```
# import pandas as pd
```

```
data = pd.read_csv("/content/Traffic.csv")
```

```
data
```

	Time	Date	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Tra Situa
0	12:00:00 AM	10	Tuesday	31	0	4	4	39	
1	12:15:00 AM	10	Tuesday	49	0	3	3	55	
2	12:30:00 AM	10	Tuesday	46	0	3	6	55	
3	12:45:00 AM	10	Tuesday	51	0	2	5	58	
4	1:00:00 AM	10	Tuesday	57	6	15	16	94	n
...	...	...	...	...	...	...	...	...	
2971	10:45:00 PM	9	Thursday	16	3	1	36	56	n

```
data.head()
```

	Time	Date	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Traffi Situatio
0	12:00:00 AM	10	Tuesday	31	0	4	4	39	lov
1	12:15:00 AM	10	Tuesday	49	0	3	3	55	lov
	12:30:00								

```
data.tail()
```

	Time	Date	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Tra Situa
2971	10:45:00 PM	9	Thursday	16	3	1	36	56	n
2972	11:00:00 PM	9	Thursday	11	0	1	30	42	n
	11:15:00								

```
data.axes
```

```
[RangeIndex(start=0, stop=2976, step=1),
 Index(['Time', 'Date', 'Day of the week', 'CarCount', 'BikeCount', 'BusCount',
        'TruckCount', 'Total', 'Traffic Situation'],
       dtype='object')]
```

```
data.shape
```

```
(2976, 9)
```

```
data.size
```

```
26784
```

```
data.columns
```

```
Index(['Time', 'Date', 'Day of the week', 'CarCount', 'BikeCount', 'BusCount',
       'TruckCount', 'Total', 'Traffic Situation'],
      dtype='object')
```

```
data.dtypes
```

```
Time                 object
Date                  int64
Day of the week      object
CarCount              int64
BikeCount             int64
BusCount              int64
TruckCount            int64
Total                 int64
Traffic Situation    object
dtype: object
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2976 entries, 0 to 2975
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Time               2976 non-null   object
 1   Date               2976 non-null   int64 
 2   Day of the week    2976 non-null   object
 3   CarCount           2976 non-null   int64 
 4   BikeCount          2976 non-null   int64 
 5   BusCount           2976 non-null   int64 
 6   TruckCount         2976 non-null   int64 
 7   Total              2976 non-null   int64 
 8   Traffic Situation  2976 non-null   object
dtypes: int64(6), object(3)
memory usage: 209.4+ KB
```

```
data.isnull().sum()
```

```
Time                 0
Date                 0
Day of the week      0
CarCount             0
BikeCount            0
BusCount             0
TruckCount           0
Total                0
Traffic Situation    0
dtype: int64
```

```
data.isnull()
```

	Time	Date	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Traffic Situation
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
2971	False	False	False	False	False	False	False	False	False
2972	False	False	False	False	False	False	False	False	False
2973	False	False	False	False	False	False	False	False	False
2974	False	False	False	False	False	False	False	False	False
2975	False	False	False	False	False	False	False	False	False

2976 rows × 9 columns

```
# LABEL ENCODING
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
```

```
data["Day of the week"].value_counts()
```

```
Day of the week
Tuesday      480
Wednesday    480
Thursday     480
Friday       384
Saturday     384
Sunday       384
```

```
Monday       384
Name: count, dtype: int64
```

```
data['Day of the week']=le.fit_transform(data['Day of the week'])
```

```
data['Day of the week'].value_counts()
```

```
Day of the week
5    480
6    480
4    480
0    384
2    384
3    384
1    384
Name: count, dtype: int64
```

```
data['Traffic Situation'].value_counts()
```

```
Traffic Situation
normal    1669
heavy      682
high       321
low        304
Name: count, dtype: int64
```

```
data['Traffic Situation']=le.fit_transform(data['Traffic Situation'])
```

```
data['Traffic Situation'].value_counts()
```

```
Traffic Situation
3    1669
0     682
1     321
2     304
Name: count, dtype: int64
```

```
data
```

	Time	Date	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Traffic Situation
0	12:00:00 AM	10	5	31	0	4	4	39	2
1	12:15:00 AM	10	5	49	0	3	3	55	2
2	12:30:00 AM	10	5	46	0	3	6	55	2
3	12:45:00 AM	10	5	51	0	2	5	58	2
4	1:00:00 AM	10	5	57	6	15	16	94	3
...	...	...	...	...	...	...	...	...	...
2971	10:45:00 PM	9	4	16	3	1	36	56	3
2972	11:00:00 PM	9	4	11	0	1	30	42	3
2973	11:15:00 PM	9	4	15	4	1	25	45	3
2974	11:30:00 PM	9	4	16	5	0	27	48	3
2975	11:45:00 PM	9	4	14	3	1	15	33	3

2976 rows × 9 columns

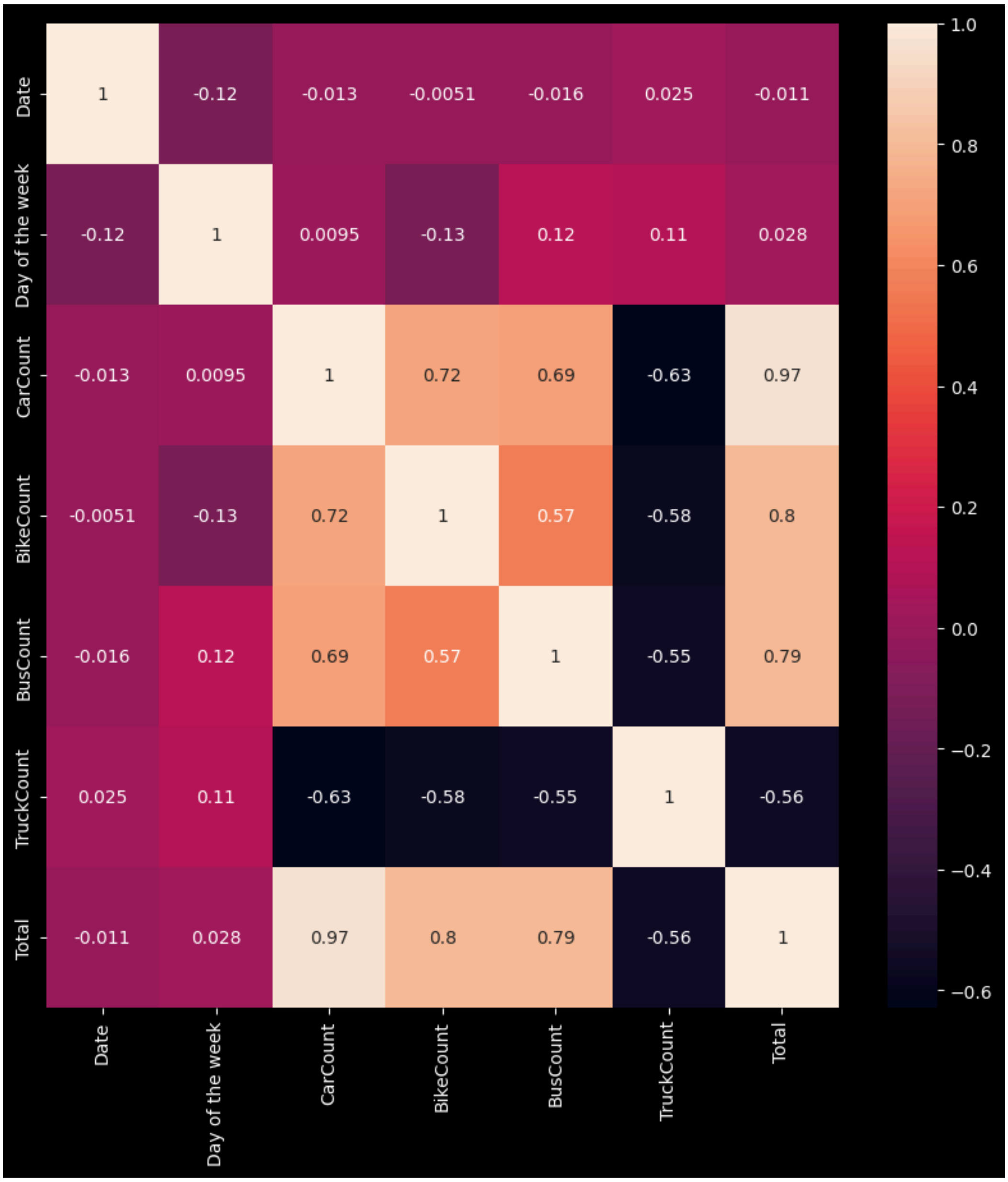
```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('dark_background')
```

```
data_df = pd.DataFrame(data)
```

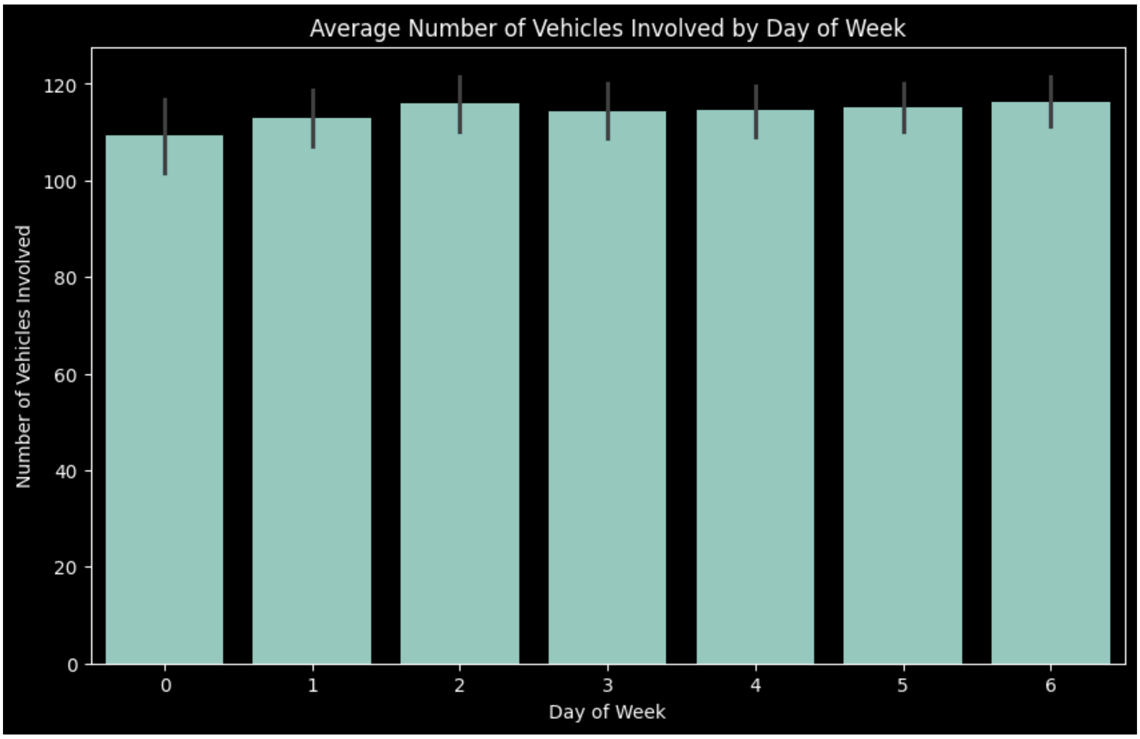
```
# import seaborn
# import matplotlib.pyplot as pt
# # pt.figure(figsize=(10,10))
# correlation=data_df.corr()
# heatmap = seaborn.heatmap(correlation,annot=True)
# pt.show()
```

```
import seaborn
import matplotlib.pyplot as pt
```

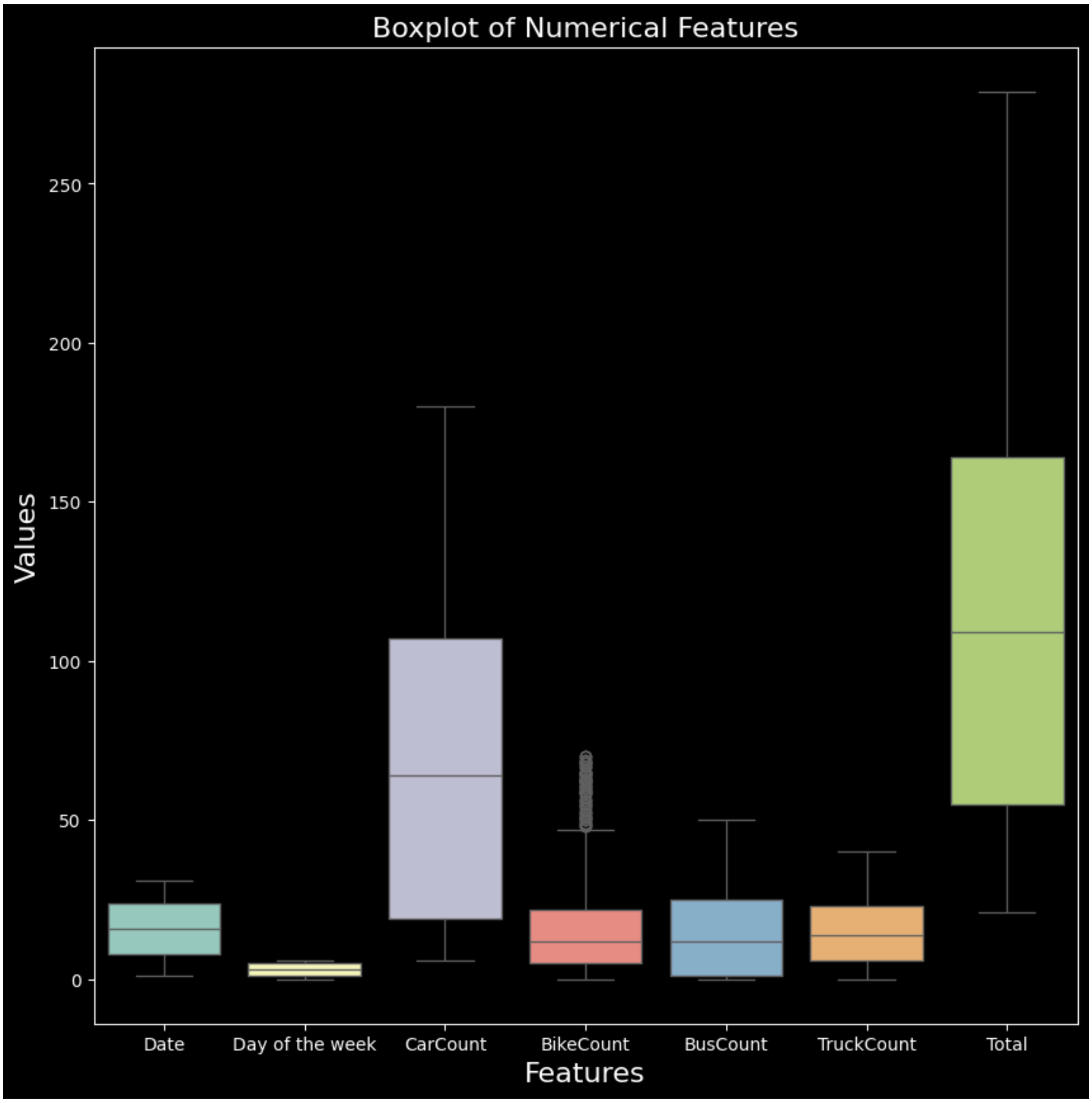
```
import pandas as pd # Import pandas for data manipulation

pt.figure(figsize=(10,10))

# Convert 'Traffic Situation' to datetime objects if it represents time
# Assuming 'Traffic Situation' is meant to represent time, let's convert it:
data_df['Traffic Situation'] = pd.to_datetime(data_df['Traffic Situation'])

# Extract numerical features for correlation
# Select only the numerical columns for correlation calculation:
numerical_data = data_df.select_dtypes(include=['number'])

correlation = numerical_data.corr()
heatmap = seaborn.heatmap(correlation, annot=True)
pt.show()
```




```
plt.figure(figsize=(10, 6))
sns.barplot(x='Day of the week', y='Total', data=data)
plt.xlabel('Day of Week')
plt.ylabel('Number of Vehicles Involved')
plt.title('Average Number of Vehicles Involved by Day of Week')
plt.show()
```

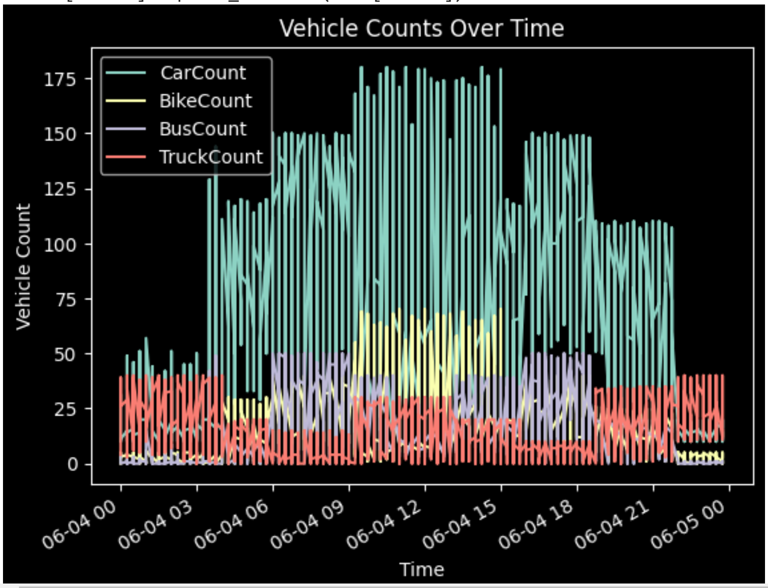



```
import matplotlib.pyplot as plt
# Boxplot of numerical features
plt.figure(figsize=(10, 10))
sns.boxplot(data=numerical_data)
plt.xlabel('Features', fontsize=16)
plt.ylabel('Values', fontsize=16)
plt.title('Boxplot of Numerical Features', fontsize=16)
plt.show()
```



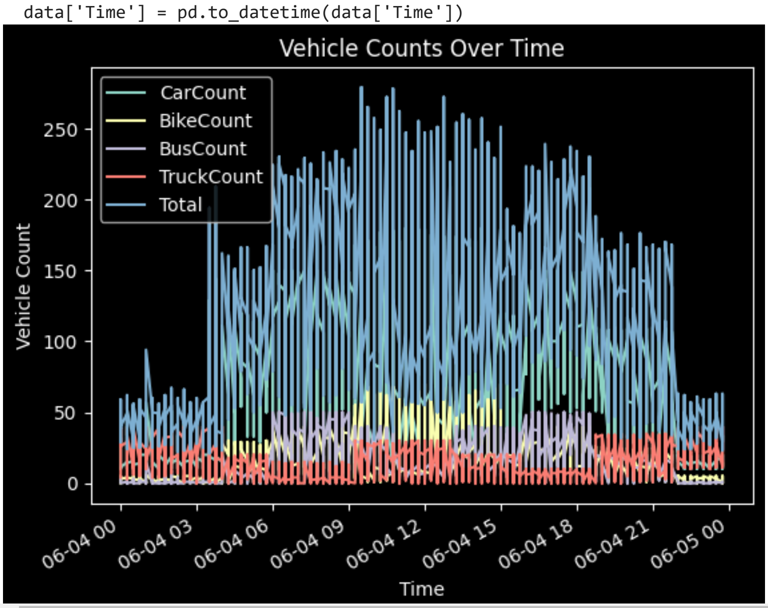

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data into a pandas DataFrame
data = pd.read_csv('/content/Traffic.csv')

# Convert Time column to datetime format
data['Time'] = pd.to_datetime(data['Time'])

# Set Time as the index
data = data.set_index('Time')

# Create a line chart for vehicle counts
data[['CarCount', 'BikeCount', 'BusCount', 'TruckCount']].plot()
plt.xlabel('Time')
plt.ylabel('Vehicle Count')
plt.title('Vehicle Counts Over Time')
plt.show()
```

```
<ipython-input-34-6ef2745294a4>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back tc
  data['Time'] = pd.to_datetime(data['Time'])
```



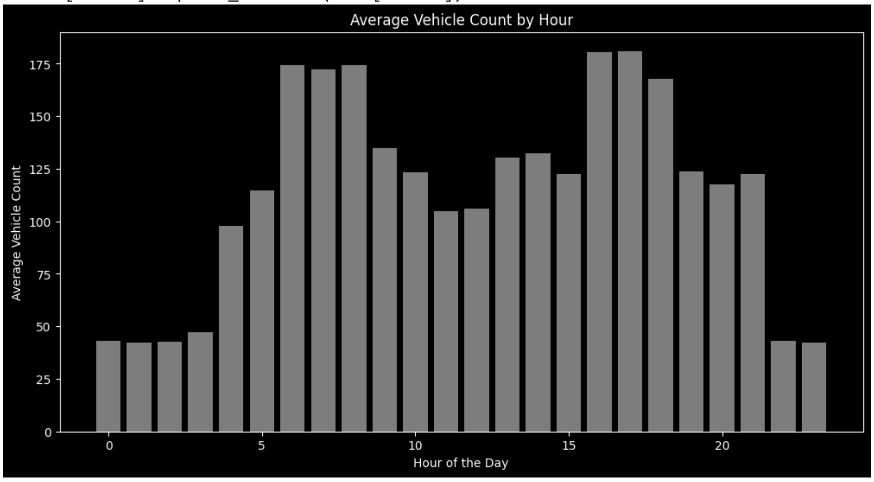

```
import pandas as pd
import matplotlib.pyplot as plt

# Load data into a pandas DataFrame
data = pd.read_csv('/content/Traffic.csv')

# Convert Time column to datetime format
data['Time'] = pd.to_datetime(data['Time'])

# Set Time as the index
data = data.set_index('Time')

# Create a line chart for vehicle counts
data[['CarCount', 'BikeCount', 'BusCount', 'TruckCount', 'Total']].plot()
plt.xlabel('Time')
plt.ylabel('Vehicle Count')
plt.title('Vehicle Counts Over Time')
plt.show()
```

```
<ipython-input-35-c77d66296049>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back tc
  data['Time'] = pd.to_datetime(data['Time'])
```

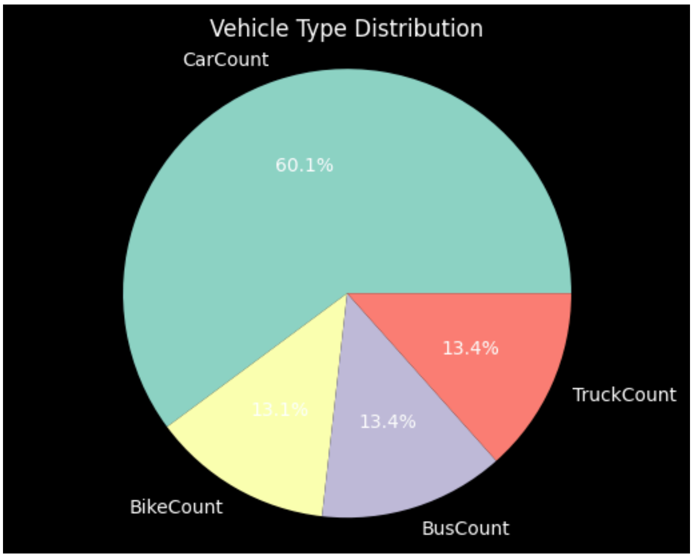


```
import pandas as pd
import matplotlib.pyplot as plt

# Load data into a pandas DataFrame
data = pd.read_csv('/content/Traffic.csv')

# Convert Time column to datetime format
data['Time'] = pd.to_datetime(data['Time'])

# Group data by hour and calculate mean vehicle counts
hourly_data = data.groupby(data['Time'].dt.hour)[['CarCount', 'BikeCount', 'BusCount', 'TruckCount', 'Total']].mean().reset_index()

# Create a bar chart
plt.figure(figsize=(12, 6))
plt.bar(hourly_data['Time'], hourly_data['Total'], color='grey')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Vehicle Count')
plt.title('Average Vehicle Count by Hour')
plt.show()
```

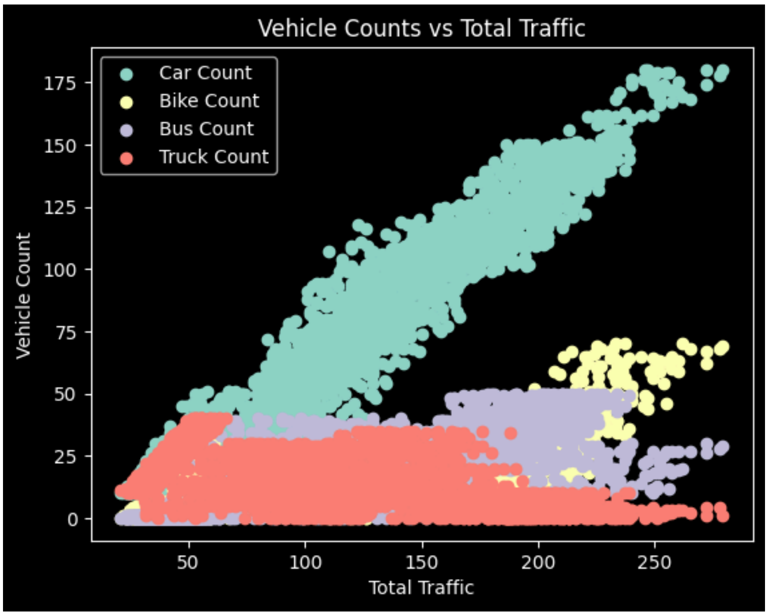
```
<ipython-input-36-9fa2e4b9c1f0>:8: UserWarning: Could not infer format, so each eleme
  data['Time'] = pd.to_datetime(data['Time'])
```




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

data = pd.read_csv('/content/Traffic.csv')


vehicle_counts = data[['CarCount', 'BikeCount', 'BusCount', 'TruckCount']].sum()
plt.pie(vehicle_counts, labels=vehicle_counts.index, autopct='%1.1f%%')
plt.axis('equal')
plt.title('Vehicle Type Distribution')
plt.show()
```

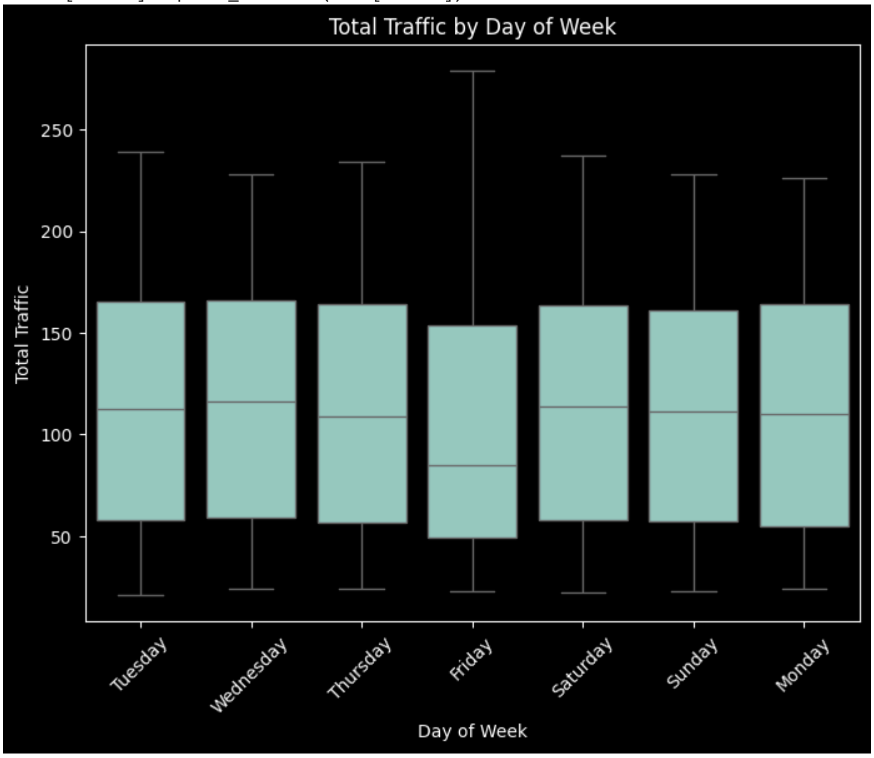



```
plt.scatter(data['Total'], data['CarCount'], label='Car Count')
plt.scatter(data['Total'], data['BikeCount'], label='Bike Count')
plt.scatter(data['Total'], data['BusCount'], label='Bus Count')
plt.scatter(data['Total'], data['TruckCount'], label='Truck Count')
plt.xlabel('Total Traffic')
plt.ylabel('Vehicle Count')
plt.title('Vehicle Counts vs Total Traffic')
plt.legend()
plt.show()
```

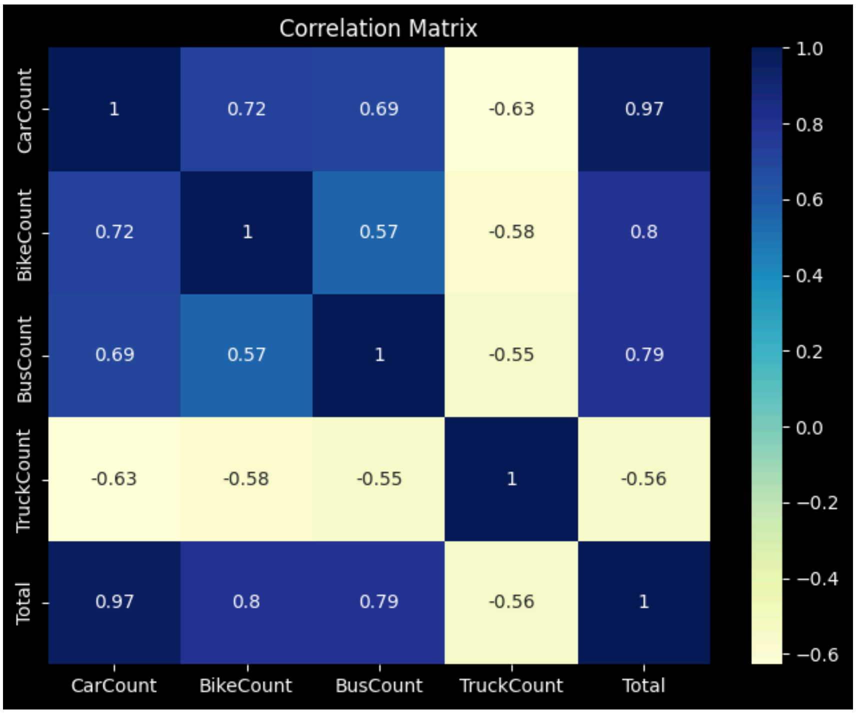



```
data['Time'] = pd.to_datetime(data['Time'])
plt.figure(figsize=(8, 6))
sns.boxplot(x='Day of the week', y='Total', data=data)
plt.xlabel('Day of Week')
plt.ylabel('Total Traffic')
plt.title('Total Traffic by Day of Week')
plt.xticks(rotation=45)
plt.show()
```

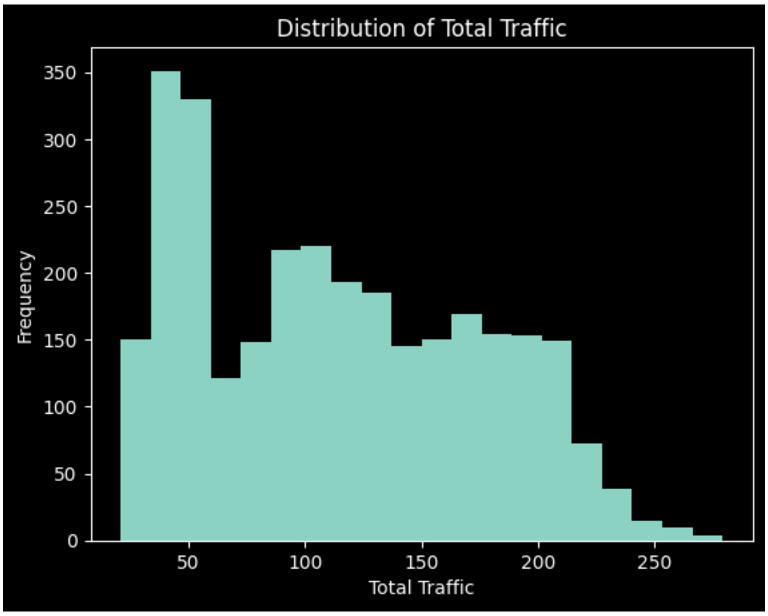
```
<ipython-input-40-429b07818c24>:1: UserWarning: Could not infer format, so each eleme
  data['Time'] = pd.to_datetime(data['Time'])
```




```
corr_matrix = data[['CarCount', 'BikeCount', 'BusCount', 'TruckCount', 'Total']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu')
plt.title('Correlation Matrix')
plt.show()
```




```
plt.hist(data['Total'], bins=20)
plt.xlabel('Total Traffic')
plt.ylabel('Frequency')
plt.title('Distribution of Total Traffic')
plt.show()
```



Linear Regression

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

# Preprocess Data
# Convert 'Time' and 'Date' to numeric features
data['Time'] = pd.to_datetime(data['Time']).dt.hour
data['Date'] = pd.to_datetime(data['Date']).dt.day


# Define input and output columns
input_features = ['Time', 'Date', 'Day of the week']
output_features = ['CarCount', 'BikeCount', 'TruckCount', 'BusCount', 'Traffic Situation']

X = data[input_features]
y = data[output_features]

# 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
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict
y_pred = regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Display predicted and actual values
print(f'Predicted values: {y_pred}')
print(f'Actual values: {y_test.values}')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-45-c04f4ec7cea3> in <cell line: 26>()
     24 # Fit the linear regression model
     25 regressor = LinearRegression()
---> 26 regressor.fit(X_train, y_train)
```

Support Vector Regression

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.multioutput import MultiOutputRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

# Preprocess Data
# Convert 'Time' and 'Date' to numeric features
data['Time'] = pd.to_datetime(data['Time']).dt.hour
data['Date'] = pd.to_datetime(data['Date']).dt.day

# Define input and output columns
input_features = ['Time', 'Date', 'Day of the week']
output_features = ['CarCount', 'BikeCount', 'TruckCount', 'BusCount', 'Traffic Situation']

X = data[input_features]
y = data[output_features]

# 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)

# Scale the data (important for SVR)
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
y_train_scaled = scaler_y.fit_transform(y_train)
y_test_scaled = scaler_y.transform(y_test)

# Initialize and fit the SVR model within a MultiOutputRegressor
svr_model = MultiOutputRegressor(SVR(kernel='rbf'))
svr_model.fit(X_train_scaled, y_train_scaled)

# Make predictions
y_pred_scaled = svr_model.predict(X_test_scaled)

# Inverse transform the predictions and true values
y_pred = scaler_y.inverse_transform(y_pred_scaled)
y_test = scaler_y.inverse_transform(y_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Display predicted and actual values
```