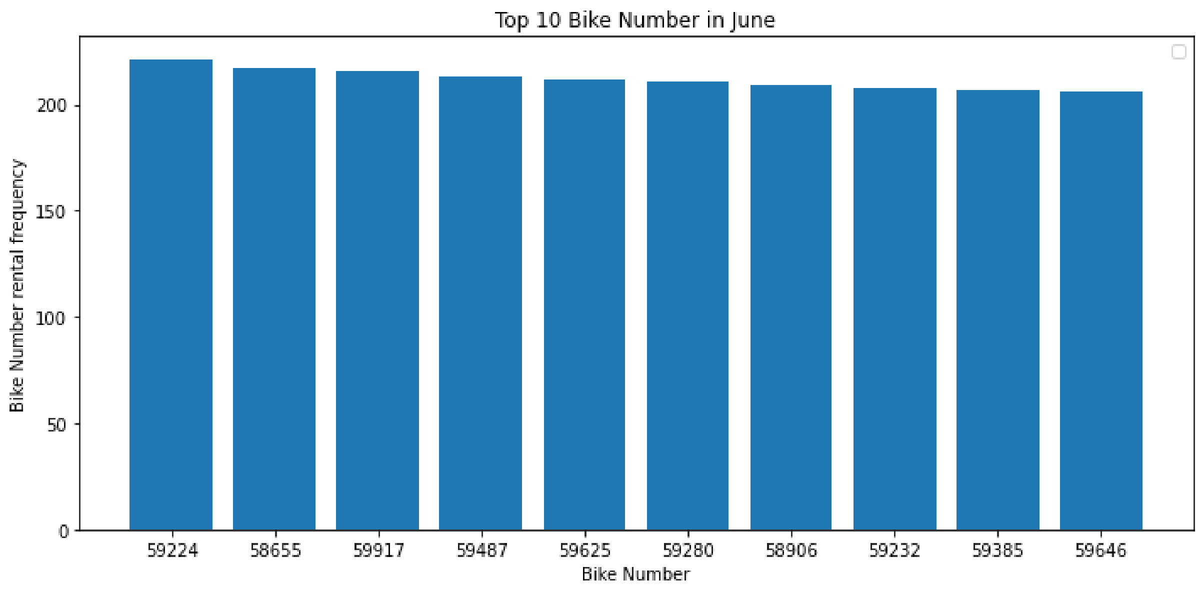


Figure 12: Top 10 Bike Number rental frequency in June 2023

4.5.9 Week-wise Insights

I performed a detailed analysis for each week in June, including metrics like total and mean duration, bike model frequency, day rental frequency, top 10 routes, start and end stations, and hourly rentals. Below is an example for Week 1.

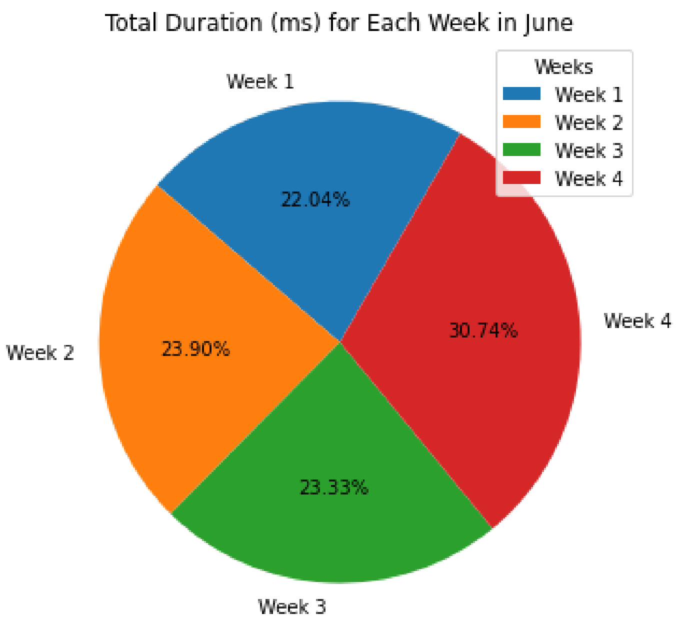


Figure 13: Total Bike Rental Duration for Each Week in June 2023

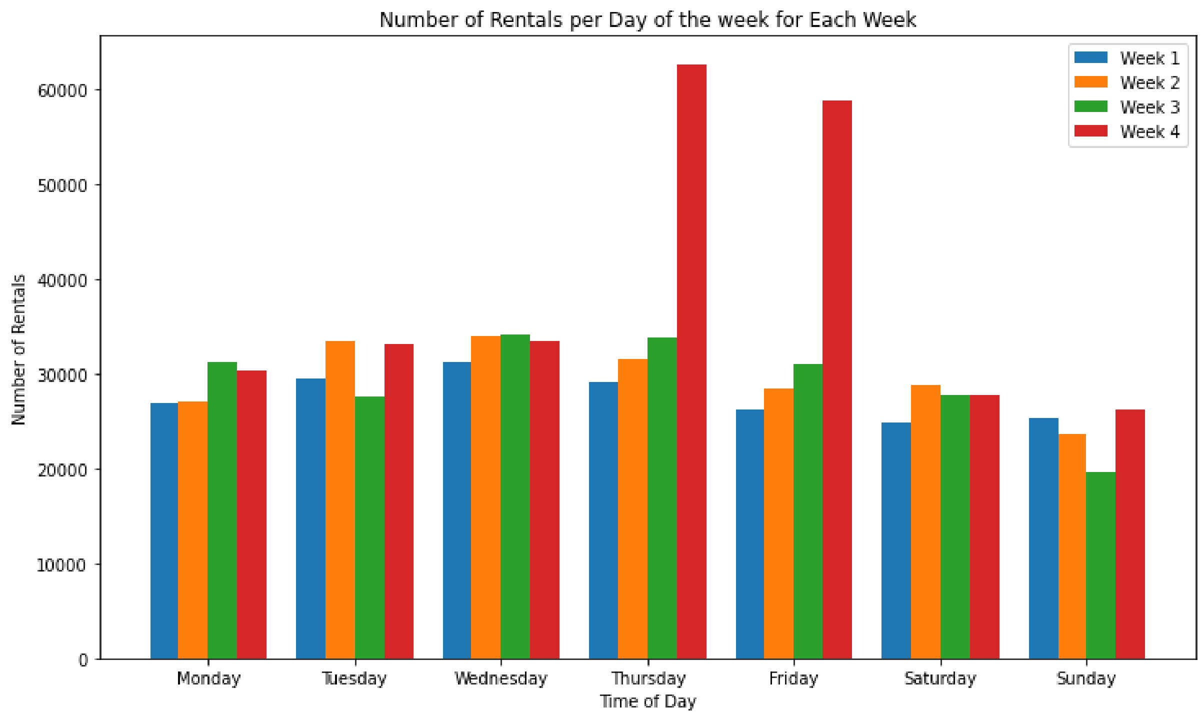


Figure 14: Bike rental frequency of each day of each week

4.5.10 Correlation Matrix

Finally, I created a correlation matrix to explore relationships between different variables in the dataset.

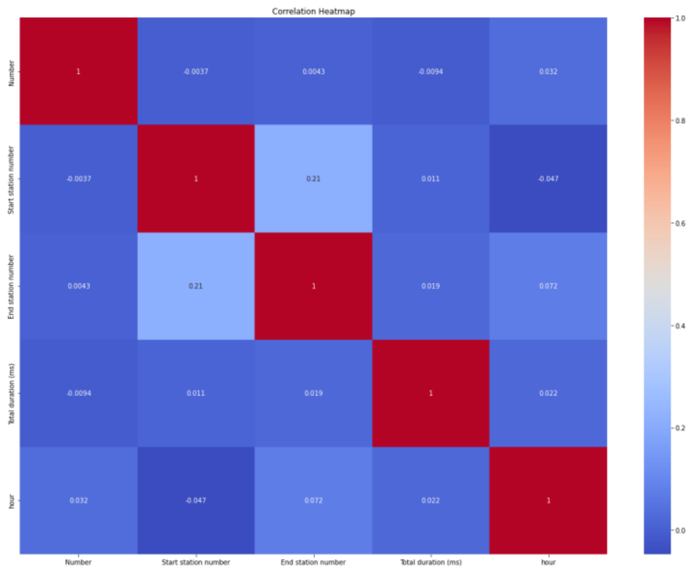


Figure 15: Correlation Heatmap of Bike Rental Data for June 2023

4.5.11 Number of rentals each per time of the day

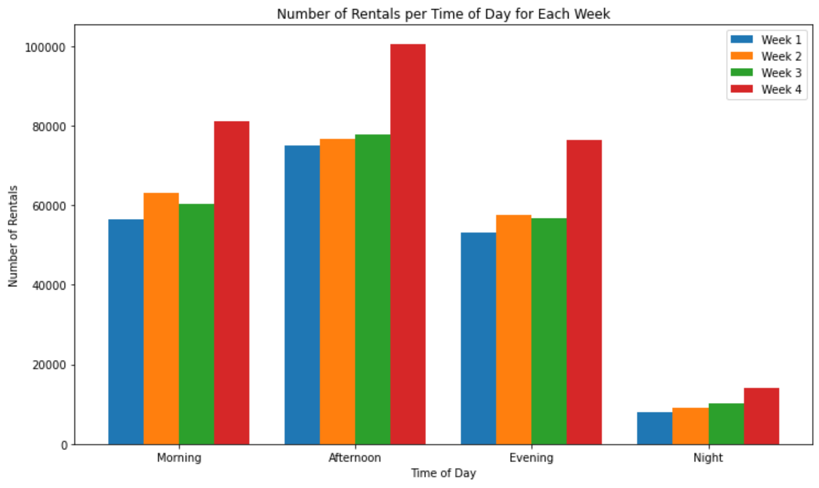


Figure 16: Bike rental frequency of each day during different times of the day

4.5.12 Further Analysis

Further Exploratory EDA Could be done by bifurcating data on the basis of day of week, week and time of the day. Example

Morning_Week1 = june_df.query('time_of_day == "Morning" and Week_Number == "Week 1"')

Morning_Week1_Total = Morning_Week1['Total duration (ms)'].sum() / 3600000

Morning_Week1_rental_frequency = Morning_Week1['Bike model'].value_counts()

Morning_Week1_Route = Morning_Week1['Route'].value_counts().head(10)

5 Predictive Models for London Bike Usage Using Machine Learning in Python

5.1 Introduction

This report outlines the process of predicting bike usage patterns using machine learning techniques. The analysis utilises weather data collected from 1st July 2023 to 30th September 2023. The objective is to identify and implement effective models that can predict bike usage based on various weather parameters.

5.2 Data Collection and Procession

To facilitate data handling, visualisation, numerical operations, and machine learning tasks, the following Python libraries were imported:
pandas: For data manipulation and analysis.
seaborn and matplotlib.pyplot: For creating various types of charts and visualisations.
numpy: For numerical operations.
scikit-learn: For implementing machine learning models.

5.3 Loading and Preparing Data

The dataset, containing weather data from 1st July 2023 to 30th September 2023 was downloaded and then was loaded using ***pd.read_csv()***. The datetime column was converted to a more manageable format using ***pd.to_datetime()***.

```
weather_df = pd.read_csv('London 2023-07-01 to 2023-09-30.csv')
weather_df['datetime'] = pd.to_datetime(weather_df['datetime'])
```

5.4 Feature Extraction

Additional columns were extracted from the datetime column to provide more information and to do further analysis:
Day name (day): e.g., Monday, Tuesday.
Month number (Month): e.g., 7 for July.
Day of the week (day_of_week): with Monday as 0 and Sunday as 6.
Weekend indicator (weekend): 1 if the day is a weekend, 0 otherwise.
Week number (week).

```
weather_df['day'] = weather_df['datetime'].dt.day_name()
weather_df['Month'] = weather_df['datetime'].dt.month
weather_df['day_of_week'] = weather_df['datetime'].dt.dayofweek
weather_df['weekend'] = weather_df['day_of_week'].isin([5, 6]).astype(int)
weather_df['week'] = weather_df['datetime'].dt.isocalendar().week
```

5.4.1 Modified columns

Sunrise and sunset times were also converted to a simpler format.

```
weather_df['sunrise'] = pd.to_datetime(weather_df['sunrise']).dt.time
weather_df['sunset'] = pd.to_datetime(weather_df['sunset']).dt.time
```

5.5 Data Cleaning

The columns which were blank or the mean was 0 were dropped for precise analysis

```
weather_df = weather_df.drop(columns=['severerisk', 'snow', 'snowdepth', 'stations'])
```

5.6 Data Transformation

To prepare categorical data for analysis, one-hot encoding was applied using ***pd.get_dummies()***.

```
weather_df = pd.get_dummies(weather_df, columns=['day', 'Month', 'preciptype'], drop_first=True)
```

5.7 Feature Engineering

5.7.1 Feature Selection

A list of features was created to include specific pieces of information relevant for further analysis.

```
features = [
'tempmax', 'tempmin', 'temp', 'feelslikemax', 'feelslikemin', 'feelslike', 'dew', 'humidity', 'precip', 'precipprob', 'precipcover', 'windspeed', 'winddir', 'cloudcover', 'solarradiation', 'solarenergy', 'uvindex', 'day_of_week', 'weekend'
]
final_data = weather_df[features]
```

5.7.2 Weekly Aggregations

Weekly aggregations were computed for summarising data, and calculating the sum and mean for various metrics.

```
weekly_agg = weather_df.groupby('week').agg({
   'tempmax': 'mean',
   'tempmin': 'mean',
   'temp': 'mean',
   'feelslikemax': 'mean',
   'feelslikemin': 'mean',
   'feelslike': 'mean',
   'dew': 'mean',
   'humidity': 'mean',
   'precip': 'sum',
   'precipprob': 'mean',
   'precipcover': 'mean',
   'windspeed': 'mean',
   'winddir': 'mean',
   'cloudcover': 'mean',
   'solarradiation': 'mean',
   'solarenergy': 'mean',
   'uvindex': 'mean'
}).reset_index()
```

5.7.3 Calculation of New Features

New features were engineered from the existing data to enhance the predictive power of the models:

5.7.3.1 *Calculated fields*

temp_feelslike: Product of temperature and feels-like temperature.
humidity_temp: Product of humidity and temperature.
wind_humidity: Product of wind speed and humidity.

```
weather_df['temp_feelslike'] = weather_df['temp'] * weather_df['feelslike']
weather_df['humidity_temp'] = weather_df['humidity'] * weather_df['temp']
weather_df['wind_humidity']=weather_df['windspeed']*weather_df['humidi
ty']
```

5.7.3.2 *Lagged feature*

Lagged features for temperature, humidity, and precipitation were created to capture the impact of previous days' weather conditions on current conditions.

```
for lag in range(1, 4):
   weather_df['temp_lag_' + str(lag)] = weather_df['temp'].shift(lag)
   weather_df['humidity_lag_' + str(lag)] = weather_df['humidity'].shift(lag)
   weather_df['precip_lag_' + str(lag)] = weather_df['precip'].shift(lag)
```

5.7.3.3 *Rolling feature*

Rolling mean and sum features were calculated for temperature, humidity, and precipitation over 7, 14, and 30-day periods to capture longer-term trends.

```
for roll_data in [7, 14, 30]:
weather_df['temp_roll_mean_'+str(roll_data)]=weather_df['temp'].rolling(ro
ll_data).mean()
weather_df['humidity_roll_mean_'+str(roll_data)]=weather_df['humidity'].ro
lling(roll_data).mean()
weather_df['precip_roll_sum_'+str(roll_data)]=weather_df['precip'].rolling(r
oll_data).sum()
```

5.8 Data Integration and Normalization

5.8.1 Combining Features

All features were combined into a single DataFrame, and weekly aggregates were merged back into the original dataset.

```
final_data = weather_df[
features +
['temp_feelslike', 'humidity_temp', 'wind_humidity'] +
['temp_lag_' + str(lag) for lag in range(1, 4)] +
 ['humidity_lag_' + str(lag) for lag in range(1, 4)] +
 ['precip_lag_' + str(lag) for lag in range(1, 4)] +
 ['temp_roll_mean_' + str(roll_data) for roll_data in [7, 14, 30]] +
 ['humidity_roll_mean_' + str(roll_data) for roll_data in [7, 14, 30]] +
 ['precip_roll_sum_' + str(roll_data) for roll_data in [7, 14, 30]]
]
weather_df = weather_df.merge(weekly_agg, on='week', how='left')
```

5.8.2 Normalizing Numerical Features

Numerical features were normalised using *MinMaxScaler*.

```
numerical_features=
final_data.select_dtypes(include=[np.number]).columns.tolist()
scaler = MinMaxScaler()
final_data[numerical_features]
scaler.fit_transform(final_data[numerical_features])
```

5.9 Correlation Matrix

```
plt.figure(figsize=(50, 35))
correlation_matrix = final_data.corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True,
cbar_kws={'shrink': .5})
plt.title("Correlation Heatmap of Engineered Features")
plt.show()
```

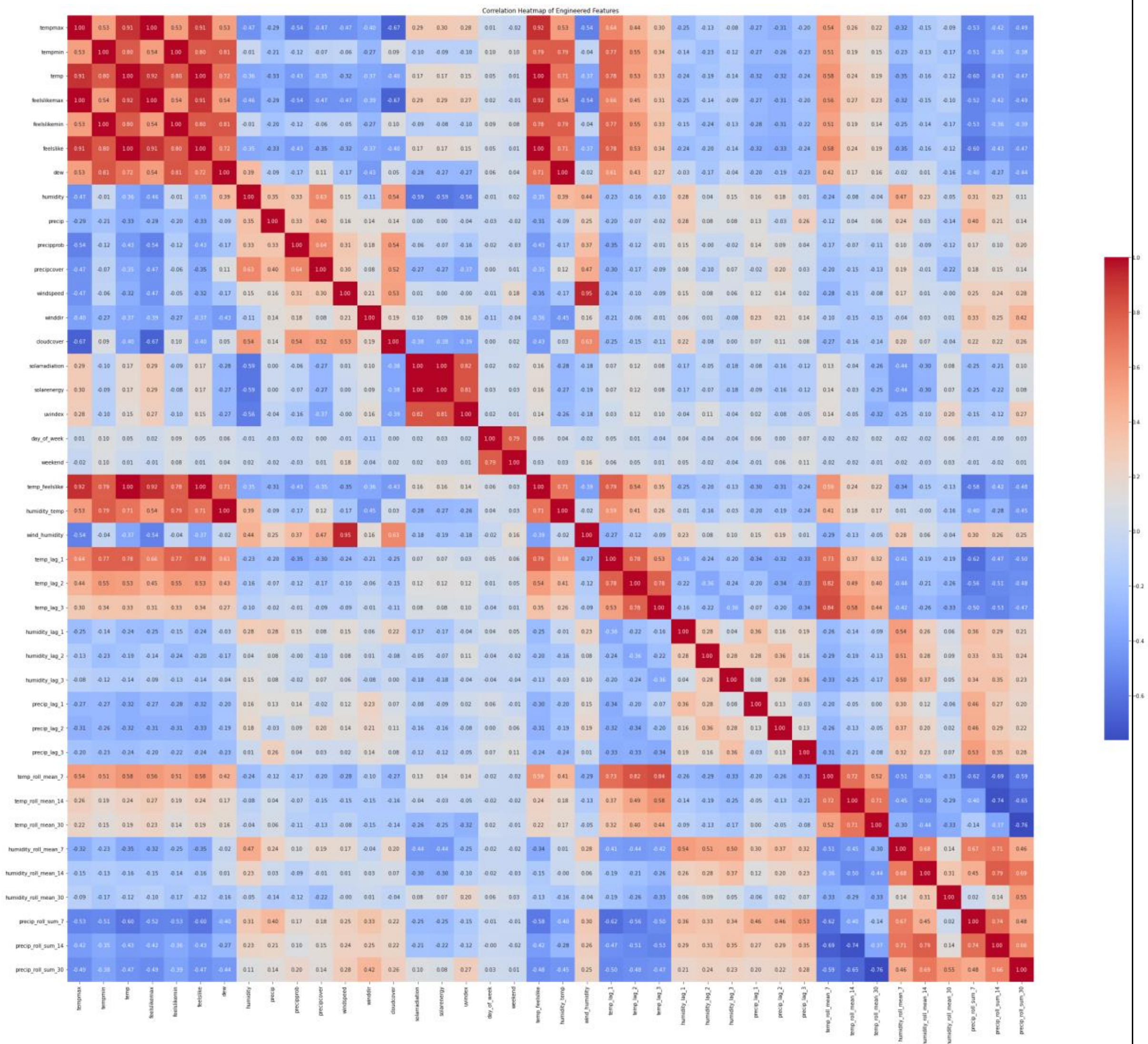


Figure 17: Correlation Heatmap of Engineered Features for June 2023

5.10 Handling Missing Values

Missing values were handled using *SimpleImputer* from *sklearn.impute*, replacing NaNs with the mean of the respective columns.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
final_data=pd.DataFrame(imputer.fit_transform(final_data),
columns=final_data.columns)
```

5.11 Data Splitting and Model Training

5.11.1 Data Splitting

The dataset was split into features (X) and target variable (y), and further into training and testing sets.

```
from sklearn.model_selection import train_test_split
x_temp = final_data.drop(columns=['temp'])  # Features
y_temp = final_data['temp']  # Target variable
x_temp_train, x_temp_test, y_temp_train, y_temp_test= train_test_split(x_temp,
y_temp, test_size=0.25, random_state=40)
```

5.11.2 Model Training and Evaluation

Various machine learning models were trained and evaluated, including Linear Regression, Decision Tree, Random Forest, Neural Network, SVM, KNN, and XGBoost. Model performance was assessed using Mean Squared Error (MSE), R-Squared (R²), and Mean Absolute Error (MAE).

```
from sklearn.metrics import mean_squared_error, r2_score
,mean_absolute_error
models = {
'Linear Regression': LinearRegression(),
'Decision Tree': DecisionTreeRegressor(random_state=48),
'Random Forest': RandomForestRegressor(random_state=48),
 'Neural Network': MLPRegressor(random_state=48, max_iter=100),
 'SVM': SVR(kernel='rbf'),
 'KNN': KNeighborsRegressor(n_neighbors=5),
 'XGBoost': xgb.XGBRegressor(objective='reg:squarederror', random_state=48)
    }
    results = {}
    for name, model in models.items():
      model.fit(x_temp_train, y_temp_train)
      y_temp_pred = model.predict(x_temp_test)
      mse = mean_squared_error(y_temp_test, y_temp_pred)
      r2 = r2_score(y_temp_test, y_temp_pred)
      mae = mean_absolute_error(y_temp_test, y_temp_pred)
      results[name] = {'MSE': mse, 'R2': r2, 'MAE': mae}
    for name, metrics in results.items():
      print(f"{name} - MSE: {metrics['MSE']:.4f}, R2: {metrics['R2']:.4f}, MAE:
    {metrics['MAE']:.4f}")
    best_model = min(results, key=lambda k: results[k]['MAE'])
    print(f'\nBest Model based on MAE: {best_model}')
```

5.12 Model Performance Visualization

Scatter plots were created to visualise the performance of each model by comparing actual vs predicted values.

5.12.1 Linear Regression:

```
Linear Regression - MSE: 0.0000, R2: 0.9989, MAE: 0.0031
```

This scatter plot compares the actual values versus the predicted values from the Linear Regression model used to predict bike usage patterns. The red dashed line represents the perfect prediction line where the predicted values equal the actual values. The close alignment of the data points along this line indicates a high level of accuracy in the model's predictions.

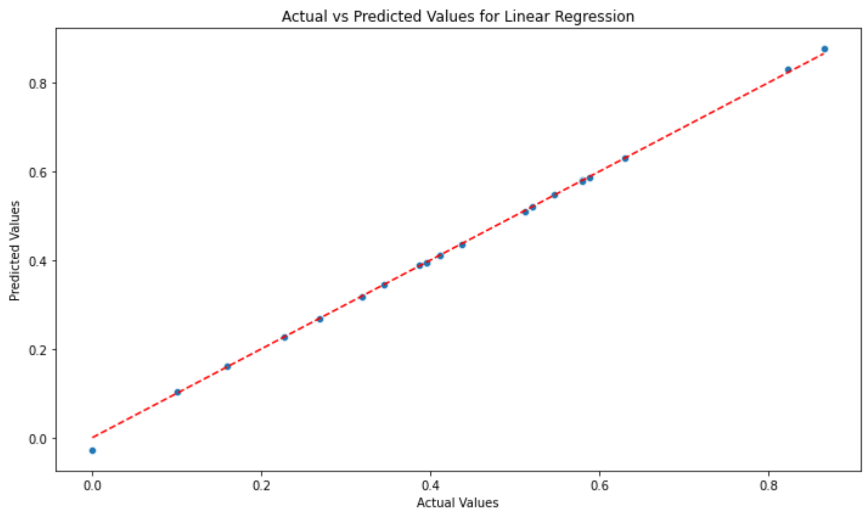


Figure 18: Actual vs Predicted Values for Linear Regression Model

5.12.2 Decision Tree

Decision Tree - MSE: 0.0007, R2: 0.9837, MAE: 0.0161
This scatter plot compares the actual values versus the predicted values from the Decision Tree model used to predict bike usage patterns. The red dashed line represents the perfect prediction line where the predicted values equal the actual values. The data points are closely aligned along this line, indicating that the Decision Tree model performs well in predicting the target variable.

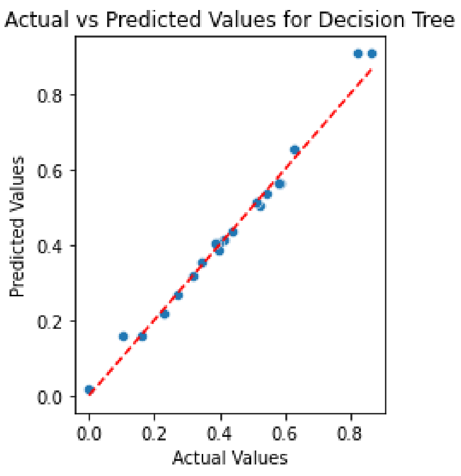


Figure 19: Actual vs Predicted Values for Decision Tree Model

5.12.3 Random Forest

Random Forest - MSE: 0.0006, R2: 0.9852, MAE: 0.0136
This scatter plot compares the actual values versus the predicted values from the Random Forest model used to predict bike usage patterns. The red dashed line represents the perfect prediction line where the predicted values equal the actual

values. The close clustering of data points along this line indicates that the Random Forest model has high accuracy in its predictions.

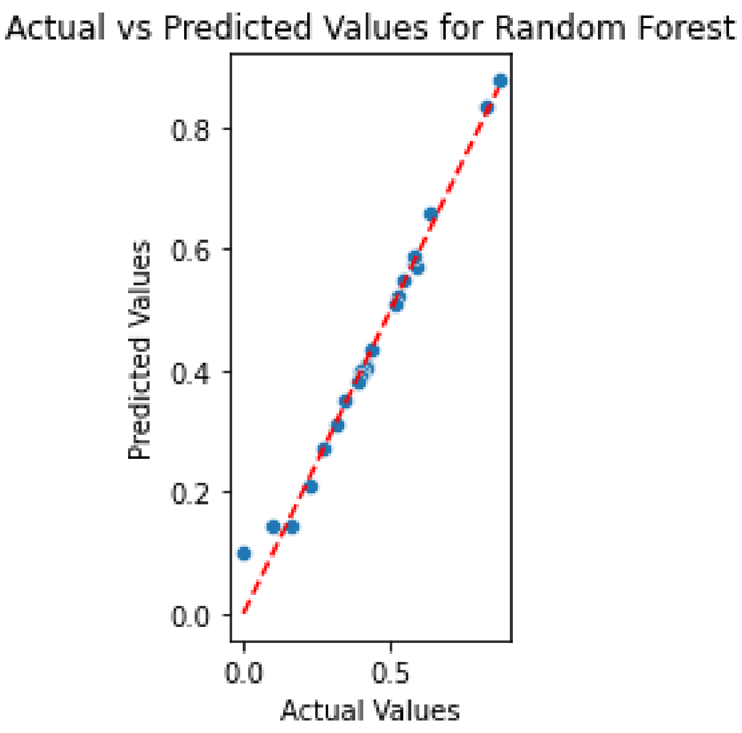


Figure 20: Actual vs Predicted Values for Random Forest Model

5.12.4 Neural Networks

Neural Network - MSE: 0.0112, R2: 0.7260, MAE: 0.0825
This scatter plot compares the actual values versus the predicted values from the Neural Network model used to predict bike usage patterns. The red dashed line represents the perfect prediction line where the predicted values equal the actual values. The data points show more dispersion around the line compared to other models, indicating that while the Neural Network model makes reasonably accurate predictions, there is some variance.

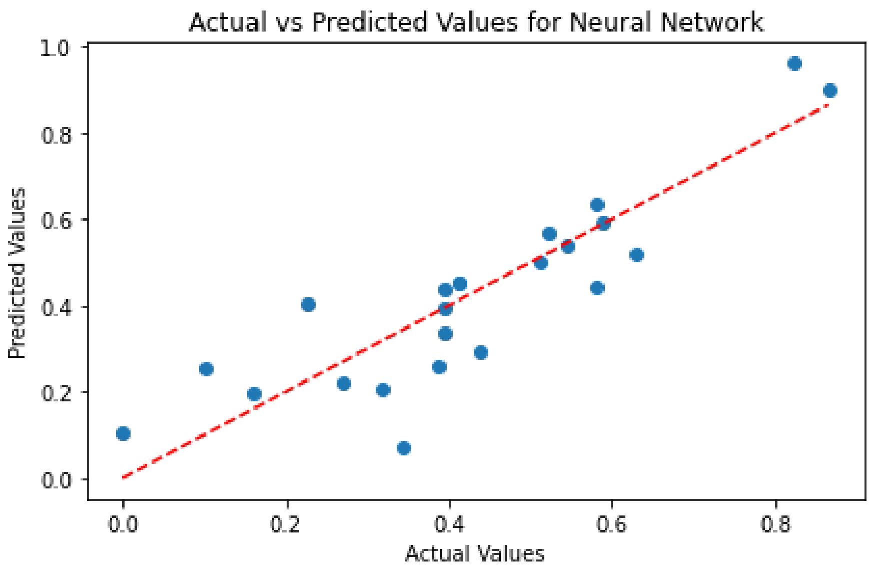


Figure 21: Actual vs Predicted Values for Neural Network Model

5.12.5 XGBoost

XGBoost - MSE: 0.0012, R2: 0.9703, MAE: 0.0191

This scatter plot compares the actual values versus the predicted values from the XGBoost model used to predict bike usage patterns. The red dashed line represents the perfect prediction line where the predicted values equal the actual values. The close clustering of data points along this line indicates that the XGBoost model performs well in predicting the target variable, with high accuracy.

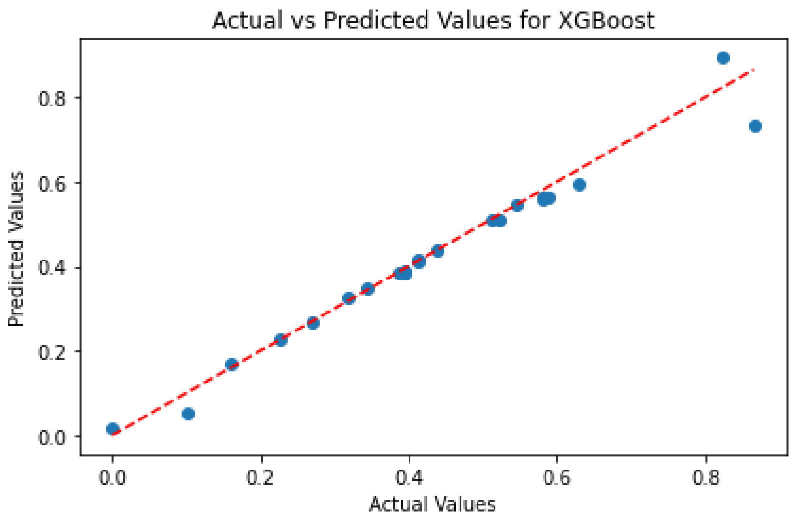


Figure 22: Actual vs Predicted Values for XGBoost

5.12.6 SVM

SVM - MSE: 0.0090, R2: 0.7808, MAE: 0.0704

This scatter plot compares the actual values versus the predicted values from the Support Vector Machine (SVM) model used to predict bike usage patterns. The red dashed line represents the perfect prediction line where the predicted values equal the actual values. The dispersion of data points around this line indicates that while the SVM model makes reasonably accurate predictions, there is some variance.

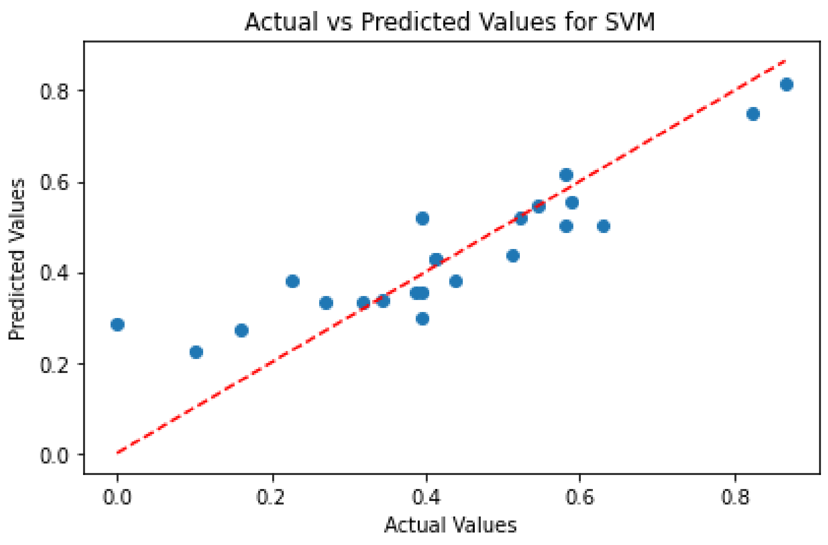


Figure 23: Actual vs Predicted Values for SVM

5.12.7 KNN

KNN - MSE: 0.0084, R2: 0.7950, MAE: 0.0712
This scatter plot compares the actual values versus the predicted values from the K-Nearest Neighbors (KNN) model used to predict bike usage patterns. The red dashed line represents the perfect prediction line where the predicted values equal the actual values. The dispersion of data points around this line suggests that while the KNN model can predict reasonably well, there are variations, especially at higher actual values.

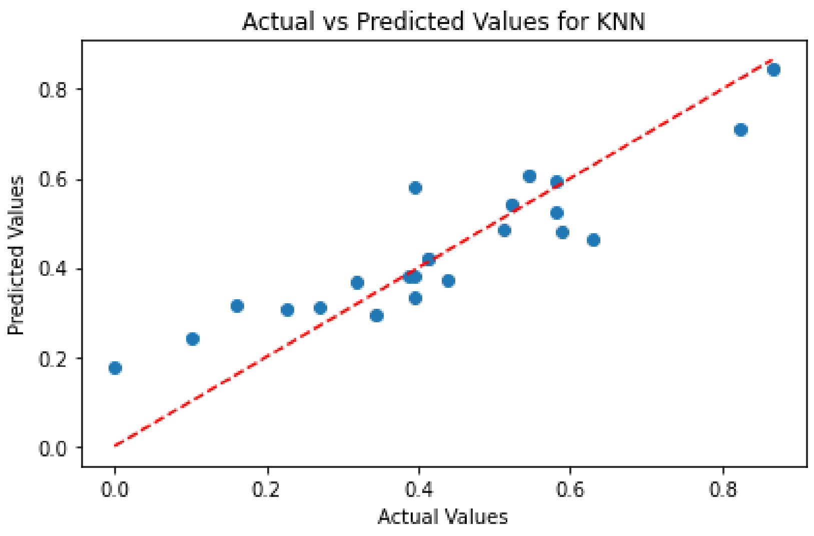


Figure 24: Actual vs Predicted Values for the KNN Model

6 Conclusion

6.1 Question 1 Conclusions

From the analysis and visualisation of the bike rental data, the following conclusions can be drawn:

1. The total number of bike rentals in June was 876,068.
2. Classic bikes significantly outnumbered PSBC_Ebikes, with 817,285 rentals (93.29%) compared to 58,783 rentals (6.71%).
3. The majority of bike rentals occurred in the afternoon, accounting for 329,758 rentals (37.64%).
4. The highest number of bikes rented in a single day was on 21st June 2023, with a total of 34,177 rentals.
5. Hyde Park Corner, Hyde Park, was the most popular start station, with 6,585 bikes rented from this location.
6. Hyde Park Corner, Hyde Park, was also the most frequent end station, with 6,633 bikes being returned there.
7. The route starting and ending at Hyde Park Corner, Hyde Park, was the most common, indicating that many bikes rented from this station were returned to the same location.
8. The highest rental activity occurred between 3 PM and 8 PM, with a notable peak at 7 AM as well.
9. The bike with the number 59224 was the most frequently used during June.
10. The highest number of rentals occurred in Week 4, particularly during Thursday and Friday afternoons.
    These insights provide a comprehensive understanding of bike rental patterns, helping to optimise bike distribution, improve service, and enhance user satisfaction.

6.2 Question 2 Conclusions

Conclusion from Forecasting and Machine Learning Models:

1. Among the various machine learning models tested, the Linear Regression method demonstrated the highest accuracy for weather forecasting. This conclusion is based on its lowest Mean Absolute Error (MAE) compared to other models such as Decision Trees, Random Forests, and Neural Networks.
2. Our analysis focused on the feature 'temperature' (temp). The results indicate that higher temperatures correlate with increased bike rentals. This insight suggests that temperature is a significant factor influencing bike rental behaviour. It can be inferred that more people are likely to rent bikes when the weather is warm and conducive to outdoor activities.
3. To enhance the robustness of our model, we can extend our testing to include additional features. This can be done by either replacing the temperature variable in the current code or by incorporating multiple features into the test code. Potential features to consider include:
    a. Humidity: Higher humidity levels might deter bike rentals.
    b. Precipitation: Rainy conditions are likely to reduce bike rentals.
    c. Wind Speed: Strong winds could negatively impact the likelihood of bike rentals.
    d. Day of the Week: Weekdays vs. weekends might show different rental patterns.

    e. Hour of the Day: Peak hours might coincide with higher rental rates.

4. While the initial results are promising, further optimisation of the models can be achieved through hyperparameter tuning. Techniques such as Grid Search and Random Search can be employed to find the best parameters for models like Random Forest and XGBoost, potentially improving their performance.
5. Although Linear Regression performed well, it assumes a linear relationship between the features and the target variable. Exploring models that can capture non-linear relationships, such as Support Vector Machines (SVM) with non-linear kernels, or ensemble methods like Gradient Boosting, could provide additional insights and potentially better performance.
6. The findings from this analysis can be used by bike rental companies to optimise their operations. For instance, they can adjust the number of available bikes based on the weather forecast or develop targeted marketing strategies to boost rentals during favourable weather conditions.
7. Continuous improvement of the model requires ongoing data collection and analysis. Ensuring high-quality, up-to-date data will help in making accurate predictions. Additionally, incorporating real-time data feeds could enhance the model's predictive capabilities, making it more responsive to changing conditions.

7 References

Feng, Y. W. (2017). *Feng, Y., Wang, S., (2017). A forecast for bicycle rental demand based on random forests and multiple linear regression. Proc. IEEE/ACIS International Conference on Computer and Information Science, ICIS, 101–105.*

Gu, K. &. (2023). *Gu, K.Y., Lin, Y., (2023). Prediction for bike-sharing demand in London using multiple linear regression and random forest. Proceedings Volume 12803, Fifth International Conference on Artificial Intelligence and Computer Science (AICS 2023); 128031I.*

MDPI. (2023). *MDPI, (2023). Short-Term Demand Prediction of Shared Bikes Based on LSTM Neural Network. Electronics, 12(6), 1381.*

ScienceDirect. (2023). *ScienceDirect, (2023). Effect of dockless bike-sharing scheme on the demand for London Cycle Hire.*

VisualCrossing. (2023). *VisualCrossing, (2023). Historical weather information for London.*

https://www.kaggle.com/code/hmavrodiev/bike-sharing-prediction-rf-xgboost

https://github.com/hristo-mavrodiev/London_bicycle_predictions/

8 Appendix

For the complete code of Question 1 that is section 3 please refer to file Ques1-28965736.py made on Spyder
For the complete code of Question 2 that is section 4 please refer to file Ques2-28965736.py made on Spyder