UNIVERSITY OF TEXAS ARLINGTON | COLLEGE OF BUSINESS

2242-INSY-5339-002

Principles of Business Data Mining

Spring 2024

Professor: Riyaz Sikora

PROJECT REPORT

URBAN TRAFFIC DENSITIES IN CITIES

GROUP MEMBERS:

Abhishek Babu Madini - ID: 1002177917
Nikhita Kalburgikar – ID: 1002160264
Nivetha Thangaraj – ID: 1002154961
Vaishnavi Palle – ID: 1002172756
Yuvarekha Mahendran - ID:1002175347

INTRODUCTION (EXECUTIVE SUMMARY):

In the bustling metropolis of our futuristic urban landscape, understanding and managing traffic dynamics is paramount for ensuring efficient and sustainable city living. To achieve this, we set out on a perceptive exploration of the complex network of variables influencing traffic patterns, trends, and energy usage.

Our main goal is to use data-driven methods to forecast traffic density while taking temporal and environmental factors into account. We want to find important relationships and patterns by exploring the subtleties of various vehicle types and their effects on traffic situations and energy use. By means of thorough study, we aim to maximize traffic management solutions, opening the door for increased sustainability and efficiency in our dynamic metropolis. Come along with us as we negotiate the challenges of urban mobility to create a future where cities flourish, traffic moves smoothly, and energy is saved.

PROJECT MOTIVATION:

The motivation of analyzing this dataset is to extract valuable insights into traffic patterns, trends, and influential factors within the futuristic urban environment.

1. Predicting traffic density and speed based on environmental and temporal variables.

2. Understanding the impact of different vehicle types on traffic conditions and energy consumption.

3. Identifying patterns and correlations between traffic variables, environmental factors, and energy consumption.

4. Optimizing traffic management strategies for enhanced efficiency and sustainability in the futuristic cityscape. give me all detailed introduction paragraph for this.

BUSINESS QUESTION:

"How can data-driven insights into traffic patterns and congestion in our futuristic cityscape inform strategic decision-making to optimize transportation efficiency and mitigate congestion-related issues?

DATASET DESCRIPTION:

The dataset under discussion provides a comprehensive overview of transportation dynamics in several futuristic cities, capturing a range of variables that affect vehicle performance and urban traffic management. The cities featured in the dataset include SolarisVille, AquaCity, Neuroburg, Ecoopolis, and TechHaven. Each entry in the dataset specifies the city, the type of vehicle used (e.g., drones, flying cars, autonomous vehicles), the prevailing weather conditions during the observation (such as snowy, solar flare, clear, or rainy), and the economic conditions (stable, recession, or booming). Additionally, the dataset records the day of the week, hour of the day, vehicle speed, whether it was peak hour, any random events, energy consumption, and traffic density. This rich set of variables allows for a multifaceted analysis of how external and operational factors influence urban transportation efficiency.

DATA ANALYSIS USING VISUALISATION AND FINDINGS:

EXPLORATORY DATA ANALYSIS:

The exploratory data analysis conducted on the futuristic city traffic dataset involved a comprehensive examination of the categorical variables present in the data. This analysis was pivotal for understanding the distribution of various attributes across the dataset. Here's a detailed breakdown of each category as visualized in the provided bar charts:

1. **City**:
   The data encompasses a variety of cities, each with varying traffic volumes. SolarisVille and AquaCity are shown to have the highest traffic counts, suggesting these may be major urban centers or hubs in the futuristic setting. Other cities like Ecopolis and Neuroburg have significantly lower traffic volumes, which could indicate smaller populations or less developed transport infrastructure.
2. **Vehicle Type**:
   The distribution of vehicle types provides insights into the prevalent modes of transportation. Traditional cars and autonomous vehicles dominate, indicating a strong trend towards automated transport solutions in urban environments. Drones and flying cars, though less numerous, represent emerging technologies that might be subject to regulatory or technological barriers at present.
3. **Weather Conditions**:
   Weather conditions such as "Solar Flare" and "Clear" are notably prevalent, which might be typical of these cities' geographical and environmental settings. The presence of these conditions could influence vehicle operation and traffic management, necessitating adaptive traffic technologies and infrastructure.

4. **Economic Condition**:
   Economic conditions are categorized as "Booming", "Recession", and "Stable", with most traffic data collected during stable economic conditions. This variable is crucial for correlating economic activity with traffic patterns, as economic booms and recessions can significantly affect traffic volumes and vehicular movement.
5. **Day of the Week**:
   Traffic volume does not show significant variation across the days of the week, suggesting a consistent level of activity and possibly reflecting a steady work-life rhythm in these cities without traditional "weekend" dips.
6. **Hour of Day**:
   The hourly distribution shows peaks and troughs, typical of urban traffic behavior, with high volumes during what can be assumed as rush hours and lower volumes at night. This pattern is critical for planning efficient traffic flow and public transport schedules.
7. **Is Peak Hour**:
   This binary category confirms the presence of peak traffic hours, with a substantial portion of data indicating 'non-peak' hours. Such information is instrumental for dynamic traffic management systems that adjust controls based on real-time data.
8. **Random Event Occurred**:
   A small portion of the data indicates the occurrence of random events, which might include accidents or unplanned disruptions. These events are essential for analyzing traffic resilience and emergency response efficacy.

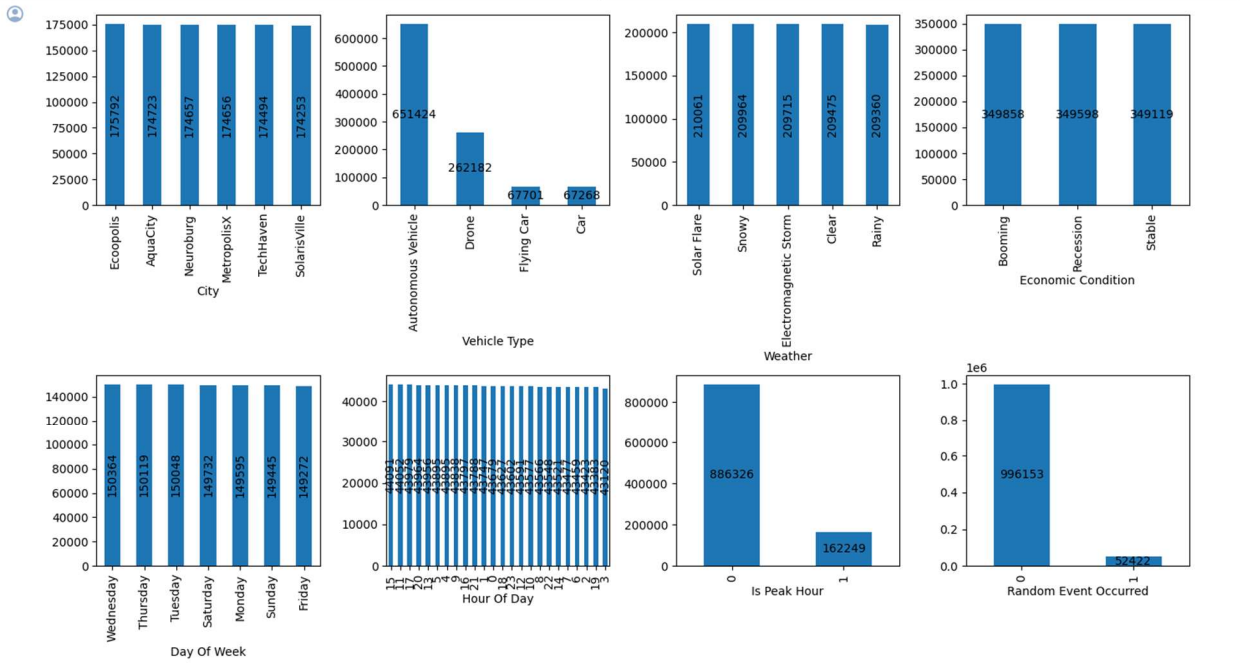
Each of these categories provides valuable insights into the dynamics of city traffic, contributing to our understanding of urban planning and traffic management in a futuristic context. The data reflects the complex interplay between urban infrastructure, technology, environmental conditions, and socioeconomic factors, all of which are crucial for developing adaptive and efficient traffic systems.

CODE:

EXPLORATORY DATA ANALYSIS

BASIC COUNT OF CATEGORICAL VALUES

```
cats = ["City", "Vehicle Type", "Weather", "Economic Condition", "Day Of Week", "Hour Of Day",
        "Is Peak Hour", "Random Event Occurred"]
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(14, 8))
index = 0
for i in range(2):
    for j in range(4):
        counts = traffic[cats[index]].value_counts()
        counts.plot(kind="bar", ax=axes[i][j])
        rotate = 0 if len(counts) <= 4 else 90
        for container in axes[i][j].containers:
            axes[i][j].bar_label(container, label_type="center", rotation=rotate)

        index += 1

plt.tight_layout()
plt.show()
```



This visual assessment was facilitated through the use of Seaborn's pairplot function, which generated a matrix of scatter plots for each pair of variables, alongside Kernel Density Estimation (KDE) plots for the distribution of individual variables.

Insights from the Visualization:

Variable Distributions:

- **Speed**: The KDE plot for speed shows a skewed distribution, indicating variability in vehicle speeds with a concentration at lower speeds and a long tail towards higher speeds.
- **Energy Consumption**: The distribution of energy consumption also appears skewed, suggesting that while most vehicles consume energy at a lower rate, there are significant outliers consuming much higher energy.
- **Traffic Density**: Traffic density is primarily concentrated at lower values with a sharp peak, indicating that high traffic density is less common but varies widely when it occurs.

Pairwise Relationships:

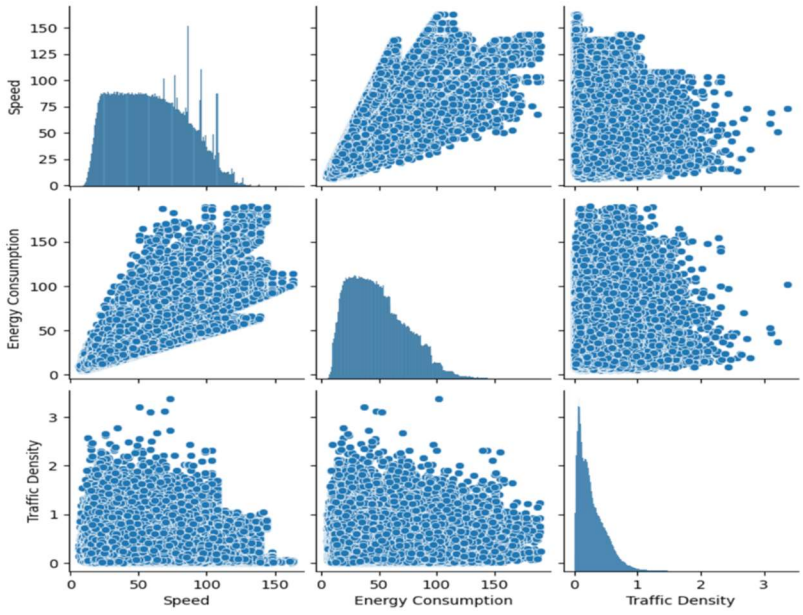
- **Speed and Energy Consumption**: The scatter plot reveals a positive relationship, suggesting that higher speeds generally correlate with higher energy consumption, likely due to increased power requirements at higher velocities.
- **Speed and Traffic Density**: The relationship between speed and traffic density displays an inverse pattern; higher traffic density is associated with lower speeds, which is typical in congested conditions.
- **Energy Consumption and Traffic Density**: This plot shows that higher traffic densities do not necessarily correlate with proportional increases in energy consumption, indicating efficient energy usage even in dense traffic scenarios.

These visualizations are pivotal in understanding the intricate dynamics within the traffic system. By examining the distribution and inter-variable relationships, we can derive actionable insights into how different factors such as speed, energy usage, and traffic density interact.

CODE:

PAIRPLOT SHOWING NUMERICAL DATA DISTRIBUTION WITH KDE PLOTS

```
[ ]  numericals = ["Speed", "Energy Consumption", "Traffic Density"]
     sns.pairplot(traffic, vars=numericals)
     plt.show()
```



CORRELATION ANALYSIS FOR KEY VARIABLES:

We utilized a correlation heatmap to quantitatively assess relationships between several key traffic variables: Hour of Day, Speed, Is Peak Hour, Random Event Occurred, Energy Consumption, and

Traffic Density. This analytical tool is instrumental in visualizing the strengths and directions of relationships among variables, offering crucial insights for strategic traffic management and planning.

Key Findings from the Heatmap:

Strong Positive Correlation:

The most notable relationship observed was between **Speed** and **Energy Consumption**, with a correlation coefficient of **0.84**. This strong positive correlation indicates that as vehicle speeds increase, energy usage also increases significantly, which has implications for both fuel consumption and emissions in urban settings.

Minor Correlations:

Speed and Traffic Density: Exhibited a slightly negative correlation of **-0.028**, suggesting that higher speeds tend to occur at lower traffic densities, which is typical in less congested conditions.

Is Peak Hour and Random Event Occurred: This pair showed a correlation of **0.025**, indicating a slight increase in the likelihood of random events during peak traffic hours, possibly due to increased vehicle interactions.

Implications for Traffic Management:

The insights derived from this heatmap are critical for formulating targeted interventions:

Speed Regulation:

Given the strong correlation between speed and energy consumption, speed regulation could be a vital strategy in energy management and environmental protection.

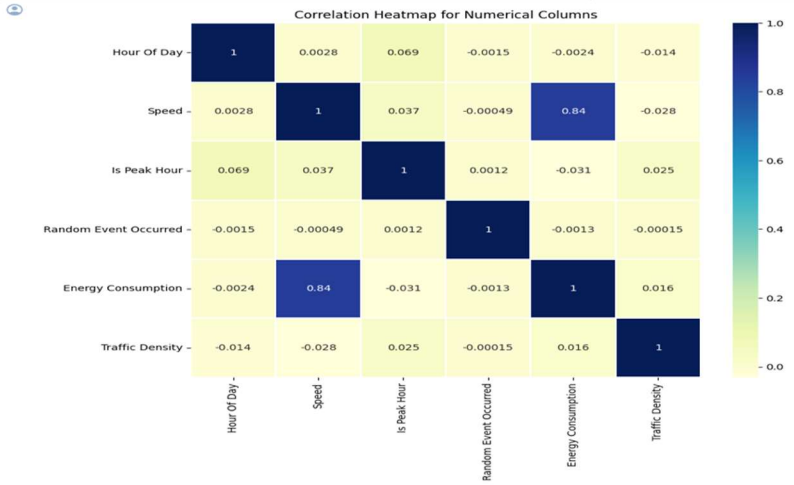
Peak Hour Management:

Understanding the slight increase in random events during peak hours could lead to enhanced safety measures or more efficient traffic routing during these times.

CODE:

CORRELATION MATRIX BETWEEN NUMERICAL VALUES

```
# Select numerical columns for analysis
numerical_columns = ['Hour Of Day', 'Speed', 'Is Peak Hour', 'Random Event Occurred', 'Energy Consumption', 'Traffic Density']
traffic_numeric = traffic[numerical_columns]

# Calculate correlation matrix
correlation_mat = traffic_numeric.corr()

# Plot correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_mat, linewidths=0.01, cmap="YlGnBu", annot=True)
plt.title("Correlation Heatmap for Numerical Columns")
plt.show()
```



DATA VISUALISATIONS:

Data visualization is the art of presenting data in a graphical or visual format, making complex information more accessible and understandable. By using charts, graphs, and other visual elements, it helps reveal patterns, trends, and insights that might otherwise go unnoticed in raw data.

Various data visualization methods utilized in the project include:

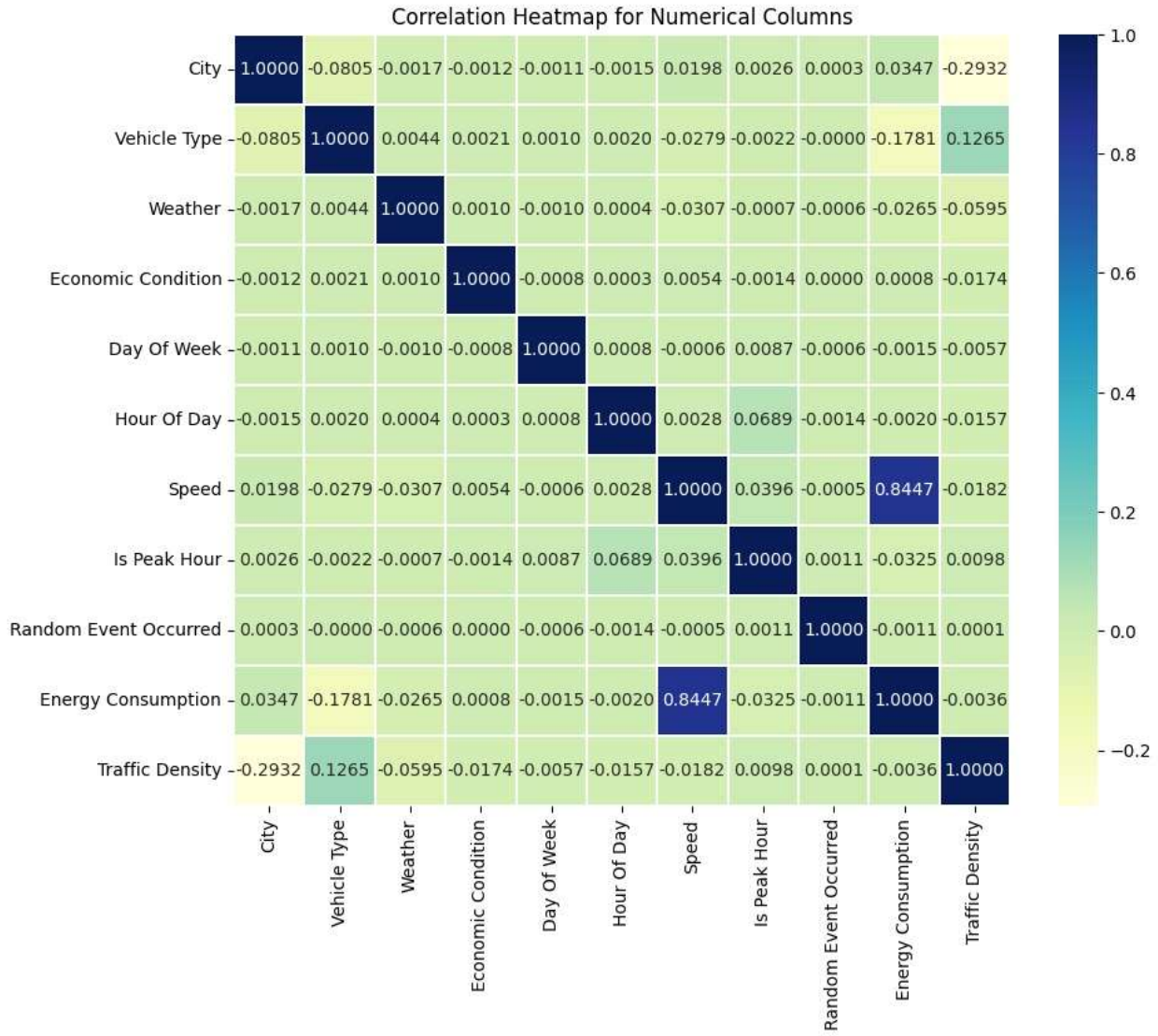
1. Correlation Heatmap

2. Scatter Plots

3. Bar Chart

1. CORRELATION HEATMAP:

A correlation heatmap displays the correlation coefficients between variables in a dataset using color gradients. It provides a visual representation of how strongly different variables are related to each other, with darker colors indicating higher correlations.



Correlation Matrix Calculation: The correlation matrix (`corr_matrix`) is computed using the `corr()` function of pandas DataFrame. This matrix illustrates the correlation coefficients between pairs of variables, indicating the strength and direction of the linear relationship between them.

Interpretation:

- The heatmap visually displays the correlations between different variables.

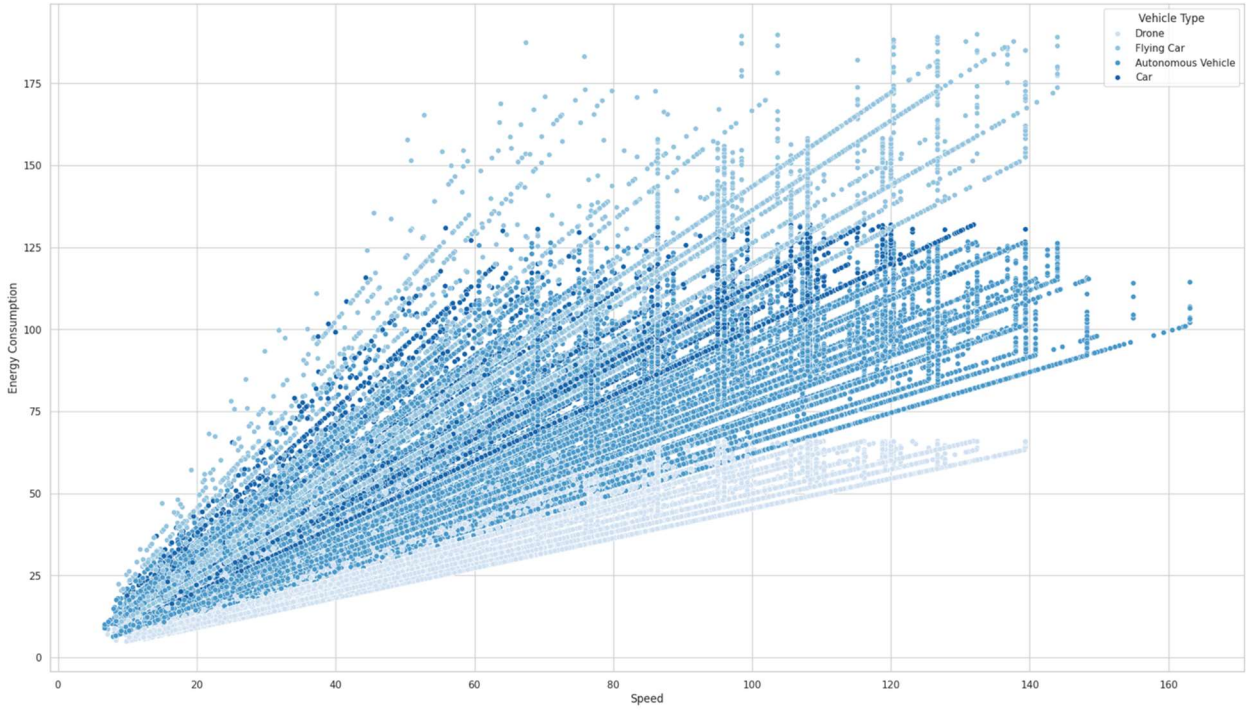
- Values near 1 signify a strong positive correlation (where one variable tends to increase as the other increases).

- Values near -1 denote a strong negative correlation (where one variable tends to decrease as the other increases).

- Values close to 0 suggest no correlation.

2. SCATTER PLOT:

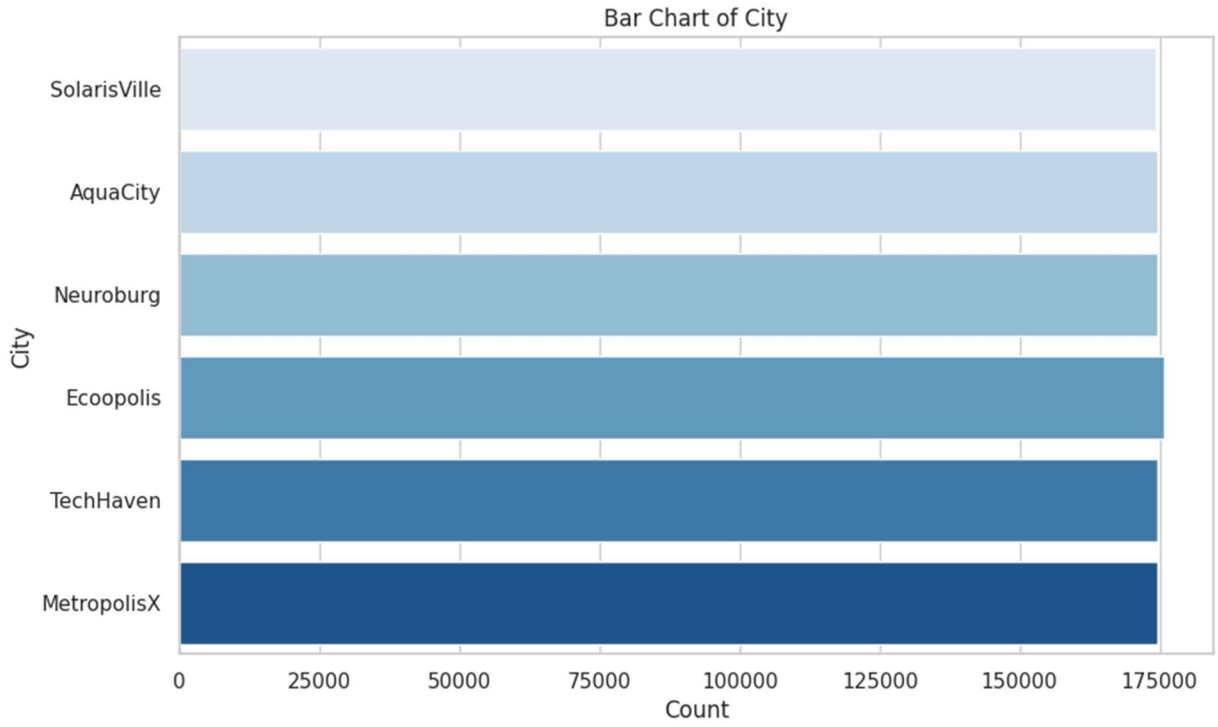
This creates a scatter plot depicting the relationship between vehicle speed and energy consumption, while categorizing data points by vehicle type using seaborn. The x-axis denotes speed, while the y-axis represents energy consumption, with each point indicating a specific data observation. The color variation distinguishes between different vehicle types. This visualization facilitates comprehension of how speed influences energy usage across various vehicle categories, assisting in analyzing energy efficiency and traffic trends within a futuristic urban setting.

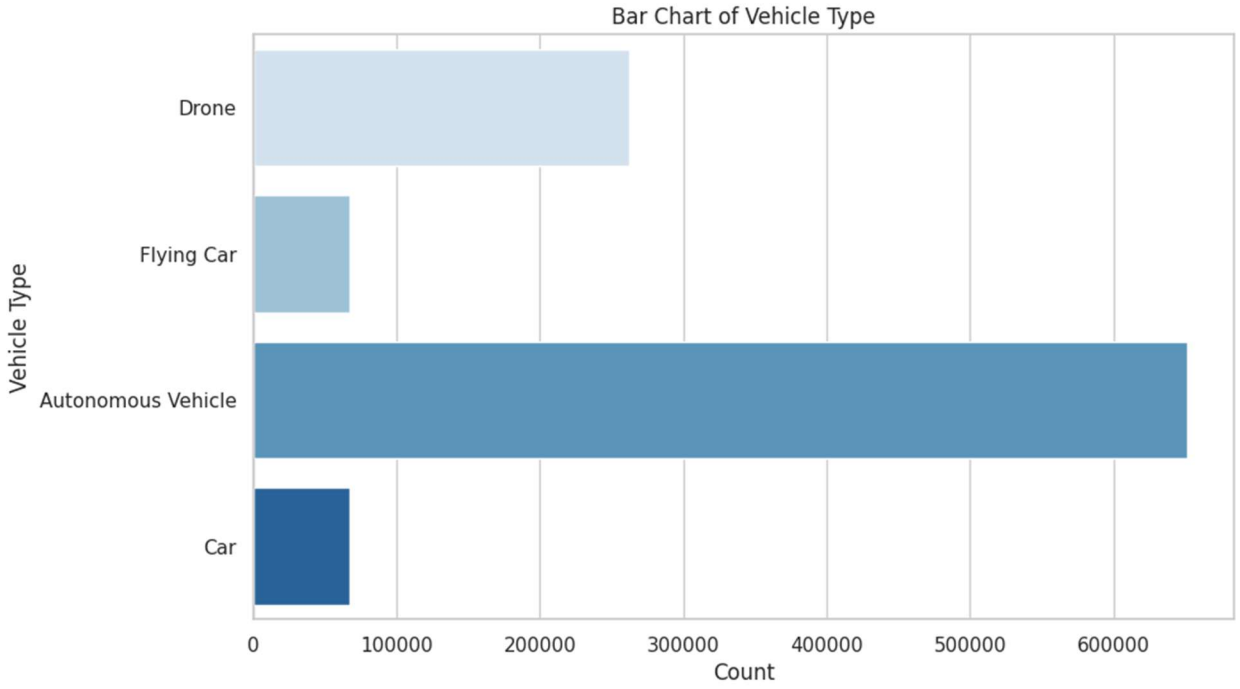


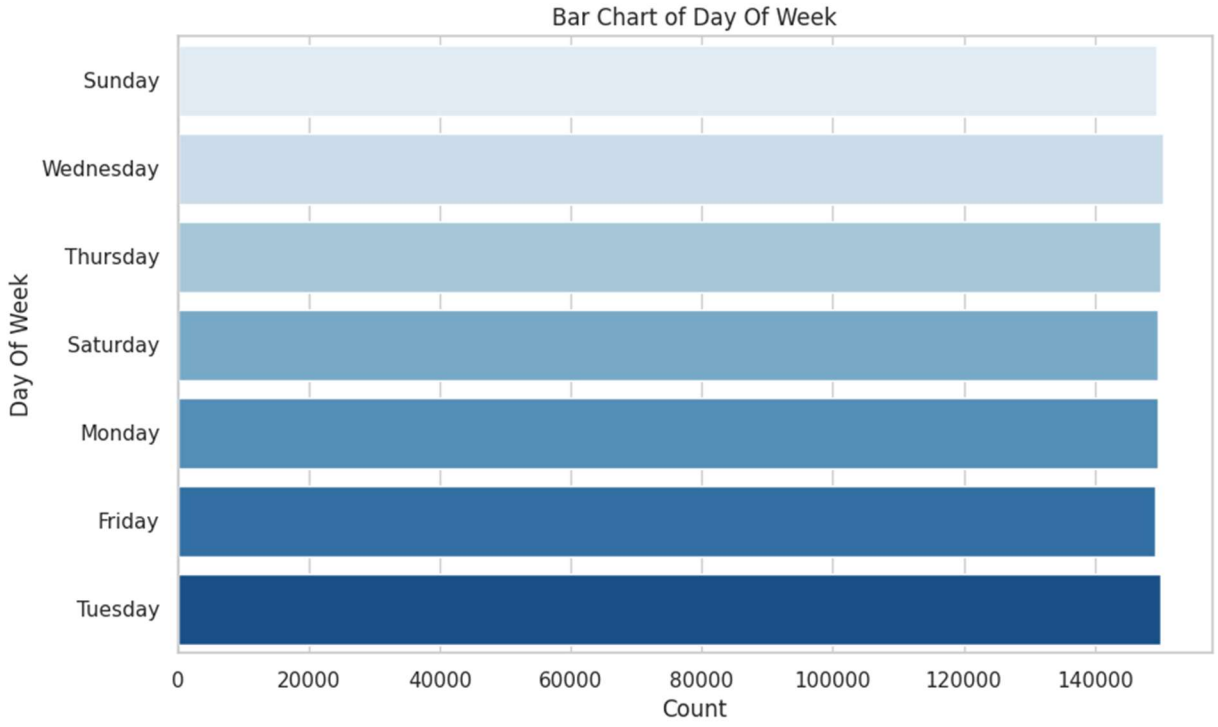
3. BAR CHART:

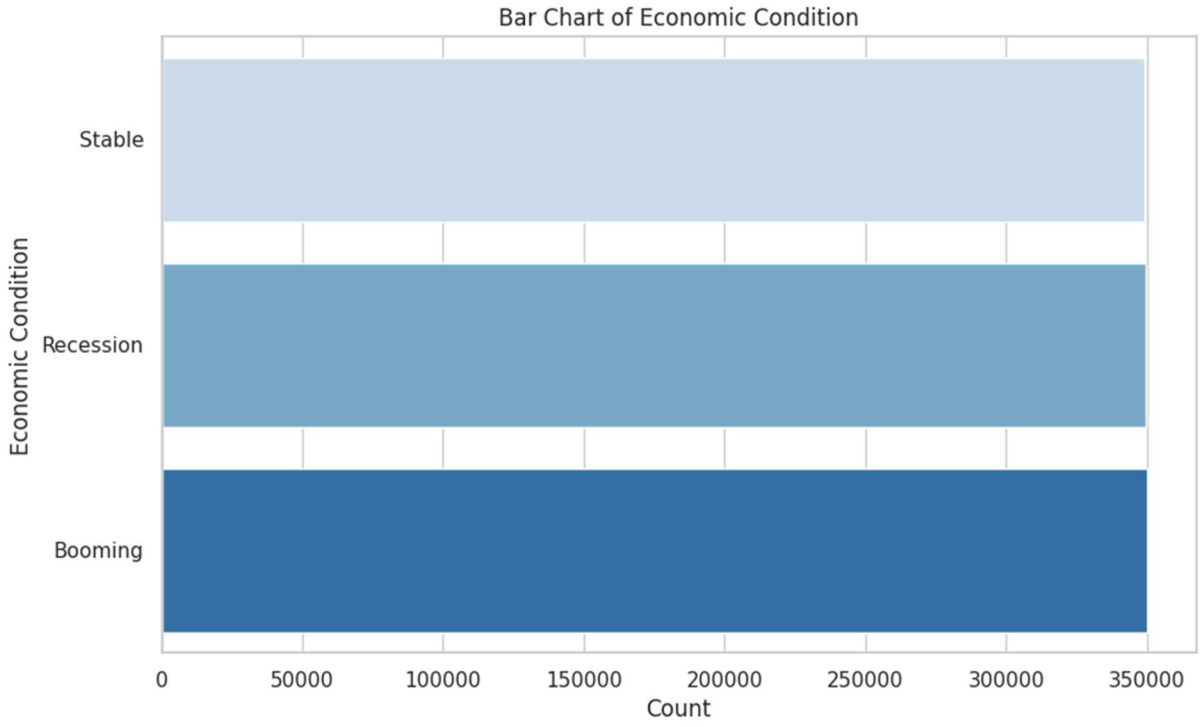
Interpreting bar charts involves grasping the correlation between the data and the visual aspects used. In a bar chart with numerical values and lines, the lines could indicate trends or benchmarks,

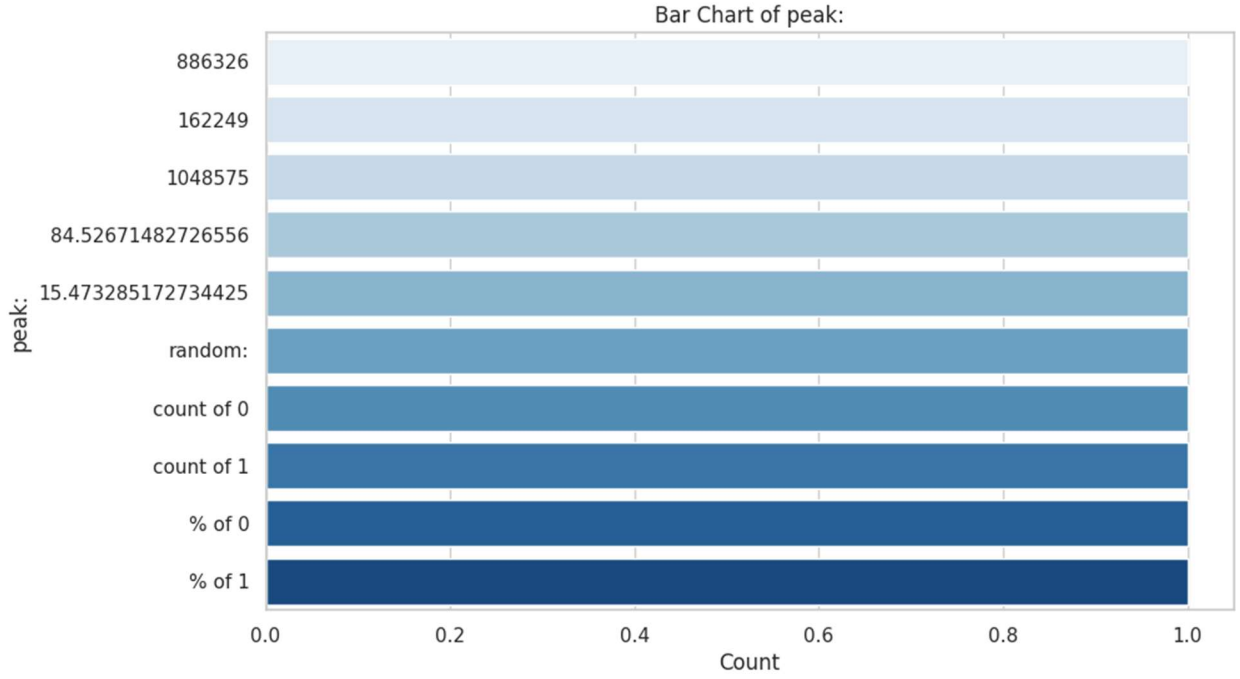
such as horizontal lines representing averages or thresholds. In a chart depicting vehicle types, each bar's height represents the frequency or quantity of each type, allowing for a simple comparison. A chart with numbers and varying shades of blue might signify different categories or levels within the data, as the intensity of the color changes. Similarly, in a chart with blue bars, the height of each bar directly reflects the numerical value it represents, facilitating easy quantity comparison. Overall, interpreting these charts entails analyzing data points, understanding context, and identifying any patterns or trends presented visually.




Bar Chart of Vehicle Type
Drone
Flying Car
Autonomous Vehicle
Car
Vehicle Type
0
100000
200000
300000
400000
500000
600000
Count


Bar Chart of Day Of Week
Sunday
Wednesday
Thursday
Saturday
Monday
Friday
Tuesday
Day Of Week
0
20000
40000
60000
80000
100000
120000
140000
Count


Bar Chart of Economic Condition
Economic Condition
Stable
Recession
Booming
0
50000
100000
150000
200000
250000
300000
350000
Count


Bar Chart of peak:
peak:
886326
162249
1048575
84.52671482726556
15.473285172734425
random:
count of 0
count of 1
% of 0
% of 1
0.0
0.2
0.4
0.6
0.8
1.0
Count

PREPROCESSING STEPS:

Handling Missing Values:

We implemented a rigorous data cleaning procedure to ensure the integrity of the traffic dataset. Initially, we conducted a comprehensive check for missing values across all columns using the **traffic.isna().sum()** function. This preliminary analysis confirmed that our dataset was devoid of missing entries, validating the robustness of the data collection process.

CODE:

```
[ ]  traffic.isna().sum()

     City                        0
     Vehicle Type                0
     Weather                     0
     Economic Condition          0
     Day Of Week                 0
     Hour Of Day                 0
     Speed                       0
     Is Peak Hour                0
     Random Event Occurred       0
     Energy Consumption          0
     Traffic Density             0
     dtype: int64
```

In our analysis of the traffic data from various futuristic cities, a critical preprocessing step was to ensure the dataset's completeness by removing any records with missing values. Initially, we recorded the total number of rows in the dataset (rows_before) and then applied the traffic.dropna(inplace=True) method to eliminate any rows containing missing data directly from the DataFrame. This method modifies the DataFrame in place, ensuring the operation is memory-efficient and that changes are immediately reflected in the DataFrame.

After the data cleaning operation, we reassessed the number of rows (rows_after), confirming that our initial data integrity was impeccable since the number of rows remained unchanged. This result—zero rows removed (rows_removed)—indicated that there were no missing values in any records. This cleanliness in data is crucial as it allows for accurate and reliable traffic analysis, avoiding potential biases or errors that missing data could introduce.

Moreover, the integrity of this dataset was further highlighted by the data shown in the DataFrame post-cleaning, which provided a comprehensive view of the variety and depth in the dataset. This includes multiple entries spanning various cities and vehicle types under different weather conditions and economic statuses, from "Snowy" and "Solar Flare" conditions to "Recession" and "Booming" economic conditions.Such robustness in the dataset sets a solid foundation for subsequent analyses, including predictive modeling and trend analysis in urban traffic patterns. It enables us to explore dependencies on socio-economic factors and city infrastructure with greater accuracy. These attributes are particularly important in our project, as they ensure that all further statistical analyses, visualizations, or modeling efforts are based on complete and reliable data.

CODE:

```
rows_before = len(traffic)
traffic.dropna(inplace=True)
rows_after = len(traffic)
rows_removed = rows_before - rows_after
print(rows_removed)
print(traffic)
```

```
0
                 City         Vehicle Type      Weather Economic Condition  \
0         SolarisVille                Drone        Snowy              Stable
1             AquaCity          Flying Car  Solar Flare           Recession
2            Neuroburg  Autonomous Vehicle  Solar Flare           Recession
3            Ecoopolis                Drone        Clear             Booming
4              AquaCity  Autonomous Vehicle  Solar Flare              Stable
...                 ...                 ...          ...                 ...
1048570    MetropolisX                 Car  Solar Flare             Booming
1048571   SolarisVille  Autonomous Vehicle  Solar Flare              Stable
1048572       AquaCity                Drone        Clear              Stable
1048573   SolarisVille                Drone  Solar Flare              Stable
1048574   SolarisVille                Drone        Clear           Recession

         Day Of Week  Hour Of Day     Speed  Is Peak Hour  \
0             Sunday           20   29.4268             0
1          Wednesday            2  118.8000             0
2          Wednesday           16  100.3904             0
3           Thursday            8   76.8000             1
4           Saturday           16   45.2176             0
...              ...          ...       ...           ...
1048570       Sunday           20   45.9472             0
1048571     Thursday           12  106.6345             0
1048572       Monday           17   86.4000             1
1048573     Saturday           19   72.2894             0
1048574     Thursday            8   95.7940             1

          Random Event Occurred  Energy Consumption  Traffic Density
0                             0             14.7134           0.5241
1                             0            143.5682           0.3208
2                             0             91.2640           0.0415
3                             0             46.0753           0.1811
4                             0             40.1934           0.4544
...                         ...                 ...              ...
1048570                       0             52.2127           0.4740
1048571                       0             85.3076           0.2005
1048572                       0             44.6384           0.6936
1048573                       0             36.1447           0.3715
1048574                       0             47.8970           0.0864

[1048575 rows x 11 columns]
```

OUTLIER DETECTION:

In our project on analyzing futuristic city traffic, we conducted a detailed examination of key numerical attributes—Speed, Energy Consumption, and Traffic Density—using box plots to provide a clear visual summary of their distributions. Here's a concise overview of the insights gained from each attribute based on the box plots:

Traffic Density:
The Traffic Density box plot revealed a tight distribution predominantly below 1.0, with a few outliers stretching up to 3.38. The minimum value was recorded at 0.01, indicating very low traffic density under certain conditions or in specific areas. This suggests that traffic congestion is generally not severe across most regions, except for occasional spikes.

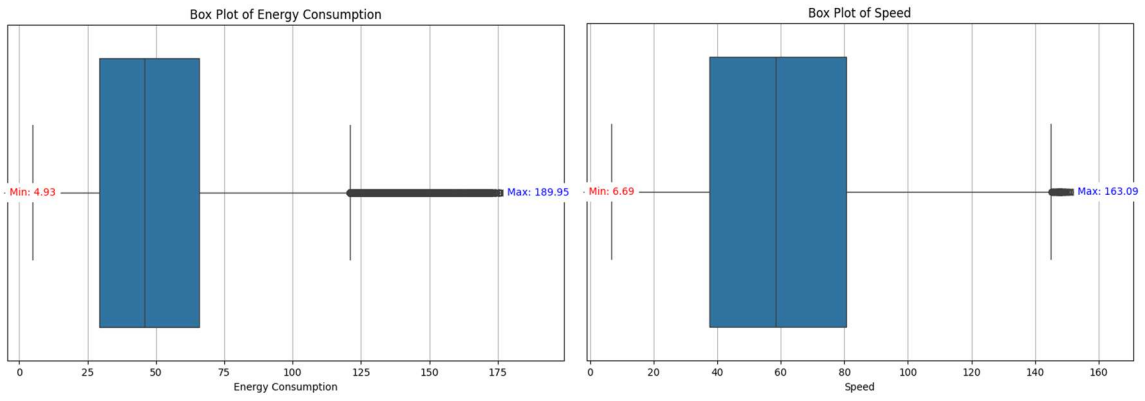
Energy Consumption:
Energy Consumption showed a broader range of values, spanning from 4.93 to 189.95. The bulk of data concentrated within a narrower middle range indicates consistent energy usage patterns for most vehicles or trips. The presence of higher outliers' points to instances of increased consumption, possibly linked to longer distances or less efficient vehicle types.

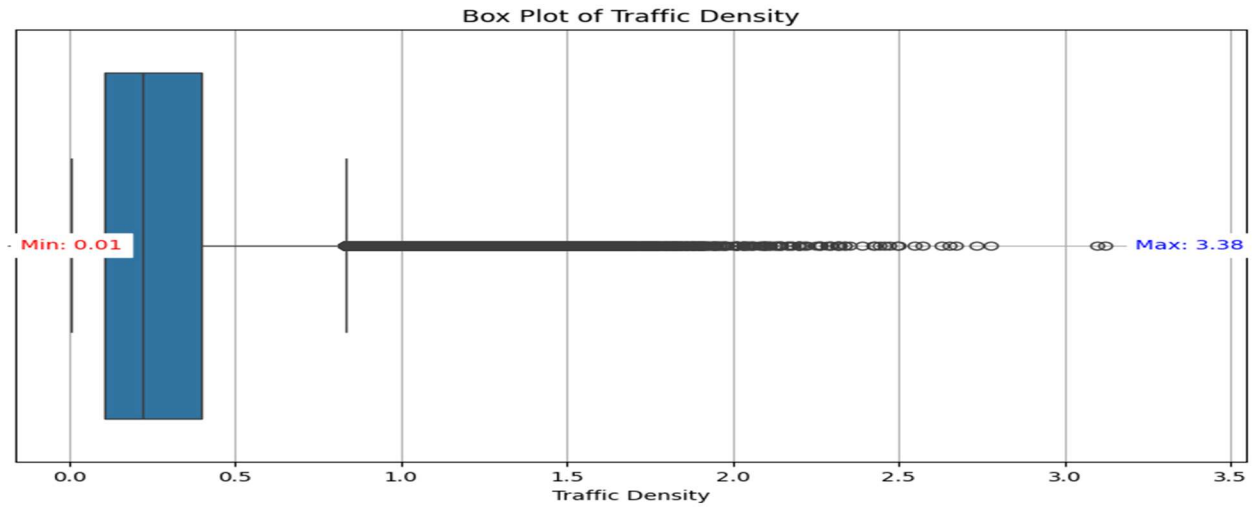
Speed:
The distribution of Speed, ranging from 6.69 to 163.09, highlights a significant variance in how fast vehicles travel across different city zones or times. The compactness of the main box suggests a common speed range adhered to by most traffic, with outliers indicating exceptionally high speeds, likely on less congested roads or designated fast lanes.

Each box plot was carefully annotated with minimum and maximum values, enhancing the interpretability of the visual data. These plots are integral to understanding the dynamics of traffic flow, energy efficiency, and speed regulation in futuristic urban settings, providing a foundation for traffic management strategies and policy-making aimed at optimizing city transportation networks

CODE:

```
# Assuming 'traffic' is your DataFrame
numerical_columns = ['Speed', 'Energy Consumption', 'Traffic Density']

# Plotting box plots for each numerical column and annotating them
for col in numerical_columns:
    plt.figure(figsize=(10, 6))  # Setting the figure size
    sns.boxplot(x=traffic[col])  # Creating a box plot
    plt.title(f'Box Plot of {col}')  # Adding a title to the box plot
    plt.grid(True)  # Adding a grid for better readability

    # Annotate min and max values on the plot
    min_val = traffic[col].min()
    max_val = traffic[col].max()
    plt.text(x=min_val, y=0, s=f"Min: {min_val:.2f}", color='red', va='center', ha='center', backgroundcolor='white')
    plt.text(x=max_val, y=0, s=f"Max: {max_val:.2f}", color='blue', va='center', ha='center', backgroundcolor='white')

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





In our analysis of futuristic city traffic data, a detailed examination of outliers was conducted to identify atypical values across key numerical variables: Speed, Energy Consumption, and Traffic Density. This was achieved using the Interquartile Range (IQR) method, a robust statistical technique that helps in detecting and managing outliers, thus ensuring the reliability of our subsequent analyses.

Methodology:

- **IQR Calculation**: For each numerical variable, the IQR was determined by subtracting the first quartile (Q1) from the third quartile (Q3). This range captures the middle 50% of values in each distribution.
- **Outlier Detection**: Outliers were defined as values that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR. These thresholds are commonly used in statistical practices to flag data points that are unusually high or low.

Findings:

Speed:
The analysis flagged 108 instances as outliers, primarily featuring high-speed values that significantly exceed typical traffic speeds. These instances were concentrated in vehicles identified as autonomous, operating under various weather and economic conditions. High speeds could indicate emergency scenarios, testing of vehicle capabilities, or anomalies in data collection.

Energy Consumption:

Many outliers (7,747) were identified in energy consumption, indicating unusually high or low energy usage. These could be influenced by vehicle type, operational mode, or external conditions like weather. High-energy consumption outliers may highlight inefficiencies or extreme operational conditions, such as high speeds or heavy traffic.

Traffic Density:
Traffic density analysis revealed 23,591 outliers, suggesting significant fluctuations in vehicle count per unit area across different times and conditions. High-density outliers are particularly valuable for urban planning and traffic management, indicating potential congestion points or popular routes.

Implications:

This outlier detection provides critical insights into the operational dynamics within the dataset. Understanding where and why outliers occur can help in refining traffic models, improving energy efficiency in transportation, and enhancing traffic flow management. It also aids in verifying data integrity and identifying potential errors or extraordinary events within the traffic system.

By systematically identifying and analyzing these outliers, we can better prepare for unusual conditions, optimize traffic management strategies, and ensure the sustainability of urban transportation systems in futuristic cities.

CODE:

```
Outliers in Speed:
             City         Vehicle Type       Weather Economic Condition  \
6503     Neuroburg  Autonomous Vehicle         Clear              Stable
15755    Neuroburg  Autonomous Vehicle  Solar Flare              Stable
21629    Neuroburg  Autonomous Vehicle  Solar Flare           Recession
42059    Neuroburg  Autonomous Vehicle  Solar Flare             Booming
46855    Neuroburg  Autonomous Vehicle  Solar Flare           Recession
...            ...                ...          ...                ...
1020461  Neuroburg  Autonomous Vehicle  Solar Flare           Recession
1030688  Neuroburg  Autonomous Vehicle         Clear             Booming
1042057  Neuroburg  Autonomous Vehicle  Solar Flare              Stable
1043529  Neuroburg  Autonomous Vehicle  Solar Flare           Recession
1046443  Neuroburg  Autonomous Vehicle         Clear           Recession

         Day Of Week  Hour Of Day     Speed  Is Peak Hour  \
6503        Thursday           18  148.2624             1
15755         Friday           18  148.2624             1
21629        Tuesday           18  163.0886             1
42059        Tuesday           18  145.9733             1
46855       Thursday           18  147.6359             1
...              ...          ...       ...           ...
1020461       Friday           18  160.5561             1
1030688     Thursday           18  146.6396             1
1042057     Thursday           18  148.2624             1
1043529      Tuesday           18  148.2279             1
1046443    Wednesday           20  148.4982             0

         Random Event Occurred  Energy Consumption  Traffic Density
6503                         0             94.5682           0.1109
15755                        0             94.6999           0.1045
21629                        0            106.8873           0.0478
42059                        0             90.7371           0.0709
46855                        0             91.7706           0.0228
...                        ...                 ...              ...
1020461                      0             99.8018           0.0417
1030688                      0             91.1513           0.0729
1042057                      0            110.7858           0.0876
1043529                      0             92.1386           0.0376
1046443                      0            115.3833           0.0249

[108 rows x 11 columns]
```

Outliers in Energy Consumption:

	City	Vehicle Type	Weather	\
1	AquaCity	Flying Car	Solar Flare	
24	MetropolisX	Flying Car	Clear	
88	MetropolisX	Flying Car	Snowy	
141	MetropolisX	Flying Car	Electromagnetic Storm	
255	MetropolisX	Autonomous Vehicle	Clear	
...	...	...	...	
1047951	AquaCity	Flying Car	Electromagnetic Storm	
1048155	SolarisVille	Flying Car	Electromagnetic Storm	
1048167	MetropolisX	Flying Car	Solar Flare	
1048168	MetropolisX	Autonomous Vehicle	Clear	
1048242	AquaCity	Flying Car	Electromagnetic Storm	

	Economic Condition	Day Of Week	Hour Of Day	Speed	Is Peak Hour	\
1	Recession	Wednesday	2	118.8000	0	
24	Booming	Friday	12	114.4823	0	
88	Booming	Monday	11	97.8597	0	
141	Recession	Thursday	8	132.4224	1	
255	Booming	Thursday	15	103.6800	0	
...	...	...	...	...	...	
1047951	Stable	Sunday	20	103.0907	0	
1048155	Recession	Tuesday	9	102.7687	0	
1048167	Stable	Tuesday	1	121.4362	0	
1048168	Booming	Sunday	16	126.7200	0	
1048242	Booming	Wednesday	15	108.0000	0	

	Random Event Occurred	Energy Consumption	Traffic Density
1	0	143.5682	0.3208
24	0	156.1122	0.4281
88	0	140.4684	0.4268
141	0	165.8871	0.5267
255	0	122.9405	1.1955
...	...	...	...
1047951	0	123.7088	0.4087
1048155	0	123.3224	0.1417
1048167	0	165.5948	0.3552
1048168	0	120.8929	0.6503
1048242	0	148.7636	0.8465

[7747 rows x 11 columns]

```
Outliers in Traffic Density:
                City         Vehicle Type                Weather  \
74         MetropolisX  Autonomous Vehicle                  Clear
75        SolarisVille                Drone                  Clear
76         MetropolisX  Autonomous Vehicle                  Clear
85         MetropolisX  Autonomous Vehicle                  Clear
184           AquaCity          Flying Car                  Snowy
...                ...                 ...                    ...
1048390       AquaCity                 Car  Electromagnetic Storm
1048406    MetropolisX  Autonomous Vehicle            Solar Flare
1048454       AquaCity  Autonomous Vehicle                  Rainy
1048492    MetropolisX               Drone                  Clear
1048550    MetropolisX               Drone            Solar Flare

         Economic Condition Day Of Week  Hour Of Day     Speed  Is Peak Hour  \
74                  Booming   Wednesday            10   65.7099             0
75                  Booming      Sunday             9   24.0919             0
76                  Booming      Sunday             9   17.6630             0
85                  Booming     Tuesday            17   76.4268             1
184                 Booming     Tuesday             2   38.2851             0
...                     ...         ...           ...       ...           ...
1048390             Booming      Monday             3  108.0000             0
1048406              Stable   Wednesday             4   30.2715             0
1048454             Booming   Wednesday            22   43.6974             0
1048492             Booming    Saturday            19   18.2426             0
1048550             Booming      Friday            14   21.0014             0

          Random Event Occurred  Energy Consumption  Traffic Density
74                            0             52.5679           0.8521
75                            0             12.0460           1.0363
76                            0             19.6255           0.9877
85                            0             67.9350           1.3003
184                           0             45.9422           0.8769
...                         ...                ...              ...
1048390                       0            108.3471           1.1035
1048406                       1             33.6350           0.9408
1048454                       0             34.9579           0.8472
1048492                       0             12.6685           1.0648
1048550                       0             14.5843           1.0511

[23591 rows x 11 columns]
```

In our project focusing on futuristic city traffic data, we undertook a rigorous data cleaning process to identify and remove outliers across key numerical variables: Speed, Energy Consumption, and Traffic Density. This procedure was essential for ensuring the integrity and robustness of our traffic analyses, particularly in predictive modeling and traffic pattern identification.

In our project focusing on futuristic city traffic data, we undertook a rigorous data cleaning process to identify and remove outliers across key numerical variables: Speed, Energy Consumption, and Traffic Density. This procedure was essential for ensuring the integrity and robustness of our traffic analyses, particularly in predictive modeling and traffic pattern identification.

Results:

Dataframe Reduction:

Original DataFrame:

Initially, the dataset comprised 1,048,575 rows, encompassing all recorded traffic data.

Filtered DataFrame:

After the removal of outliers, the dataset was reduced to 1,017,798 rows.

Outliers Removed: 30,777 rows were identified as outliers and excluded from the dataset.

Implications:

This focused approach to outlier management enhances the reliability of statistical analyses and ensures that subsequent traffic models are not adversely affected by anomalous data. By filtering out these outliers, we maintain a high level of data quality, crucial for accurate forecasting and planning in urban traffic management systems.

The removal of over 30,000 outliers significantly purifies the dataset, potentially leading to more accurate insights into traffic flow patterns, energy utilization, and vehicular behavior under typical conditions. This cleaner dataset is pivotal for developing efficient, data-driven traffic solutions and strategies in futuristic urban environments.

CODE:

```
import pandas as pd

# Assuming 'traffic' is your DataFrame
numerical_columns = ['Speed', 'Energy Consumption', 'Traffic Density']

# Function to detect outliers using IQR
def detect_outliers_iqr(traffic, column):
    Q1 = traffic[column].quantile(0.25)
    Q3 = traffic[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Return the indices of rows that are not outliers
    non_outliers = traffic[(traffic[column] >= lower_bound) & (traffic[column] <= upper_bound)]
    return non_outliers.index

# Find indices of rows that are non-outliers in all numerical columns
non_outlier_indices = set(traffic.index)  # Start with all indices
for col in numerical_columns:
    col_non_outliers = detect_outliers_iqr(traffic, col)
    non_outlier_indices = non_outlier_indices.intersection(col_non_outliers)

# Convert set to list before filtering
non_outlier_indices_list = list(non_outlier_indices)

# Filter the DataFrame to only include non-outliers
filtered_traffic = traffic.loc[non_outlier_indices_list]

# Printing the shape of the dataframes to compare
print(f"Original DataFrame shape: {traffic.shape}")
print(f"Filtered DataFrame shape without outliers: {filtered_traffic.shape}")
print(f"Number of removed outliers: {traffic.shape[0] - filtered_traffic.shape[0]}")
```

```
Original DataFrame shape: (1048575, 11)
Filtered DataFrame shape without outliers: (1017798, 11)
Number of removed outliers: 30777
```

FEATURE ENGINEERING:

LABEL ENCODING:

An important step involved in preparing the dataset for advanced modeling was the application of label encoding to several categorical variables. Using the **LabelEncoder** from the sklearn.preprocessing module, key categorical features such as 'City', 'Vehicle Type', 'Weather', 'Economic Condition', 'Day of Week', and 'Hour of Day' were transformed from textual to numerical formats, which are more amenable to the algorithms we plan to use in subsequent analyses.

This encoding process ensures that our machine learning models can interpret the input data correctly by converting the categorical text data into a series of integers. Each unique string value in the columns was assigned a unique integer, making the data uniform and standardized for processing. For instance, different cities are represented as integers like 0, 1, 2, etc., instead of their names.

The transformed dataset, now stored in **encoded_traffic**, retains the original structure with the categorical values replaced by their encoded counterparts, as visible in the output of the **.head()** method, which displays the first few rows of the encoded DataFrame. This modification is crucial for feeding this data into statistical models and machine learning algorithms, which typically require numerical input values to perform effectively. The successful encoding of these key variables is a significant advancement in our data preparation stage, setting a strong foundation for robust and reliable traffic pattern analysis and predictions.

CODE:

LABEL ENCODING

```
[ ] from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    cats = ["City", "Vehicle Type", "Weather", "Economic Condition", "Day Of Week", "Hour Of Day",
            "Is Peak Hour", "Random Event Occurred"]
    encoded_traffic = filtered_traffic.copy()
    for i in cats[:-3]:
        encoded_traffic[i] = le.fit_transform(encoded_traffic[i])
```

```
[ ] encoded_traffic.head()
```

	City	Vehicle Type	Weather	Economic Condition	Day Of Week	Hour Of Day	Speed	Is Peak Hour	Random Event Occurred	Energy Consumption	Traffic Density
0	4	2	3	2	3	20	29.4268	0	0	14.7134	0.5241
2	3	0	4	1	6	16	100.3904	0	0	91.2640	0.0415
3	1	2	0	0	4	8	76.8000	1	0	46.0753	0.1811
4	0	0	4	2	2	16	45.2176	0	0	40.1934	0.4544
5	5	0	3	1	4	20	30.5179	0	0	37.5562	0.0843

Analysis of Correlation Among Numerical Variables in Traffic Data:

In our ongoing project analyzing traffic data for futuristic cities, we conducted a correlation analysis to understand the relationships between various numerical variables in the dataset. This analysis was performed using a correlation matrix, which was then visualized through a heatmap to provide a clear and intuitive representation of how different variables are interrelated.

The heatmap illustrates the correlation coefficients between pairs of variables, with values ranging from -1 to 1. A value close to 1 indicates a strong positive correlation, meaning that as one variable increases, the other tends to increase as well. Conversely, a value close to -1 indicates a strong negative correlation, where one variable increases as the other decreases. Values near zero suggest no linear relationship.

Key observations from the heatmap include:

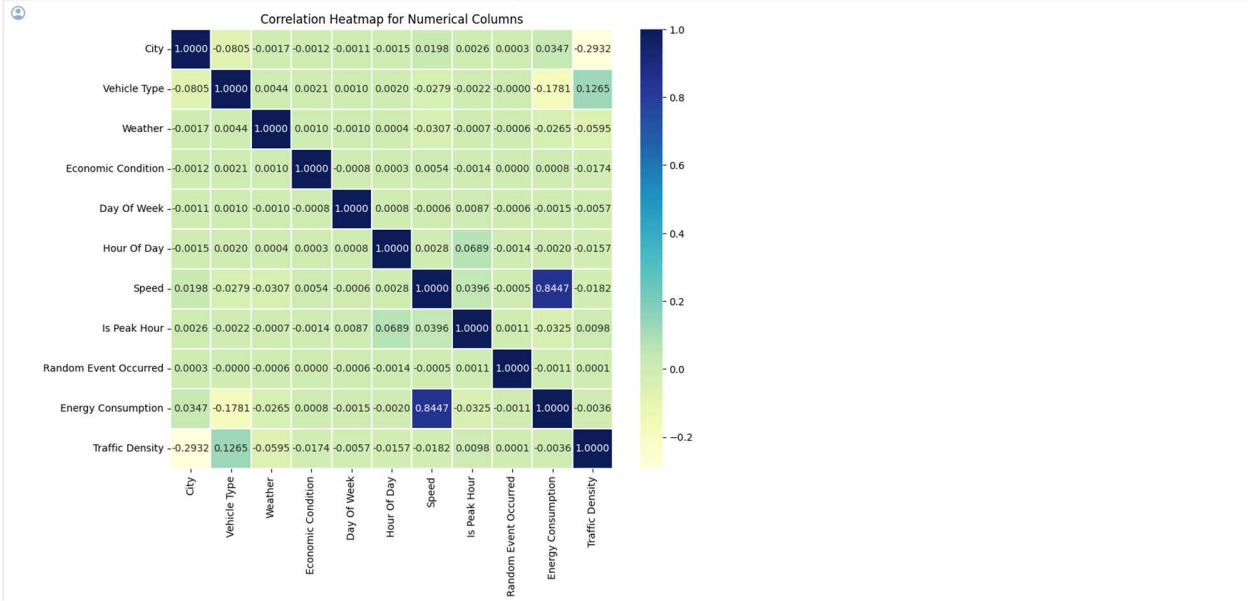
- **High Positive Correlation**: The variables 'Speed' and 'Energy Consumption' exhibited a notably high correlation, suggesting that higher speeds tend to coincide with increased energy usage, which is logical given the greater power requirements at higher velocities.

- **Negative Correlation**: 'Traffic Density' showed a significant negative correlation with 'City', indicating that specific cities might experience lower traffic densities, possibly due to better infrastructure or less population density.
- **Low Correlation**: Several variables such as 'Weather', 'Economic Condition', and 'Random Event Occurred' had very low correlations with other factors, implying that these aspects have minimal linear influence on other variables like speed or energy consumption in this dataset.

This correlation analysis is crucial for identifying factors that significantly impact traffic behaviors, which can further guide targeted data collection and more nuanced analyses. Understanding these relationships helps in building more accurate predictive models and in making informed decisions for urban planning and traffic management in futuristic settings.

CODE:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
# Select numerical columns for analysis
numerical_columns = ["City", "Vehicle Type", "Weather", "Economic Condition", "Day Of Week",'Hour Of Day', 'Speed', 'Is Peak Hour', 'Random Event Occurred', 'Energy Consumption', 'Traffic Density']
traffic_numeric = encoded_traffic[numerical_columns]

# Calculate correlation matrix
correlation_mat = traffic_numeric.corr()

# Plot correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_mat, linewidths=0.01, cmap="YlGnBu", annot=True, fmt=".4f")
plt.title("Correlation Heatmap for Numerical Columns")
plt.show()
```



STANDARD SCALAR:

We focused on standardizing the dataset to optimize performance for machine learning algorithms. Utilizing the **StandardScaler** from sklearn's preprocessing module, we transformed the numerical columns of our encoded traffic dataset. Standardization adjusts the distribution of data such that it has a mean of zero and a standard deviation of one,

which is crucial for many machine learning models that are sensitive to the scale of input features.

Copying the Encoded Dataset: We initiated by copying the encoded data to ensure that the original dataset remains unchanged for integrity and comparison purposes. Applying Standards Caler: The fit transform method of Standards Caler was applied to the entire dataset. This method computes the mean and standard deviation for each feature, and subsequently, scales the data accordingly.

Results and Implications:

Transformed Data Check: The output from the transformation shows the dataset with scaled features. Each value represents how many standard deviations away from the mean the original value was. For example, values close to 0 are near the mean, and values such as 1.7557167 indicate the feature value is approximately 1.76 standard deviations above the mean.

Data Type: The scaled data remains in a NumPy array format, which is ideal for input into machine learning models and facilitates efficient computations.

This standardization is instrumental for our analysis as it reduces bias towards features with larger scales and improves the algorithm's convergence during training. It ensures that each feature contributes equally to the prediction process, enhancing the model's fairness and accuracy. This step is vital in preparing our traffic dataset for sophisticated predictive modeling and analysis, ensuring that the forthcoming stages of machine learning can proceed with the most refined data possible.

CODE:

STANDARD SCALAR

```
[ ] from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    scaled_traffic = encoded_traffic.copy()

    # Fit and transform the numerical columns
    scaled_traffic = scaler.fit_transform(scaled_traffic)

    # Check the transformed data
    print(scaled_traffic)
    type(scaled_traffic)
```

```
[[ 0.86179643  1.24059926  0.70480446 ... -0.22947291 -1.40954465
   1.40435389]
 [ 0.27688127 -0.72155168  1.41376881 ... -0.22947291  1.75517167
  -1.15672588]
 [-0.89294906  1.24059926 -1.4220886  ... -0.22947291 -0.11299679
  -0.41589137]
 ...
 [-1.47786422  1.24059926 -1.4220886  ... -0.22947291 -0.17240039
   2.30386284]
 [ 0.86179643  1.24059926  1.41376881 ... -0.22947291 -0.52354265
   0.59453049]
 [ 0.86179643  1.24059926 -1.4220886  ... -0.22947291 -0.03768499
  -0.91844887]]
numpy.ndarray
```

STANDARDISING DATA INTO DATAFRAME:

Conversion Process:

DataFrame Conversion: The standardized data, initially in a NumPy array format following scaling, was converted into a pandas DataFrame. This conversion was done to ensure that the data could be easily manipulated and accessed using DataFrame operations, which are more intuitive and offer extensive functionalities for data analysis.

Column Assignment: The columns of the new DataFrame were named after the original dataset to preserve context and meaning, ensuring that each scaled value could be directly associated with its respective feature.

Analysis of Transformed Data:

- The **head ()** function provided a snapshot of the newly transformed DataFrame, showcasing the first few rows of scaled values. Each entry in the DataFrame represents a z-score, indicating how many standard deviations a value is from the feature's mean.For instance:
    - The 'City' feature values such as 0.861796 and -0.892949 indicate how these values relate to the average city index within the context of the dataset.
    - 'Speed' shows a scaled value of 1.540892 for one entry, suggesting this speed is significantly above the average.
    - 'Energy Consumption' with a value like 1.755172 highlights higher than average energy usage.
    - 'Traffic Density' values range from positive to negative, such as 1.404354 and -1.156726, illustrating the variability in traffic conditions across different data points.

Converting the scaled data back into a DataFrame not only facilitates easier data handling but also prepares the dataset for advanced analytical procedures such as machine learning modeling.

CODE:

```
[ ] import pandas as pd
    from sklearn.preprocessing import StandardScaler

    # Assuming scaled_traffic is your scaled NumPy array
    scaled_traffic = scaler.fit_transform(encoded_traffic)

    # Convert the scaled array to a DataFrame
    scaled_traffic_df = pd.DataFrame(scaled_traffic, columns=encoded_traffic.columns)

    # Check the transformed DataFrame
    #print(scaled_traffic_df)
    scaled_traffic_df.head()
```

	City	Vehicle Type	Weather	Economic Condition	Day Of Week	Hour Of Day	Speed	Is Peak Hour	Random Event Occurred	Energy Consumption	Traffic Density
0	0.861796	1.240599	0.704804	1.232784	-0.003877	1.227082	-1.148663	-0.425512	-0.229473	-1.409545	1.404354
1	0.276881	-0.721552	1.413769	0.001071	1.495590	0.648998	1.540892	-0.425512	-0.229473	1.755172	-1.156726
2	-0.892949	1.240599	-1.422089	-1.230642	0.495945	-0.507170	0.646805	2.350108	-0.229473	-0.112997	-0.415891
3	-1.477864	-0.721552	1.413769	1.232784	-0.503700	0.648998	-0.550184	-0.425512	-0.229473	-0.356163	1.034467
4	1.446712	-0.721552	0.704804	0.001071	0.495945	1.227082	-1.107310	-0.425512	-0.229473	-0.465189	-0.929593

PRINCIPAL COMPONENT ANALYSIS (PCA):

We implemented Principal Component Analysis (PCA) to perform dimensionality reduction on the scaled traffic dataset. PCA is a statistical technique that transforms a set of possibly correlated variables into a smaller number of uncorrelated variables called principal components. This method is particularly beneficial in reducing the complexity of data, enhancing interpretability while retaining most of the information.

Implementation and Analysis:

- **PCA Application**: Using the PCA module from sklearn.decomposition, we applied PCA to the dataset after standardizing the features. The goal was to capture the most significant patterns and relationships in the data with fewer variables, reducing computational costs for future modeling and eliminating potential multicollinearity issues among features.
- **Component Loadings**: The loadings for each principal component were examined to understand how strongly each original feature influences the component. The PCA component loadings indicate each variable's contribution to the direction of the corresponding principal component. For instance, the first few components showed significant loadings from features like 'Speed', 'Energy Consumption', and 'Traffic Density', which suggest these variables are major drivers of variance within the dataset.

- **Interpretation of Results**:
    - **Component Breakdown**: Each principal component represents a direction in feature space along which the dataset varies the most. The first principal component (PC1), for example, may capture most of the variance related to speed and energy usage patterns in the traffic data.
    - **Insights and Usage**: The analysis of these components provides insights into underlying patterns that may not be immediately obvious from direct observation of the data. For example, the relationship between 'Hour of Day' and 'Energy

Consumption' in a principal component suggests a time-related pattern in energy usage across the city.

The PCA provided a transformed dataset with reduced dimensions, which is now prepared for more efficient and effective data processing and modeling. By focusing on the components that explain the most variance, we ensure that our predictive models are both robust and computationally efficient

CODE:

PRINCIPLE COMPONENT ANALYSIS (PCA)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Example data setup
# Assuming X_scaled is your already scaled feature matrix
pca = PCA()
pca.fit(scaled_traffic)

# Get the PCA components (loadings)
components = pca.components_
# Display the loadings as a DataFrame for better readability
feature_names = encoded_traffic.columns  # make sure X.columns contains the names of your features
loadings_df = pd.DataFrame(components, columns=feature_names, index=[f'PC{i+1}' for i in range(len(components))])

top_features_per_component = {}
for component in loadings_df.columns:
    top_features = loadings_df[component].abs().sort_values(ascending=False).head(5).index.tolist()
    top_features_per_component[component] = top_features

print("Top 5 features for each principal component:")
for component, features in top_features_per_component.items():
    print(f"{component}: {features}")

print("PCA Component Loadings:\n", loadings_df)
```

```
Top 5 features for each principal component:
City: ['PC10', 'PC2', 'PC8', 'PC4', 'PC7']
Vehicle Type: ['PC8', 'PC2', 'PC1', 'PC4', 'PC11']
Weather: ['PC7', 'PC4', 'PC10', 'PC5', 'PC2']
Economic Condition: ['PC4', 'PC7', 'PC6', 'PC5', 'PC8']
Day Of Week: ['PC6', 'PC5', 'PC4', 'PC7', 'PC3']
Hour Of Day: ['PC9', 'PC3', 'PC6', 'PC5', 'PC10']
Speed: ['PC11', 'PC1', 'PC8', 'PC2', 'PC4']
Is Peak Hour: ['PC3', 'PC9', 'PC11', 'PC10', 'PC8']
Random Event Occurred: ['PC5', 'PC6', 'PC7', 'PC4', 'PC9']
Energy Consumption: ['PC11', 'PC1', 'PC2', 'PC7', 'PC9']
Traffic Density: ['PC10', 'PC2', 'PC8', 'PC1', 'PC4']
PCA Component Loadings:
          City  Vehicle Type   Weather  Economic Condition  Day Of Week  \
PC1   0.082316     -0.180962 -0.041481            0.005563    -0.001442
PC2  -0.628750      0.333049 -0.126902           -0.027828    -0.008064
PC3   0.002840      0.040710  0.003941           -0.005701     0.095890
PC4   0.098991     -0.163428 -0.618679           -0.706751     0.240622
PC5   0.021332     -0.022150 -0.131229            0.084658    -0.531971
PC6  -0.011589      0.044042 -0.005404            0.351878     0.794614
PC7  -0.088473      0.070207  0.739972           -0.599793     0.107746
PC8   0.370356      0.888624 -0.115667           -0.079929    -0.018161
PC9   0.050061     -0.022493  0.031148            0.022866    -0.078630
PC10  0.663198     -0.118745  0.142133            0.044820     0.017649
PC11 -0.013083      0.127854 -0.006012            0.002775    -0.000264

      Hour Of Day     Speed  Is Peak Hour  Random Event Occurred  \
PC1      0.001644  0.682851      0.005057              -0.001221
PC2     -0.022449  0.131951      0.014624              -0.000427
PC3      0.693177  0.035078      0.711512              -0.002985
PC4     -0.046422 -0.055916      0.022313               0.122710
PC5      0.052060 -0.008414      0.026454               0.829451
PC6     -0.103153  0.010834     -0.004944               0.481316
PC7     -0.022616  0.042360     -0.004748               0.254031
PC8     -0.034473  0.166565     -0.026729              -0.001401
PC9     -0.706649  0.021877      0.697885              -0.027262
PC10     0.048510 -0.053670     -0.031292              -0.000087
PC11    -0.001811 -0.691997      0.059410               0.000456

      Energy Consumption  Traffic Density
PC1             0.698433        -0.067816
PC2             0.089189         0.671360
PC3            -0.031453        -0.010914
PC4            -0.025636         0.042869
PC5            -0.007714         0.007678
PC6             0.000383        -0.014810
PC7             0.035238        -0.014880
PC8             0.002880        -0.153065
PC9            -0.032610         0.025117
PC10            0.021583         0.718935
PC11            0.706931        -0.035965
```

PREDICTION MODELS AND FINDINGS:

Prediction models are defined as computational algorithm techniques that use past data to predict future outcomes or events. Also, they tend to analyze existing relationships between the patterns of data for better decision-making. During predictive modeling process, we improve data quality, or enhance model performance during the predictive modeling process.

We preferred four prediction models like

Linear Regression,

Random Forest Regression,

Gradient Boost Regression, and

AdaBoost Regression.

1. **<u>Linear Regression</u>**:

   We know the fact that linear regression will fit a linear equation to the data that tells the relationship between the variables.

   This is the general form of a regression model:

   $$y=\beta 0+\beta 1 x1+\beta 2 x2+\ldots+\beta n xn+\epsilon$$

   where:

- y is the dependent variable (target).
- x is the independent variable (predictor).
- $\beta 0$ is the intercept (the value of y when predictors are 0).
- $\beta 1, \beta 2, \ldots, \beta n$ are the coefficients (slopes) associated with each predictor.
- ϵ represents the error term (residuals), which captures the difference between the actual and predicted values.

2. **<u>Random Forest Regression</u>**:

   To increase prediction accuracy and robustness, Random Forest Regression is an ensemble learning technique that integrates the predictions of several decision tree regressors. It works especially well for managing nonlinear relationships and identifying intricate patterns in data. Because of its ensemble structure and capacity to manage intricate data interactions, Random Forest Regression frequently yields high prediction accuracy.

   It is appropriate for datasets from the real world since it is resistant to noisy and overfitting data. The model helps with feature selection and data pattern analysis by offering insights into feature relevance. We experimented with different hyperparameter values to determine the best configuration for Random Forest, which may be utilized for a variety of regression problems.

3. **Gradient Boost Regression**:

   Gradient Boosting Another effective ensemble learning method is regression, which creates a sequential ensemble of weak learners—usually decision trees—to produce precise predictions in regression tasks. Gradient Boosting Regression can capture intricate nonlinear linkages and interactions in the data and tends to have a high prediction power. Overfitting is less likely to occur. Gradient Boosting yields feature significance scores that show how each feature affects the predictive performance of the model. Gradient Boosting Regression frequently produces predictions with high accuracy, especially when the hyperparameters are tuned properly.

4. **AdaBoost Regression**:

   AdaBoost Regression is a boosting technique that builds weak learners (usually decision trees) progressively to produce correct predictions in regression tasks. It is sometimes referred to as Adaptive Boosting Regression. AdaBoost's ensemble architecture improves prediction performance by lowering bias and variance. AdaBoost Regression frequently combines the strengths of several weak learners to obtain high accuracy. It can withstand noisy data and overfitting. It performs well with both numerical and categorical variables and is applicable to a wide range of regression situations. We tried varying hyperparameter values, like the number of weak learners (n_estimators), in our experiments. To assess model performance and determine the ideal model setup, we used cross-validation.

COMMON EVALUATION METRICS USED:

We measured the accuracy and magnitude of errors using various metrics that offer distinct viewpoints on the performance of regression models, such as MSE, RMSE, MAE, and MAPE. R-squared evaluates the overall goodness-of-fit of the model in relation to the variability in the data. To have a thorough grasp of the model's performance, it is typical to combine these measures.

Some key regression evaluation metrics:

- Mean squared error
- Mean absolute error
- Root Mean Squared error
- R squared

MEAN SQUARED ERROR:

The average squared difference (MSE) between the actual target values and the projected values is a measurement. The computation involves dividing the total number of data points by the sum

of squared differences. Because of squaring, MSE penalizes large errors significantly and is susceptible to outliers.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MEAN ABSOLUTE ERROR:

The average absolute difference (MAE) between the predicted and actual target values is calculated. Compared to MSE, it is less susceptible to outliers because it does not square the differences.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

ROOT MEAN SQUARED ERROR:

The average magnitude of the mistakes is given in the same units as the target variable by the root mean square error, or RMSE. Because it is simpler to understand than MSE, this metric is widely used, particularly in situations when the target variable's size is significant.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

R-SQUARED:

R-squared calculates the percentage of the target variable's variance that the model can account for. It has a range of 0 to 1, where 1 represents a perfect fit and the model's ability to explain all variance and 0 represents the model's inability to do so. R-squared can be understood as the model's goodness-of-fit.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

CODE:

When we use traffic density as target variable in Linear Regression:

MODEL EVALUATION

LINEAR REGRESSION USING TRAFFIC DENSITY AS TARGET VARIABLE

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import statsmodels.api as sm

# Assuming encoded_traffic contains your dataset

selected_features = ["City", "Vehicle Type", "Weather", "Economic Condition",
                     "Speed", "Day Of Week", "Hour Of Day", "Is Peak Hour",
                     "Random Event Occurred", "Energy Consumption", "Traffic Density"]

selected_scaled_traffic = scaled_traffic_df[selected_features]

scaler = StandardScaler()
selected_scaled_traffic = scaler.fit_transform(selected_scaled_traffic)

X = selected_scaled_traffic[:, :-1]
y = selected_scaled_traffic[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using various metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Perform cross-validation and calculate CV scores
cv_scores = cross_val_score(model, X, y, cv=10)
print("Cross-Validation Scores:")
print(cv_scores)  # Print individual CV scores
mean_cv_score = cv_scores.mean()

# Calculate p-values for the coefficients
X_with_intercept = sm.add_constant(X_train)  # Add intercept term for statsmodels
model_sm = sm.OLS(y_train, X_with_intercept).fit()
p_values = model_sm.pvalues

# Get the coefficients and create a DataFrame to display with p-values
coefficients = model_sm.params
coefficients_df = pd.DataFrame({"Coefficient": coefficients, "P-value": p_values})

# Display the evaluation metrics, coefficients, and p-values
print(f"\nR-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"Mean CV Score: {mean_cv_score:.4f}")
print("\nCoefficients and P-values:")
print(coefficients_df)

```

```
Cross-Validation Scores:
[0.10567354 0.10270628 0.10681576 0.10474209 0.1053143  0.10808033
 0.10591111 0.10175439 0.1059     0.10841984]

R-squared: 0.1049
Mean Squared Error: 0.8977
Mean Absolute Error: 0.7855
Root Mean Squared Error: 0.9475
Mean CV Score: 0.1055

Coefficients and P-values:
    Coefficient       P-value
0     -0.000235  8.222883e-01
1     -0.285325  0.000000e+00
2      0.124025  0.000000e+00
3     -0.060592  0.000000e+00
4     -0.017530  8.223444e-63
5     -0.120034  0.000000e+00
6     -0.006601  2.976724e-10
7     -0.016789  1.703619e-57
8      0.020879  2.739009e-86
9     -0.000037  9.719575e-01
10     0.129361  0.000000e+00
```

When we use traffic density as target variable in random forest regressor:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
import statsmodels.api as sm

# Read input data from an Excel file
df = pd.read_excel("/Users/vaishnavipalle/Library/Containers/com.microsoft.Excel/Data/Downloads/futuristic_city_traffic_encoded.xlsx")

selected_features = ["City", "Vehicle Type", "Weather", "Economic Condition",
                     "Speed", "Day Of Week", "Hour Of Day", "Is Peak Hour",
                     "Random Event Occurred", "Energy Consumption", "Traffic Density"]

selected_scaled_traffic = df[selected_features]

# Scale the selected features
scaler = StandardScaler()
selected_scaled_traffic = scaler.fit_transform(selected_scaled_traffic)

X = selected_scaled_traffic[:, :-1]
y = selected_scaled_traffic[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the Random Forest Regression model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using various metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = sqrt(mean_squared_error(y_test, y_pred))

# Perform cross-validation and calculate CV scores
cv_scores = cross_val_score(model, X, y, cv=10)
print("Cross-Validation Scores:")
print(cv_scores)  # Print individual CV scores
mean_cv_score = cv_scores.mean()

# Calculate p-values for the coefficients
X_with_intercept = sm.add_constant(X_train)  # Add intercept term for statsmodels
model_sm = sm.OLS(y_train, X_with_intercept).fit()
p_values = model_sm.pvalues

# Get the coefficients and create a DataFrame to display with p-values
coefficients = model_sm.params
coefficients_df = pd.DataFrame({"Coefficient": coefficients, "P-value": p_values})


# Display the evaluation metrics
print(f"\nR-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"Mean CV Score: {mean_cv_score:.4f}")
print("\nCoefficients and P-values:")
print(coefficients_df)
```

```
Cross-Validation Scores:
[0.79401344 0.79124619 0.79165463 0.79076073 0.79306345 0.79241445
 0.79229796 0.79604956 0.79195744 0.79295114]

R-squared: 0.7920
Mean Squared Error: 0.2083
Mean Absolute Error: 0.3101
Root Mean Squared Error: 0.4564
Mean CV Score: 0.7926

Coefficients and P-values:
    Coefficient       P-value
0      0.000085  9.337188e-01
1     -0.289073  0.000000e+00
2      0.138552  0.000000e+00
3     -0.058386  0.000000e+00
4     -0.013758  3.728189e-41
5     -0.196597  0.000000e+00
6     -0.011671  4.513187e-30
7     -0.015802  1.995739e-53
8      0.039876  0.000000e+00
9     -0.000152  8.820232e-01
10     0.205052  0.000000e+00
```

When we use traffic density as target variable in gradient boost regressor:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
import statsmodels.api as sm

# Read input data from an Excel file
df = pd.read_excel("/Users/vaishnavipalle/Library/Containers/com.microsoft.Excel/Data/Downloads/futuristic_city_traffic_encoded.xlsx")

selected_features = ["City", "Vehicle Type", "Weather", "Economic Condition",
                     "Speed", "Day Of Week", "Hour Of Day", "Is Peak Hour",
                     "Random Event Occurred", "Energy Consumption", "Traffic Density"]

selected_scaled_traffic = df[selected_features]

# Scale the selected features
scaler = StandardScaler()
selected_scaled_traffic = scaler.fit_transform(selected_scaled_traffic)

X = selected_scaled_traffic[:, :-1]
y = selected_scaled_traffic[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the Gradient Boosting Regression model
model = GradientBoostingRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using various metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = sqrt(mean_squared_error(y_test, y_pred))

# Perform cross-validation and calculate CV scores
cv_scores = cross_val_score(model, X, y, cv=10)
print("Cross-Validation Scores:")
print(cv_scores)  # Print individual CV scores
mean_cv_score = cv_scores.mean()

# Calculate p-values for the coefficients
X_with_intercept = sm.add_constant(X_train)  # Add intercept term for statsmodels
model_sm = sm.OLS(y_train, X_with_intercept).fit()
p_values = model_sm.pvalues

# Get the coefficients and create a DataFrame to display with p-values
coefficients = model_sm.params
coefficients_df = pd.DataFrame({"Coefficient": coefficients, "P-value": p_values})


# Display the evaluation metrics
print(f"\nR-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"Mean CV Score: {mean_cv_score:.4f}")
print("\nCoefficients and P-values:")
print(coefficients_df)
```

```
Cross-Validation Scores:
[0.72706517 0.72662495 0.72799608 0.72880881 0.72911714 0.7269805
 0.72238998 0.73029536 0.72575992 0.72873111]

R-squared: 0.7294
Mean Squared Error: 0.2709
Mean Absolute Error: 0.3457
Root Mean Squared Error: 0.5205
Mean CV Score: 0.7274

Coefficients and P-values:
    Coefficient       P-value
0      0.000085  9.337188e-01
1     -0.289073  0.000000e+00
2      0.138552  0.000000e+00
3     -0.058386  0.000000e+00
4     -0.013758  3.728189e-41
5     -0.196597  0.000000e+00
6     -0.011671  4.513187e-30
7     -0.015802  1.995739e-53
8      0.039876  0.000000e+00
9     -0.000152  8.820232e-01
10     0.205052  0.000000e+00
```

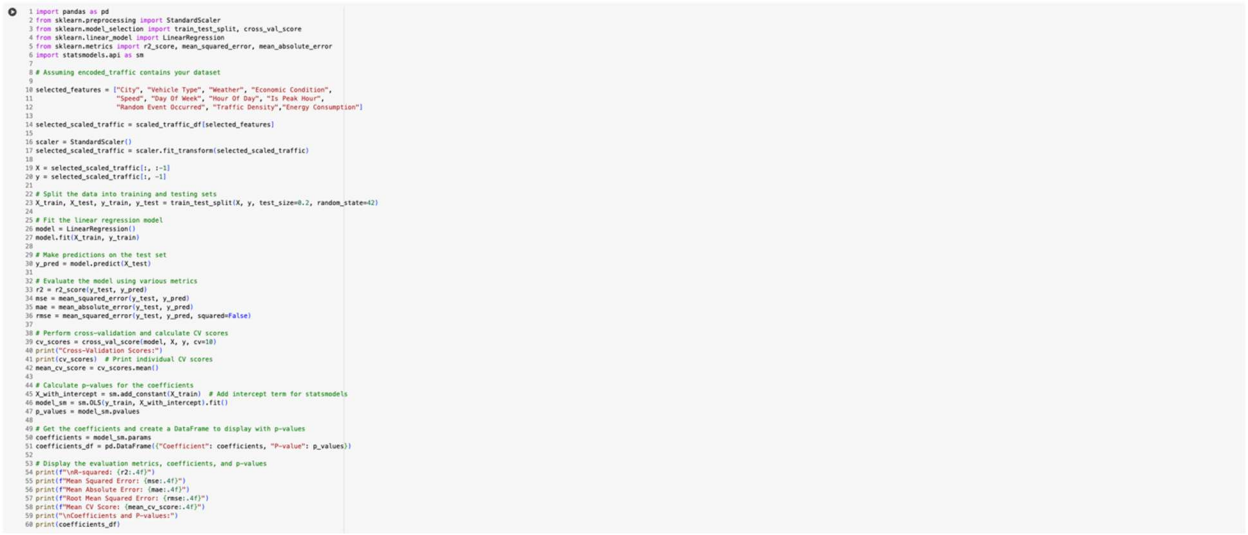
When we use traffic density as target variable in adaboost regressor:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
import statsmodels.api as sm

# Read input data from an Excel file
df = pd.read_excel("/Users/vaishnavipalle/Library/Containers/com.microsoft.Excel/Data/Downloads/futuristic_city_traffic_encoded.xlsx")

selected_features = ["City", "Vehicle Type", "Weather", "Economic Condition",
                     "Speed", "Day Of Week", "Hour Of Day", "Is Peak Hour",
                     "Random Event Occurred", "Energy Consumption", "Traffic Density"]

selected_scaled_traffic = df[selected_features]

# Scale the selected features
scaler = StandardScaler()
selected_scaled_traffic = scaler.fit_transform(selected_scaled_traffic)

X = selected_scaled_traffic[:, :-1]
y = selected_scaled_traffic[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the AdaBoost Regression model
model = AdaBoostRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using various metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = sqrt(mean_squared_error(y_test, y_pred))

# Perform cross-validation and calculate CV scores
cv_scores = cross_val_score(model, X, y, cv=10)
print("Cross-Validation Scores:")
print(cv_scores)  # Print individual CV scores
mean_cv_score = cv_scores.mean()

# Calculate p-values for the coefficients
X_with_intercept = sm.add_constant(X_train)  # Add intercept term for statsmodels
model_sm = sm.OLS(y_train, X_with_intercept).fit()
p_values = model_sm.pvalues

# Get the coefficients and create a DataFrame to display with p-values
coefficients = model_sm.params
coefficients_df = pd.DataFrame({"Coefficient": coefficients, "P-value": p_values})


# Display the evaluation metrics
print(f"\nR-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"Mean CV Score: {mean_cv_score:.4f}")
print("\nCoefficients and P-values:")
print(coefficients_df)
```

```
Cross-Validation Scores:
[0.27116982 0.34779193 0.30038186 0.30748562 0.28720241 0.32933624
 0.33880958 0.40924144 0.33019864 0.2709824 ]

R-squared: 0.2819
Mean Squared Error: 0.7188
Mean Absolute Error: 0.6412
Root Mean Squared Error: 0.8478
Mean CV Score: 0.3193

Coefficients and P-values:
    Coefficient       P-value
0      0.000085  9.337188e-01
1     -0.289073  0.000000e+00
2      0.138552  0.000000e+00
3     -0.058386  0.000000e+00
4     -0.013758  3.728189e-41
5     -0.196597  0.000000e+00
6     -0.011671  4.513187e-30
7     -0.015802  1.995739e-53
8      0.039876  0.000000e+00
9     -0.000152  8.820232e-01
10     0.205052  0.000000e+00
```

When we used energy consumption as target variable in Linear Regression:

LINEAR REGRESSION FOR ENERGY CONSUMPTION



```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import statsmodels.api as sm

# Assuming encoded_traffic contains your dataset

selected_features = ["City", "Vehicle Type", "Weather", "Economic Condition",
                     "Speed", "Day Of Week", "Hour Of Day", "Is Peak Hour",
                     "Random Event Occurred", "Traffic Density","Energy Consumption"]

selected_scaled_traffic = scaled_traffic_df[selected_features]

scaler = StandardScaler()
selected_scaled_traffic = scaler.fit_transform(selected_scaled_traffic)

X = selected_scaled_traffic[:, :-1]
y = selected_scaled_traffic[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using various metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Perform cross-validation and calculate CV scores
cv_scores = cross_val_score(model, X, y, cv=10)
print("Cross-Validation Scores:")
print(cv_scores)  # Print individual CV scores
mean_cv_score = cv_scores.mean()

# Calculate p-values for the coefficients
X_with_intercept = sm.add_constant(X_train)  # Add intercept term for statsmodels
model_sm = sm.OLS(y_train, X_with_intercept).fit()
p_values = model_sm.pvalues

# Get the coefficients and create a DataFrame to display with p-values
coefficients = model_sm.params
coefficients_df = pd.DataFrame({"Coefficient": coefficients, "P-value": p_values})

# Display the evaluation metrics, coefficients, and p-values
print(f"\nR-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"Mean CV Score: {mean_cv_score:.4f}")
print("\nCoefficients and P-values:")
print(coefficients_df)
```

```
Cross-Validation Scores:
[0.73859345 0.74314868 0.74466966 0.74454216 0.74242332 0.74470883
 0.74327346 0.74463977 0.74100455 0.74348662]

R-squared: 0.7427
Mean Squared Error: 0.2570
Mean Absolute Error: 0.3538
Root Mean Squared Error: 0.5069
Mean CV Score: 0.7430

Coefficients and P-values:
    Coefficient        P-value
0      0.000027   9.623704e-01
1      0.015590  1.129033e-154
2     -0.157920   0.000000e+00
3      0.001684   2.790478e-03
4     -0.002913   2.173221e-07
5      0.843176   0.000000e+00
6      0.000282   6.156056e-01
7      0.001086   5.396577e-02
8     -0.067078   0.000000e+00
9     -0.000987   7.882638e-02
10     0.037194   0.000000e+00
```

When we used energy consumption as target variable in Random Forest regressor:

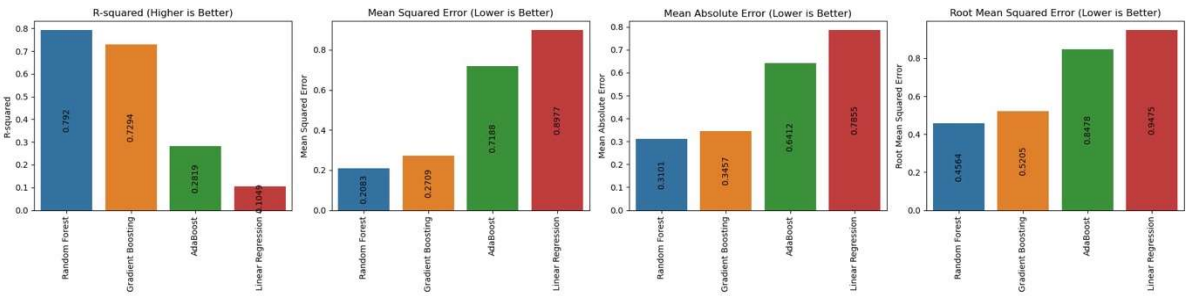
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
import statsmodels.api as sm

# Read input data from an Excel file
df = pd.read_excel("/Users/vaishnavipalle/Library/Containers/com.microsoft.Excel/Data/Downloads/futuristic_city_traffic_encoded.xlsx")

selected_features = ["City", "Vehicle Type", "Weather", "Economic Condition",
                     "Speed", "Day Of Week", "Hour Of Day", "Is Peak Hour",
                     "Random Event Occurred", "Traffic Density","Energy Consumption"]

selected_scaled_traffic = df[selected_features]

# Scale the selected features
scaler = StandardScaler()
selected_scaled_traffic = scaler.fit_transform(selected_scaled_traffic)

X = selected_scaled_traffic[:, :-1]
y = selected_scaled_traffic[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the Random Forest Regression model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model using various metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = sqrt(mean_squared_error(y_test, y_pred))

# Perform cross-validation and calculate CV scores
cv_scores = cross_val_score(model, X, y, cv=10)
print("Cross-Validation Scores:")
print(cv_scores)  # Print individual CV scores
mean_cv_score = cv_scores.mean()

# Calculate p-values for the coefficients
X_with_intercept = sm.add_constant(X_train)  # Add intercept term for statsmodels
model_sm = sm.OLS(y_train, X_with_intercept).fit()
p_values = model_sm.pvalues

# Get the coefficients and create a DataFrame to display with p-values
coefficients = model_sm.params
coefficients_df = pd.DataFrame({"Coefficient": coefficients, "P-value": p_values})


# Display the evaluation metrics
print(f"\nR-squared: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"Mean CV Score: {mean_cv_score:.4f}")
print("\nCoefficients and P-values:")
print(coefficients_df)
```

```
Cross-Validation Scores:
[0.98141726 0.98129785 0.98198168 0.98123609 0.98094411 0.98129578
 0.98084125 0.98144513 0.98095924 0.98063072]

R-squared: 0.9813
Mean Squared Error: 0.0189
Mean Absolute Error: 0.0521
Root Mean Squared Error: 0.1373
Mean CV Score: 0.9812

Coefficients and P-values:
    Coefficient        P-value
0     -0.000113   8.425501e-01
1      0.018260  3.916822e-204
2     -0.119500   0.000000e+00
3      0.001216   3.338105e-02
4     -0.002998   1.465908e-07
5      0.842902   0.000000e+00
6      0.000124   8.276322e-01
7      0.000231   6.858584e-01
8     -0.063988   0.000000e+00
9     -0.000774   1.757932e-01
10     0.063611   0.000000e+00
```

MODEL COMPARISON:

Comparing accuracy of all the models where traffic density is considered as target variable.

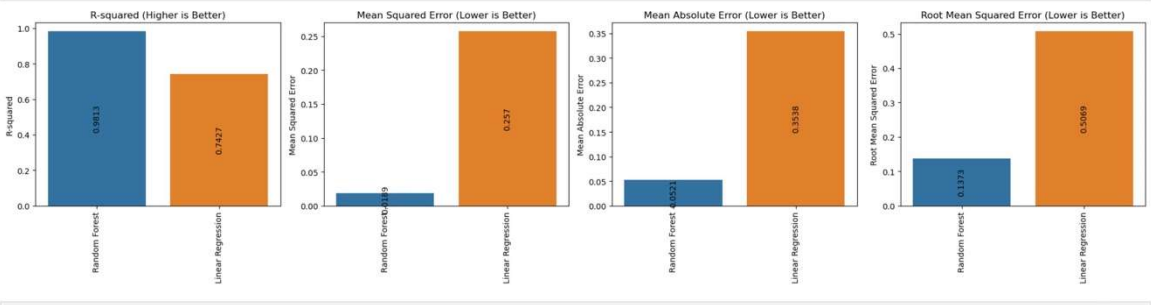
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define the data
dd = {
    "Linear Regression": {"R-squared": 0.1049 ,"Mean Squared Error": 0.8977 ,
                          "Mean Absolute Error": 0.7855,"Root Mean Squared Error": 0.9475},
    "Random Forest": {"R-squared": 0.7920 , "Mean Squared Error": 0.2083,
                      "Mean Absolute Error": 0.3101 ,"Root Mean Squared Error": 0.4564},
    "Gradient Boosting": {"R-squared": 0.7294, "Mean Squared Error": 0.2709,
                          "Mean Absolute Error": 0.3457,"Root Mean Squared Error": 0.5205},
    "AdaBoost": {"R-squared": 0.2819 , "Mean Squared Error": 0.7188,
                 "Mean Absolute Error": 0.6412 ,"Root Mean Squared Error": 0.8478}
}

# Create a DataFrame
dt = pd.DataFrame(dd)

# Transpose the DataFrame
dt = dt.T

# Sort the DataFrame based on R-squared scores
dt = dt.sort_values("R-squared", ascending=False)

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(20, 5))

# Plot R-squared scores
sns.barplot(x=dt.index, y=dt["R-squared"], ax=axes[0])
for container in axes[0].containers:
    axes[0].bar_label(container, label_type="center", rotation=90)
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=90)
axes[0].set_title("R-squared (Higher is Better)")

# Plot Mean Squared Error
sns.barplot(x=dt.index, y=dt["Mean Squared Error"], ax=axes[1])
for container in axes[1].containers:
    axes[1].bar_label(container, label_type="center", rotation=90)
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=90)
axes[1].set_title("Mean Squared Error (Lower is Better)")

# Plot Mean Absolute Error
sns.barplot(x=dt.index, y=dt["Mean Absolute Error"], ax=axes[2])
for container in axes[2].containers:
    axes[2].bar_label(container, label_type="center", rotation=90)
axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation=90)
axes[2].set_title("Mean Absolute Error (Lower is Better)")

# Plot Root Mean Squared Error
sns.barplot(x=dt.index, y=dt["Root Mean Squared Error"], ax=axes[3])
for container in axes[3].containers:
    axes[3].bar_label(container, label_type="center", rotation=90)
axes[3].set_xticklabels(axes[3].get_xticklabels(), rotation=90)
axes[3].set_title("Root Mean Squared Error (Lower is Better)")

plt.tight_layout()
plt.show()
```



Comparing accuracy of all the models where energy consumption is considered as target variable.

When we tried taking energy consumption as target variable for linear regression and random forest regressor models where random forest regressor gave the better results compared to linear regression.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define the data
dd = {
    "Linear Regression": {"R-squared":0.7427,"Mean Squared Error":0.2570,
                          "Mean Absolute Error":0.3538, "Root Mean Squared Error": 0.5069,},
    "Random Forest": {"R-squared": 0.9813 , "Mean Squared Error": 0.0189,
                      "Mean Absolute Error": 0.0521 , "Root Mean Squared Error": 0.1373}
}

# Create a DataFrame
dt = pd.DataFrame(dd)

# Transpose the DataFrame
dt = dt.T

# Sort the DataFrame based on R-squared scores
dt = dt.sort_values("R-squared", ascending=False)

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(20, 5))

# Plot R-squared scores
sns.barplot(x=dt.index, y=dt["R-squared"], ax=axes[0])
for container in axes[0].containers:
    axes[0].bar_label(container, label_type="center", rotation=90)
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=90)
axes[0].set_title("R-squared (Higher is Better)")

# Plot Mean Squared Error
sns.barplot(x=dt.index, y=dt["Mean Squared Error"], ax=axes[1])
for container in axes[1].containers:
    axes[1].bar_label(container, label_type="center", rotation=90)
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=90)
axes[1].set_title("Mean Squared Error (Lower is Better)")

# Plot Mean Absolute Error
sns.barplot(x=dt.index, y=dt["Mean Absolute Error"], ax=axes[2])
for container in axes[2].containers:
    axes[2].bar_label(container, label_type="center", rotation=90)
axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation=90)
axes[2].set_title("Mean Absolute Error (Lower is Better)")

# Plot Root Mean Squared Error
sns.barplot(x=dt.index, y=dt["Root Mean Squared Error"], ax=axes[3])
for container in axes[3].containers:
    axes[3].bar_label(container, label_type="center", rotation=90)
axes[3].set_xticklabels(axes[3].get_xticklabels(), rotation=90)
axes[3].set_title("Root Mean Squared Error (Lower is Better)")

plt.tight_layout()
plt.show()
```



MANAGERIAL IMPLICATIONS:

The managerial implications of the predictive modeling for traffic congestion in the futuristic urban environment are profound. By leveraging data-driven insights, businesses and transportation authorities can optimize traffic management strategies, reduce congestion-related disruptions, enhance operational efficiency, and ultimately improve the overall quality of transportation

services. These implications underscore the importance of adopting proactive and data-driven approaches to address contemporary urban transportation challenges effectively.

CONCLUSION:

In conclusion, the Random Forest model stands out for its remarkable accuracy in predicting traffic congestion, offering invaluable insights and actionable strategies for tackling the business problem associated with congestion. The model can reliably estimate congestion levels because it can capture intricate interactions between multiple factors.

With these forecasts in hand, companies, transit agencies, and local governments can take focused action to lessen the negative impacts of traffic congestion. Through the strategic deployment of resources and interventions, stakeholders can maximize traffic flow, reduce delays, and improve overall transportation efficiency by utilizing the model's forecasts.