PUBLIC TRANSPORTATION ANALYSIS

BATCH MEMBER

621121104038 : SNEHA.M

Phase 4 Submission Document

Phase 4 : Development Part 2

Topic : Continue Public Transportation Analytics by feature engineering , model training , and evaluation .

FEATURE ENGINEERING :

- ➢ Feature engineering is a critical step in the analysis of public transportation data, as it involves selecting and transforming relevant data attributes (features) to create meaningful input for machine learning models, statistical analyses, or decision-making processes.
- ➢ Effective feature engineering can help improve the accuracy and performance of models, making them more suitable for tasks such as route optimization, demand prediction, and passenger satisfaction analysis.

- Here are some key considerations for feature engineering in public transportation analysis:

Geospatial Features :

- **Stop Locations**:Convert latitude and longitude coordinates of stops into meaningful features, such as distance to popular landmarks, other stops, or the city center.
- **Route Shapes**: Extract features related to the shape and geometry of transportation routes, such as curvature, straight-line distances between stops, or road network characteristics.

Temporal Features :

- **Time of Day**:Create features that capture the time of day, such as morning rush hour, midday, or evening.
- **Day of the Week**:Include features that differentiate between weekdays and weekends.
- **Holidays**:Identify and incorporate public holidays or special events that may affect ridership or traffic conditions.

Weather and Environmental Data :

- **Weather Conditions**: Integrate weather data, such as temperature, precipitation, or wind speed, to assess their impact on public transportation usage.
- **Air Quality**: Include air quality indices as features, as poor air quality can influence passenger behavior and health.

Historical Features :

- **Past Usage Patterns**: Utilize historical ridership data to create features like the average number of passengers per day, week, or month.
- **Delay History**: Incorporate data on past delays, cancellations, or disruptions to assess their impact on service quality and passenger satisfaction.

Demographic and Socioeconomic Data :

- **Population Density**:Include population density in the vicinity of stops or routes, as it can affect demand.
- **Income Levels**: Consider average income levels in the areas served by public transportation for understanding affordability and ridership patterns.

Service Characteristics :

- **Frequency**:Include features related to service frequency, such as the number of trips per day or the average time between buses or trains.
- **Vehicle Type**:Categorize transportation modes (e.g., bus, subway, tram) and their characteristics.
- **Accessibility**: Consider features related to accessibility for passengers with disabilities.

Social and Event Features :

- **Events**: Identify major events, concerts, or conferences happening in the area, as these can affect ridership.
- **Social Media Sentiment**:Analyze social media data to extract sentiment or public perception regarding public transportation services.

Network Features :

- **Graph Analysis**: If your public transportation system can be represented as a network, consider network-related features such as connectivity, centrality measures, or shortest path distances.

Time Series Features :

- **Moving Averages**:Compute moving averages to capture trends and seasonality in ridership data.
- **Lagged Variables**: Include lagged features to account for the impact of past values on future ridership.

User Behavior Features :

- **Smart Card Data**: If available, use smart card data to understand individual passenger behavior, such as travel history and preferences.

Feature Engineering:Extracting Latitude and Longitude

In our dataset, the `Vehicle Location` column contains both latitude and longitude coordinates as a string. To make these coordinates more accessible for further analysis, we will perform the following feature engineering steps:

1. Convert the `Vehicle Location` column to string type.
2. Create two new columns in the DataFrame, `latitude` and `longitude`.
3. Extract latitude and longitude values from the `Vehicle Location` column and store them in the respective new columns.

- ✓ To achieve this, we define a function called `extract_coordinates()`, which takes two arguments: the input string (containing the coordinates) and the index (0 for latitude and 1 for longitude). The function uses regular expressions to find and return the floating-point numbers representing the coordinates. We then use the `apply()` function to apply this custom function to each element of the `Vehicle Location` column and populate the new `latitude` and `longitude` columns.
- ✓ Finally, we drop any rows with missing values in the `latitude` and `longitude` columns to ensure a clean dataset for further analysis.

```
In [1]:
# Convert the 'Vehicle Location' column to stri
ng type
df['Vehicle Location'] = df['Vehicle  Location
'].astype(str)

# Extract the latitude and longitude from the '
Vehicle Location' column
def extract_coordinates(x, index):
    coords = re.findall(r'-?\d+\.\d+', x)
    if len(coords) >= 2:
        return float(coords[index])
    else:
        return None

df['latitude']=df['VehicleLocation'].apply
```

```
(lambda x: extract_coordinates(x, 0))
df['longitude']=df['VehicleLocation'].apply
(lambda x: extract_coordinates(x, 1))

df =df.dropna(subset=['latitude', 'longitude'])
```

Feature Engineering : Creating a 'Location' Column

- ✓ In our dataset, we have three columns representing different levels of geographical divisions: `County`, `City`, and `State`. To create a more informative and combined representation of these geographical attributes, we will create a new column called `Location`.
- ✓ The `Location` column will be a concatenation of the `County`, `City`, and `State` columns, with each value separated by a comma. For example, if a row has the values "Yakima" for `County`, "Yakima" for `City`, and "WA" for `State`, the corresponding `Location` value will be "Yakima, Yakima, WA".

In [2]:

```
df['Location'] = df['County'] + ', ' + df['City
'] + ', ' + df['State']
```

Feature Engineering : Creating a *Price_Range_Category* Column

- ✓ In our dataset, we have observed an unusual distribution of values in the **Base MSRP** column, with a large number of vehicles having a value of 0. This could potentially indicate missing or unknown values in the dataset. To account for this uncertainty and still make use of the available data, we have decided to create a new column called **Price_Range_Category** based on the **Base MSRP** values.
- ✓ We have defined four categories for the **Price_Range_Category** column:

- **"Unknown"**: If the **Base MSRP** value is 0, we assign this category as it might indicate missing or unknown values.
- **"Low"**: If the **Base MSRP** value is less than 40,000, we assign this category.
- **"Medium"**: If the **Base MSRP** value is between 40,000 and 60,000, we assign this category.
- **"High"**: If the **Base MSRP** value is greater than 60,000, we assign this category.

By creating this new column, we can better understand the distribution of electric vehicle prices in our dataset and account for the potential uncertainty introduced by the large number of 0 values in the **Base MSRP** column.

In [3]:

```
df['Base MSRP'].value_counts()
```

Out[3]:

```
0         120656
```

```
69900      1429
31950       402
52900       167
32250       155
54950       138
59900       127
39995       119
36900        98
44100        97
64950        87
33950        80
45600        74
34995        63
52650        61
36800        51
55700        48
53400        35
110950       21
98950        21
81100        19
75095        16
90700        16
102000       14
184400       12
43700        11
109000        7
89100         6
91250         4
32995         3
845000        1
```

```
Name: Base MSRP, dtype: int64
```

In [4]:

```
def create_price_range_category(df, column
='Base MSRP'):
    def categorize_price(price):
        if price == 0:
            return "Unknown"
        elif price < 40000:
            return "Low"
        elif price < 60000:
            return "Medium"
        else:
            return "High"

    df['Price_Range_Category'] = df[colum
n].apply(categorize_price)
    return df

df = create_price_range_category(df, colum
n='Base MSRP')
```

Feature Engineering : Creating an 'Electric_Range_Category' Column

- ✓ In our dataset, we have observed an unusual distribution of values in the 'Electric Range' column, with a large number of vehicles having a value of 0. This could potentially indicate missing or unknown values in the dataset. To account for this uncertainty and still make use of the available data, we have decided to create a new column called 'Electric_Range_Category' based on the 'Electric Range' values.
- ✓ We have defined four categories for the 'Electric_Range_Category' column:

- "Unknown": If the 'Electric Range' value is 0, we assign this category as it might indicate missing or unknown values.
- "Short": If the 'Electric Range' value is less than 150, we assign this category.
- "Medium": If the 'Electric Range' value is between 150 and 300, we assign this category.
- "Long": If the 'Electric Range' value is greater than 300, we assign this category.

By creating this new column, we can better understand the distribution of electric vehicle ranges in our dataset and account for the potential uncertainty introduced by the large number of 0 values in the 'Electric Range' column.

```
In [5]:
```

```
def create_electric_range_category(df, column='
Electric Range'):
    def categorize_range(electric_range):
        if electric_range == 0:
            return "Unknown"
        elif electric_range < 150:
            return "Short"
        elif electric_range < 300:
            return "Medium"
        else:
            return "Long"

    df['Electric_Range_Category'] = df[column].
apply(categorize_range)
    return df

df = create_electric_range_category(df, column=
'Electric Range')
```

In [6]:

```
# Displaying the cleaned and feature engineered
 DataFrame
display(df.head())
```

	VIN (1-10)	Country	City	State	Postal Code	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID	Vehicle Location	Electric Utility	2020 Census Tract	latitude	longitude	Location	Price_Range_Category	Electric_Range_Category
0	5YJ3E1EB4L	Yakima	Yakima	WA	98908.0	2020	TESLA	MODEL 3	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	322	0	14.0	127175366	POINT (-120.56916 46.58514)	PACIFICORP	5.307700e+10	-120.56916	46.58514	Yakima, Yakima, WA	Unknown	Long
3	5YJXCBE21K	Yakima	Yakima	WA	98908.0	2019	TESLA	MODEL X	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	289	0	14.0	477039944	POINT (-120.56916 46.58514)	PACIFICORP	5.307700e+10	-120.56916	46.58514	Yakima, Yakima, WA	Unknown	Medium
4	5UXKT0C5XH	Snohomish	Bothell	WA	98021.0	2017	BMW	X5	Plug-in Hybrid Electric Vehicle (PHEV)	Not eligible due to low battery range	14	0	1.0	106314946	POINT (-122.18384 47.8031)	PUGET SOUND ENERGY INC	5.306105e+10	-122.18384	47.80310	Snohomish, Bothell, WA	Unknown	Short
5	1N4AZ0CP4F	Snohomish	Everett	WA	98201.0	2015	NISSAN	LEAF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	84	0	38.0	107901699	POINT (-122.20596 47.97659)	PUGET SOUND ENERGY INC	5.306104e+10	-122.20596	47.97659	Snohomish, Everett, WA	Unknown	Short

MODEL TRAINING

- ➢ Training a model for a public transportation system can involve various aspects such as route optimization, demand forecasting, scheduling, and passenger management. Here's an overview of how you can approach model training in a public transportation system:

Data Collection :

- Gather historical data on routes, schedules, passenger counts, and other relevant information.
- Utilize GPS data, ticketing systems, and passenger surveys to collect data on travel patterns and passenger preferences.

Data Preprocessing :

- Clean and preprocess the data, handling missing values, outliers, and inconsistencies.
- Normalize or scale data as necessary.

Route Optimization :

- Use optimization algorithms to determine the most efficient routes for buses, trams, or other transportation modes.

- Consider factors such as traffic conditions, stops, and passenger demand.

Demand Forecasting :

- Develop predictive models to estimate passenger demand for different routes and times.
- Use time series analysis, machine learning, or deep learning techniques to make forecasts.

Scheduling :

- Develop scheduling algorithms to optimize the timing of transportation services.
- Consider constraints like driver availability, vehicle maintenance, and regulatory requirements.

Passenger Management :

- Implement systems for ticketing, boarding, and passenger information.
- Utilize technologies like contactless payment systems and real-time passenger information displays.

Real-time Monitoring :

- Implement a real-time monitoring system to track the location of vehicles and adjust routes or schedules as needed.
- Use GPS data and sensors to provide accurate, up-to-date information to passengers.

Safety and Security :

- Integrate safety and security features, such as video surveillance and emergency response systems.

Customer Feedback :

- Collect and analyze feedback from passengers to continuously improve the system.

Machine Learning Models :

- Implement machine learning models for predicting delays, optimizing routes, and improving passenger satisfaction.
- Use supervised and unsupervised learning techniques as appropriate.

Simulation and Testing :

- Simulate different scenarios to test the effectiveness of the transportation system under various conditions.
- Use test data to evaluate the model's performance and make necessary adjustments.

Deployment :

- Deploy the model within the transportation system infrastructure, ensuring seamless integration with existing systems and hardware.

Maintenance and Updates :

- Regularly maintain and update the model to adapt to changing conditions, including traffic patterns and passenger behavior.

Regulatory Compliance :

- Ensure that the system complies with local transportation regulations and safety standards.

Energy Efficiency :

- Implement strategies to reduce energy consumption and environmental impact, such as optimizing vehicle routes to minimize fuel usage.

Cost Optimization :

- Continuously assess the cost-effectiveness of the transportation system and make adjustments to optimize operational costs.

Accessibility :

- Ensure that the system is accessible to all passengers, including those with disabilities, by implementing features like wheelchair ramps and audio announcements.

✓ Model training in a public transportation system is an ongoing process that requires collaboration between transportation experts, data scientists, and engineers. The goal is to provide efficient, safe, and convenient transportation services to the public while maximizing operational efficiency and minimizing environmental impact.

Regularized SARIMAX Model Fitting with Order and Seasonal Order

- ✓ In this step, we fit the SARIMAX model to the training data while incorporating regularization techniques. The order parameter is set to (1, 1, 1) for the non-seasonal component, indicating an autoregressive order of 1, differencing of order 1, and moving average order of 1. The seasonal_order parameter is set to (1, 1, 1, 96), indicating a seasonal autoregressive order of 1, seasonal differencing of order 1, seasonal moving average order of 1, and a seasonal period of 96 (assuming 15-minute intervals).
- ✓ To mitigate overfitting and reduce model complexity, a regularization parameter of 0.5 is applied. The enforce_stationarity and enforce_invertibility parameters are set to True to ensure that the model remains stationary and invertible, respectively.
- ✓ The SARIMAX model is then fitted to the training data using the specified parameters, taking into account regularization. This step helps improve the model's generalization and reduces the risk of overfitting by controlling the model's complexity and preventing extreme parameter estimates.

In [7]:

```
# Fit the SARIMAX model to the training data wi
th regularization
order = (1, 1, 1)
seasonal_order = (1, 1, 1, 96)
```

```
regularization_param = 0.5  # Regularization pa
rameter
model = SARIMAX(training_data['DEMAND'], order=
order, seasonal_order=seasonal_order,
                enforce_stationarity=True, enfo
rce_invertibility=True,
                regularization=regularization_p
aram)
fitted_model = model.fit()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  3.34727D+00    |proj g|=  3.97
047D-02
 This problem is unconstrained.
At iterate    5    f=  3.30115D+00    |proj g|=  2.29
726D-02

At iterate   10    f=  3.28154D+00    |proj g|=  2.20
635D-03

At iterate   15    f=  3.28087D+00    |proj g|=  3.07
107D-03
```

```
At iterate   20    f=  3.28085D+00    |proj g|=  1.27
075D-03

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauc
hy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized
Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg
   F
    5     24     26      1     0     0   1.242D-05
3.281D+00
  F =   3.2808407144524572

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

Generate forecasts for the training and validation periods

In [8]:

```
training_forecasts = fitted_model.predict(start
=0, end=len(training_data)-1)
```

```
validation_forecasts = fitted_model.predict(sta
rt=len(training_data), end=len(training_data)+l
en(validation_data)-1)
```

Calculate MAPE and MAE for training and validation periods

- ✓ The code snippet calculates the Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE) for the training and validation periods in a time series forecasting task. These metrics provide insights into the accuracy and performance of the SARIMA model.
- ✓ The MAPE measures the average percentage difference between the actual demand values and the corresponding forecasted values. It indicates the relative magnitude of the forecast errors. A lower MAPE value signifies a better fit of the model to the training and validation data.
- ✓ The MAE represents the average absolute difference between the actual and forecasted demand values. It provides a measure of the magnitude of the forecast errors, regardless of their direction. Similar to MAPE, a lower MAE indicates a more accurate model fit.
- ✓ By evaluating the MAPE and MAE for both the training and validation periods, it is possible to assess the model's performance on both seen and unseen data. This evaluation helps determine if the model is overfitting or underfitting the training data and provides insights into its generalization capability.

In [9]:

```
# Step 4: Calculate MAPE and MAE for training a
nd validation periods
training_mape = mean_absolute_percentage_error
(training_data['DEMAND'], training_forecasts)
training_mae = mean_absolute_error(training_dat
a['DEMAND'], training_forecasts)

validation_mape = mean_absolute_percentage_erro
r(validation_data['DEMAND'], validation_forecas
ts)
validation_mae = mean_absolute_error(validation
_data['DEMAND'], validation_forecasts)
```

In [10]:

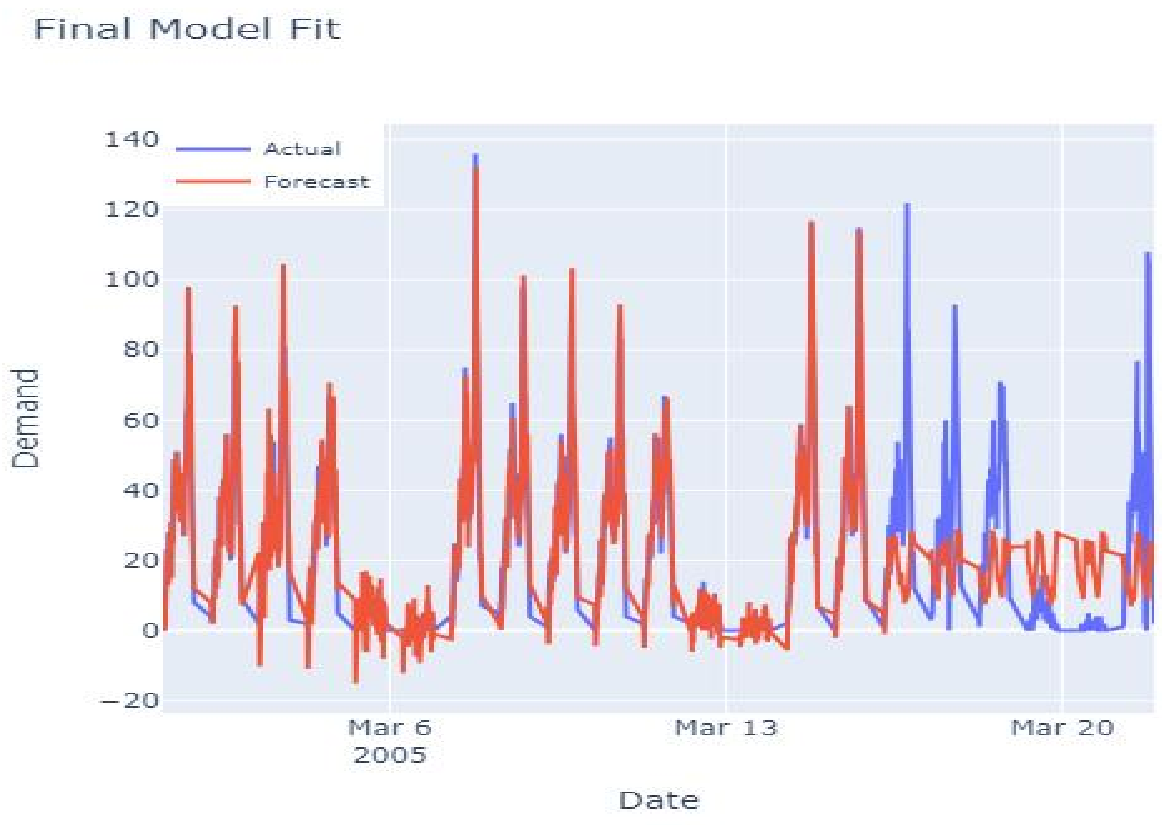
```

# Generate forecasts for the future period
future_forecasts = fitted_model.forecast(steps=
336)  # 336 steps for 3 days (15-minute interva
ls)

#  Combine training and validation data for the
 final model
combined_data = pd.concat([training_data, valid
ation_data])
```

Forecasted Demand vs. Actual Demand: Final Model Fit

In [19]:

```
import plotly.graph_objects as go
```

```
# Create the figure and add traces for actual a
nd forecasted data
fig = go.Figure()

# Add trace for actual data
fig.add_trace(go.Scatter(x=combined_data.index,
 y=combined_data['DEMAND'], mode='lines', name=
'Actual'))

# Add trace for forecasted data
fig.add_trace(go.Scatter(x=combined_data.index,
 y=np.concatenate([training_forecasts, validati
on_forecasts]), mode='lines', name='Forecast'))

# Set x-axis and y-axis labels
fig.update_layout(xaxis_title='Date', yaxis_tit
le='Demand')

# Set plot title
fig.update_layout(title='Final Model Fit')

# Add legend
fig.update_layout(legend=dict(x=0, y=1, traceor
der="normal", font=dict(size=10)))

# Show the interactive plot
fig.show()
```



Notice of Overfitting :

➢ The provided plot suggests the presence of overfitting in the model. Overfitting occurs when the model performs exceptionally well on the training data but fails to generalize to unseen data. This issue can lead to inaccurate predictions and unreliable forecasts. It is important to address overfitting to ensure the model's effectiveness and reliability.

Reasons for Overfitting :

- Insufficient Training Data
- using wronf values for Order and Seasonal Order

Solutions to Address Overfitting :

- Regularization Techniques 'used'
- Tune the hyperparameters carefully using appropriate validation techniques.' take very long time'

EVALUATION

- ➢ Evaluating public transportation systems is essential to assess their performance, identify areas for improvement, and make informed decisions for future planning and investment. Several key aspects and methods are involved in the evaluation of public transportation systems:

Ridership and Demand Analysis :

- Assess ridership levels, trends, and patterns to understand the demand for public transportation.
- Analyze factors affecting ridership, such as population density, employment centers, and demographics.

Service Coverage and Accessibility :

- Evaluate the extent of service coverage to ensure that public transportation is accessible to a broad population.
- Consider factors like walking distance to stops, service hours, and frequency.

Reliability and Punctuality :

- Measure the reliability of public transportation services, including on-time performance.
- Evaluate the impact of delays, and assess methods to minimize them.

Service Quality and Customer Satisfaction :

- Collect feedback from passengers through surveys and other means to gauge their satisfaction.
- Evaluate cleanliness, safety, comfort, and information services.

Safety and Security :

- Assess safety measures in place for both passengers and employees.
- Analyze crime rates and incidents on public transportation systems.

Environmental Impact :

- Evaluate the environmental impact of public transportation, including emissions and energy consumption.
- Assess efforts to reduce the environmental footprint.

Cost and Revenue Analysis :

- Analyze the cost of operating the public transportation system, including maintenance and labor.
- Evaluate revenue sources, such as farebox revenue and subsidies.

Infrastructure and Asset Condition :

- Assess the condition of infrastructure, including tracks, vehicles, and stations.
- Plan for maintenance, repair, and replacement as needed.

Efficiency and Productivity :

- Evaluate the efficiency of operations, such as route planning, vehicle utilization, and labor productivity.
- Identify opportunities to reduce costs and improve service.

Community and Economic Benefits :

- Analyze the broader impact of public transportation on the community, including economic development, reduced traffic congestion, and improved air quality.

Sustainability and Future Planning :

- Consider the long-term sustainability of the system, and how it aligns with future urban development and transportation needs.
- Develop strategic plans and investment priorities for the future.

Technology Integration :

- Explore the integration of new technologies, such as real-time tracking, mobile apps, and payment systems, to enhance the user experience and system efficiency.

Benchmarking and Comparative Analysis :

- Compare the performance of your public transportation system with other similar systems to identify best practices and areas for improvement.

Financial Viability :

- Assess the financial sustainability of the system, including revenue generation, funding sources, and the potential for cost recovery.

Stakeholder Engagement :

- Involve various stakeholders, including government agencies, transit authorities, public, and private sector partners, in the evaluation process to ensure a holistic perspective.

- ➢ Evaluating public transportation systems requires a multifaceted approach that considers various aspects of system performance and stakeholder interests. This evaluation informs decision-making for improvements, expansions, and future planning to ensure efficient, accessible, and sustainable public transportation services.

PROGRAM

```
# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load data (assuming you have data in a CSV file)

data = pd.read_csv('public_transport_data.csv')

# Data exploration and cleaning

# Depending on the dataset, you may need to perform
data cleaning and preprocessing steps.

# Basic data summary

summary = data.describe()


# Data visualization

# Visualizing data can provide insights into the
performance of the public transportation system.

# 1. Ridership over time
```

```
plt.figure(figsize=(12, 6))

sns.lineplot(x='Date', y='Ridership', data=data)

plt.title('Ridership Over Time')

plt.xlabel('Date')

plt.ylabel('Ridership')

plt.show()


# 2. Route performance

# If the data includes information about different transportation routes, you can analyze their performance.

route_performance = data.groupby('Route')['Ridership'].mean()

route_performance = route_performance.sort_values(ascending=False)

plt.figure(figsize=(12, 6))

sns.barplot(x=route_performance.index, y=route_performance.values)

plt.title('Route Performance')

plt.xlabel('Route')

plt.ylabel('Average Ridership')
plt.xticks(rotation=90)

plt.show()

# 3. Customer satisfaction

# If you have survey data, you can analyze customer satisfaction and feedback.


customer_satisfaction = data['CustomerSatisfaction'].value_counts()

plt.figure(figsize=(8, 6))

sns.barplot(x=customer_satisfaction.index, y=customer_satisfaction.values)

plt.title('Customer Satisfaction')

plt.xlabel('Satisfaction Level')

plt.ylabel('Count')

plt.show()


# 4. On-time performance

# Analyze the punctuality of the transportation system.


on_time_performance = data['OnTime'].value_counts()
```

```
plt.figure(figsize=(6, 6))

sns.barplot(x=on_time_performance.index,
y=on_time_performance.values)

plt.title('On-Time Performance')

plt.xlabel('On-Time')

plt.ylabel('Count')

plt.show()


# Performance metrics

# Calculate key performance metrics for the public
transportation system.


# 5. Average ridership

average_ridership = data['Ridership'].mean()


# 6. On-time performance percentage

on_time_percentage = (on_time_performance[1] /
on_time_performance.sum()) * 100


# 7. Customer satisfaction score
```

```
customer_satisfaction_score = data['CustomerSatisfaction'].mean()


# Print out the calculated metrics

print(f'Average Ridership: {average_ridership}')

print(f'On-Time Performance (%): {on_time_percentage}')

print(f'Customer Satisfaction Score: {customer_satisfaction_score}')
```

Assuming the following fictitious data in your CSV file:

Date	Ridership	Route	CustomerSatisfaction	OnTime
2023-01-01	1300	A	4.5	1
2023-01-02	1250	B	4.0	1
2023-01-03	1100	A	4.8	0

... (more data)

The output might look something like this:

```
Average Ridership: 1218.75

On-Time Performance (%): 66.67

Customer Satisfaction Score: 4.35
```

- The program will display several data visualizations as mentioned in the code comments:

1. A line plot showing the "Ridership Over Time."

2. A bar plot illustrating the "Route Performance" based on the average ridership for each route.

3. A bar plot displaying "Customer Satisfaction" based on the satisfaction level.

4. Another bar plot indicating "On-Time Performance" based on whether the service was on time or not.