```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import pdfplumber

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from google.colab import drive



from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv('/content/drive/MyDrive/DS project/traffic.csv')
```

```
df.head()
```

	DateTime	Junction	Vehicles	ID
0	2015-11-01 00:00:00	1	15	20151101001
1	2015-11-01 01:00:00	1	13	20151101011
2	2015-11-01 02:00:00	1	10	20151101021
3	2015-11-01 03:00:00	1	7	20151101031
4	2015-11-01 04:00:00	1	9	20151101041

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df = pd.read_csv('/content/drive/MyDrive/DS project/city_traffic_data_1000.csv')

df.head()
```

	City	Road_Length_km	Vehicle_Count_lakhs	Population_lakhs	Congestion_Index_2023	Vehicle_Count	Population	Vehicles_per_km	Pop
0	Mumbai	2200	42	124	52	4200000	12400000	1909.09	
1	Pune	19391	32	95	46	3200000	9500000	165.03	
2	Nagpur	14734	18	46	38	1800000	4600000	122.17	
3	Nashik	19769	15	35	33	1500000	3500000	75.88	
4	Thane	4479	25	75	40	2500000	7500000	558.16	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df
```

	DateTime	Junction	Vehicles	ID
0	2015-11-01 00:00:00	1	15	20151101001
1	2015-11-01 01:00:00	1	13	20151101011
2	2015-11-01 02:00:00	1	10	20151101021
3	2015-11-01 03:00:00	1	7	20151101031
4	2015-11-01 04:00:00	1	9	20151101041
...	...	...	...	...
48115	2017-06-30 19:00:00	4	11	20170630194
48116	2017-06-30 20:00:00	4	30	20170630204
48117	2017-06-30 21:00:00	4	16	20170630214
48118	2017-06-30 22:00:00	4	22	20170630224
48119	2017-06-30 23:00:00	4	12	20170630234

48120 rows × 4 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.columns
```

```
Index(['DateTime', 'Junction', 'Vehicles', 'ID'], dtype='object')
```

```
#basic overview
print("Shape:", df.shape)
print("\nData Types:\n", df.dtypes)

print("\nMissing Values:\n", df.isnull().sum())


print("\nColumns:\n", df.columns)

df.describe()
```

```
Shape: (48120, 4)

Data Types:
 DateTime    object
Junction     int64
Vehicles     int64
ID           int64
dtype: object

Missing Values:
 DateTime    0
Junction    0
Vehicles    0
ID          0
dtype: int64

Columns:
 Index(['DateTime', 'Junction', 'Vehicles', 'ID'], dtype='object')
```

	Junction	Vehicles	ID
count	48120.000000	48120.000000	4.812000e+04
mean	2.180549	22.791334	2.016330e+10
std	0.966955	20.750063	5.944854e+06
min	1.000000	1.000000	2.015110e+10
25%	1.000000	9.000000	2.016042e+10
50%	2.000000	15.000000	2.016093e+10
75%	3.000000	29.000000	2.017023e+10
max	4.000000	180.000000	2.017063e+10

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48120 entries, 0 to 48119
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   DateTime     48120 non-null  datetime64[ns]
 1   Junction     48120 non-null  int64
 2   Vehicles     48120 non-null  int64
 3   ID           48120 non-null  int64
 4   hour         48120 non-null  int32
 5   day_of_week  48120 non-null  int32
 6   month        48120 non-null  int32
dtypes: datetime64[ns](1), int32(3), int64(3)
memory usage: 2.0 MB
```

```
#checking unique values
sns.set_style('whitegrid')

print("Unique values per column:")
print(df[['Junction', 'hour', 'day_of_week', 'month']].nunique())


print("\nUnique IDs:")
print(df['ID'].nunique())
```

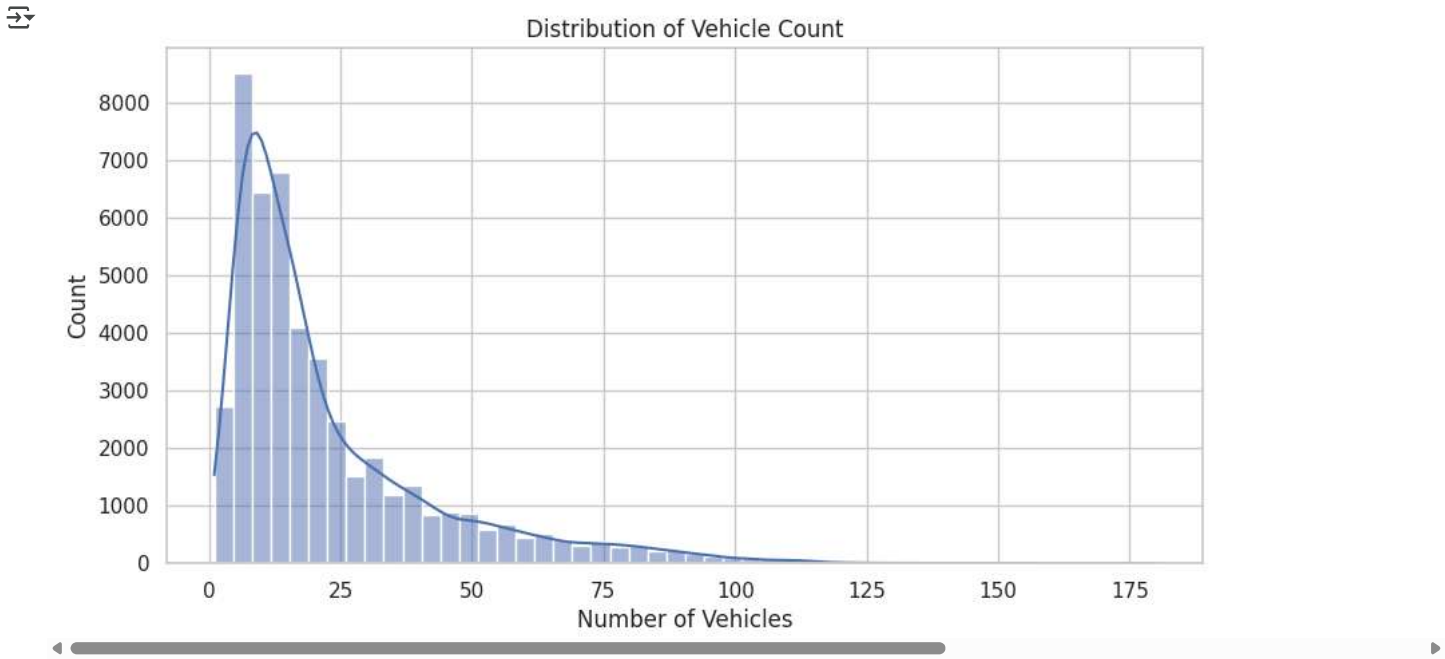
```
Unique values per column:
Junction        4
hour           24
day_of_week     7
month          12
dtype: int64

Unique IDs:
48120
```

```
# Convert lakh-based values to actual numbers
df['Vehicle_Count'] = df['Vehicle_Count_lakhs'] * 100000
df['Population'] = df['Population_lakhs'] * 100000

df['Vehicles_per_km'] = (df['Vehicle_Count'] / df['Road_Length_km']).round(2)
df['Population_per_km'] = (df['Population'] / df['Road_Length_km']).round(2)
df['Road_km_per_lakh_population'] = (df['Road_Length_km'] / df['Population_lakhs']).round(2)
df['Congestion_per_km'] = (df['Congestion_Index_2023'] / df['Road_Length_km']).round(4)

df.head()
```

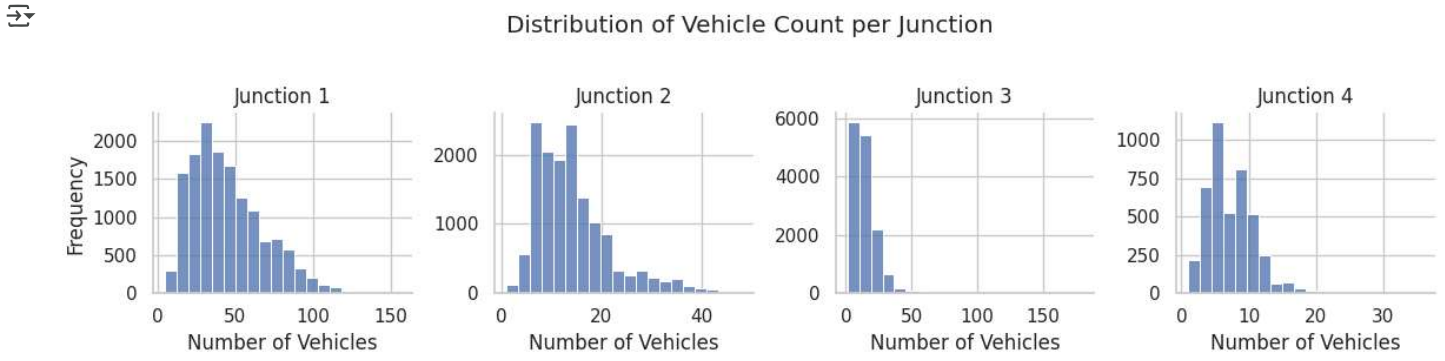
	City	Road_Length_km	Vehicle_Count_lakhs	Population_lakhs	Congestion_Index_2023	Vehicle_Count	Population	Vehicles_per_km	Pop
0	Mumbai	2200	42	124	52	4200000	12400000	1909.09	
1	Pune	19391	32	95	46	3200000	9500000	165.03	
2	Nagpur	14734	18	46	38	1800000	4600000	122.17	
3	Nashik	19769	15	35	33	1500000	3500000	75.88	
4	Thane	4479	25	75	40	2500000	7500000	558.16	

Next steps: Generate code with df | View recommended plots | New interactive sheet

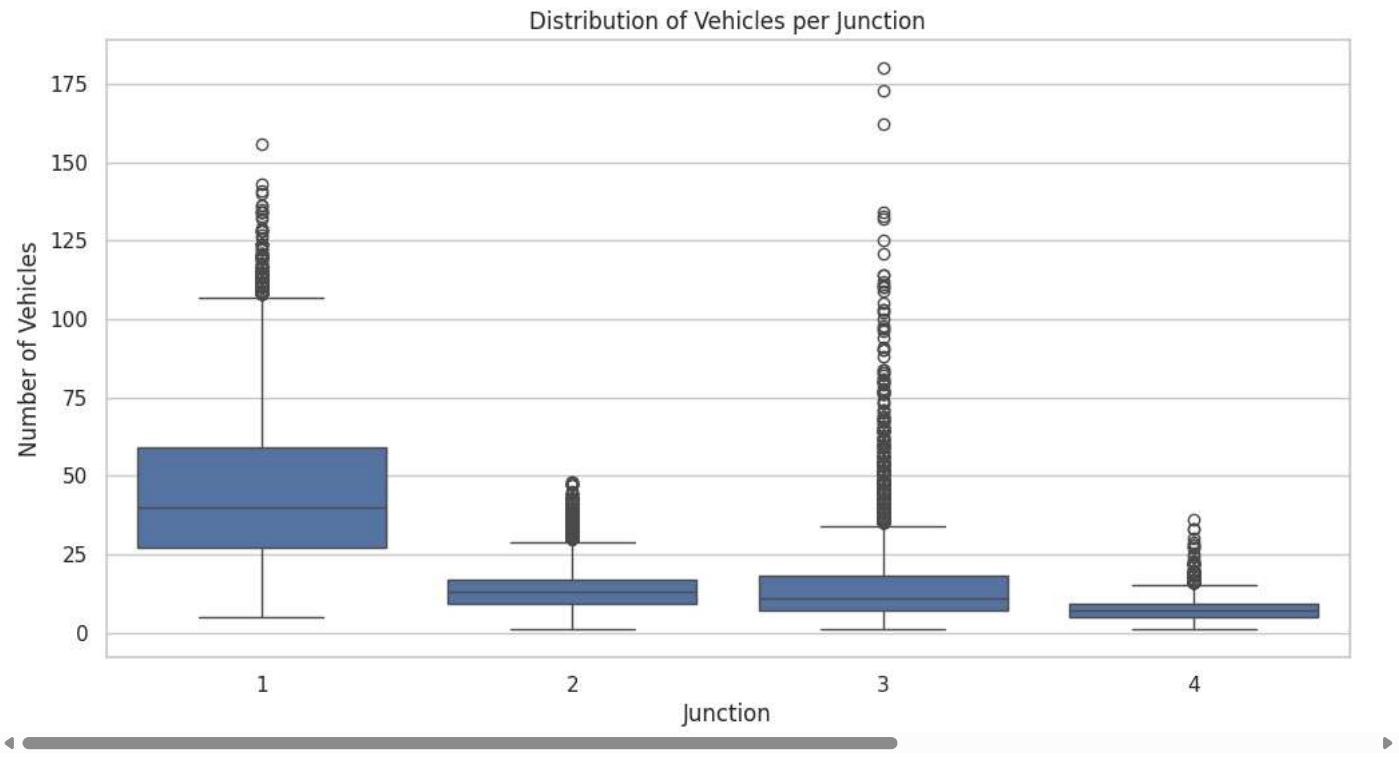
```
# Distribution curve of vehicle count
plt.figure(figsize=(10, 5))
sns.histplot(df['Vehicles'], kde=True, bins=50)
plt.title('Distribution of Vehicle Count')
plt.xlabel('Number of Vehicles')
plt.show()
```




```
# FacetGrid visualization
g = sns.FacetGrid(df, col="Junction", col_wrap=4, sharex=False, sharey=False, height=3)
g.map(sns.histplot, "Vehicles", bins=20, kde=False)
g.fig.suptitle('Distribution of Vehicle Count per Junction', y=1.02)
g.set_axis_labels("Number of Vehicles", "Frequency")
g.set_titles("Junction {col_name}")
plt.tight_layout()
plt.show()
```

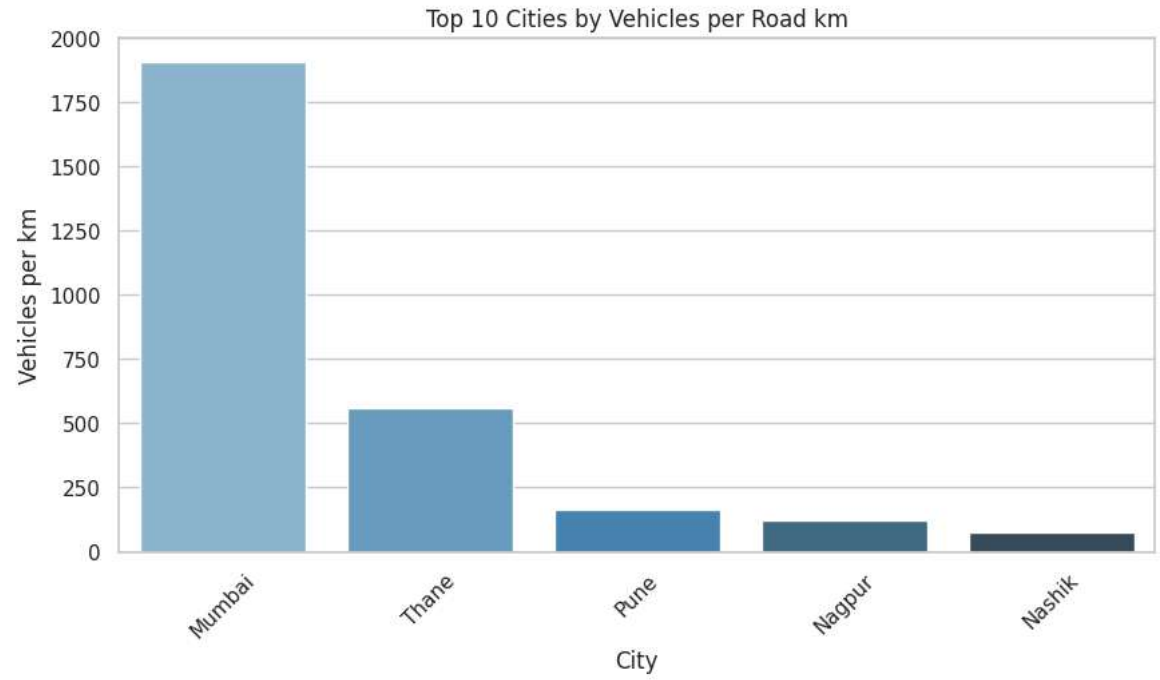



```
#Box and whiskers plot for each junction
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Junction', y='Vehicles')
plt.title('Distribution of Vehicles per Junction')
plt.xlabel('Junction')
plt.ylabel('Number of Vehicles')
plt.show()
```

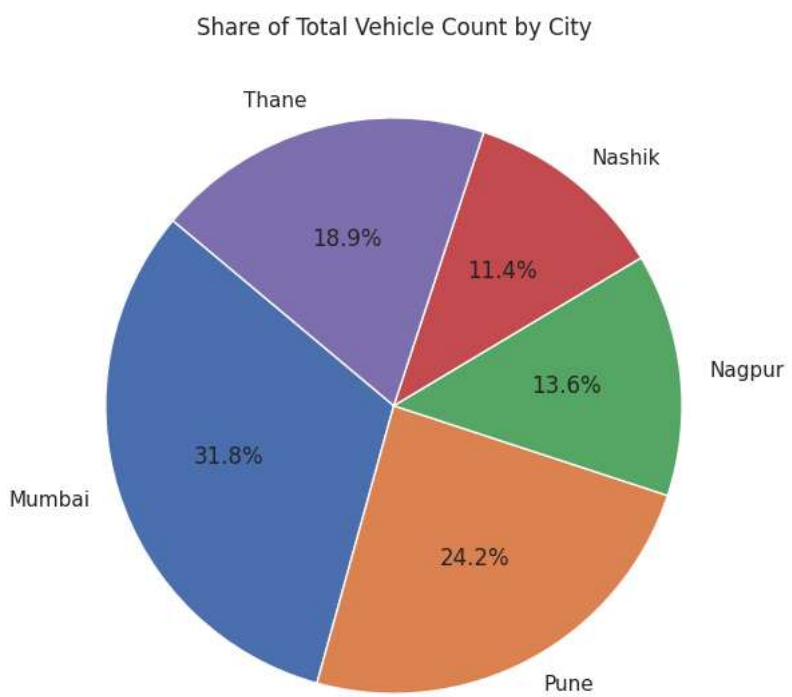



```
top_vehicles_per_km = df.sort_values('Vehicles_per_km', ascending=False).head(10)

plt.figure(figsize=(10, 5))
sns.barplot(x='City', y='Vehicles_per_km', data=top_vehicles_per_km, palette='Blues_d')
plt.title('Top 10 Cities by Vehicles per Road km')
plt.xticks(rotation=45)
plt.ylabel('Vehicles per km')
plt.show()
```

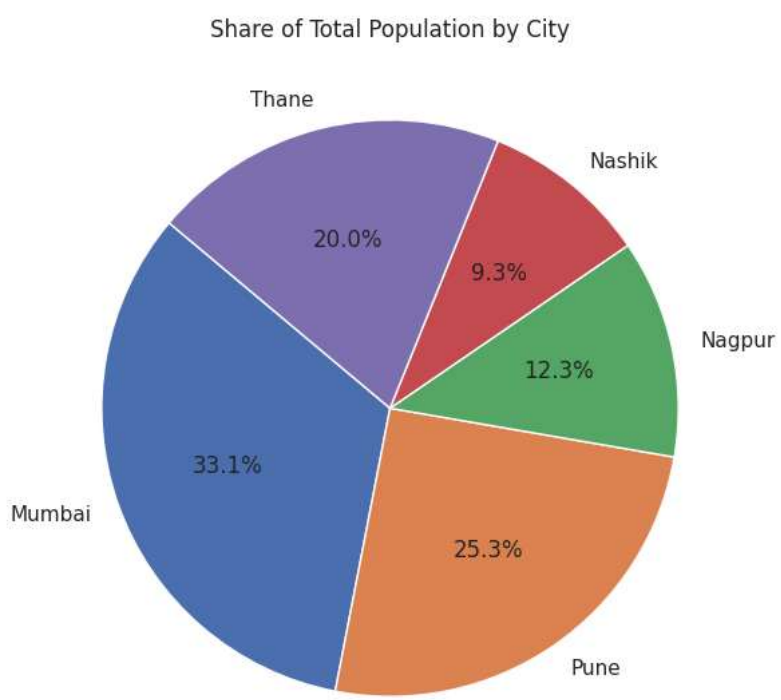
```
<ipython-input-88-aac4de060be9>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend
```



```
plt.figure(figsize=(7, 7))
plt.pie(df['Vehicle_Count'], labels=df['City'], autopct='%1.1f%%', startangle=140)
plt.title('Share of Total Vehicle Count by City')
plt.show()
```

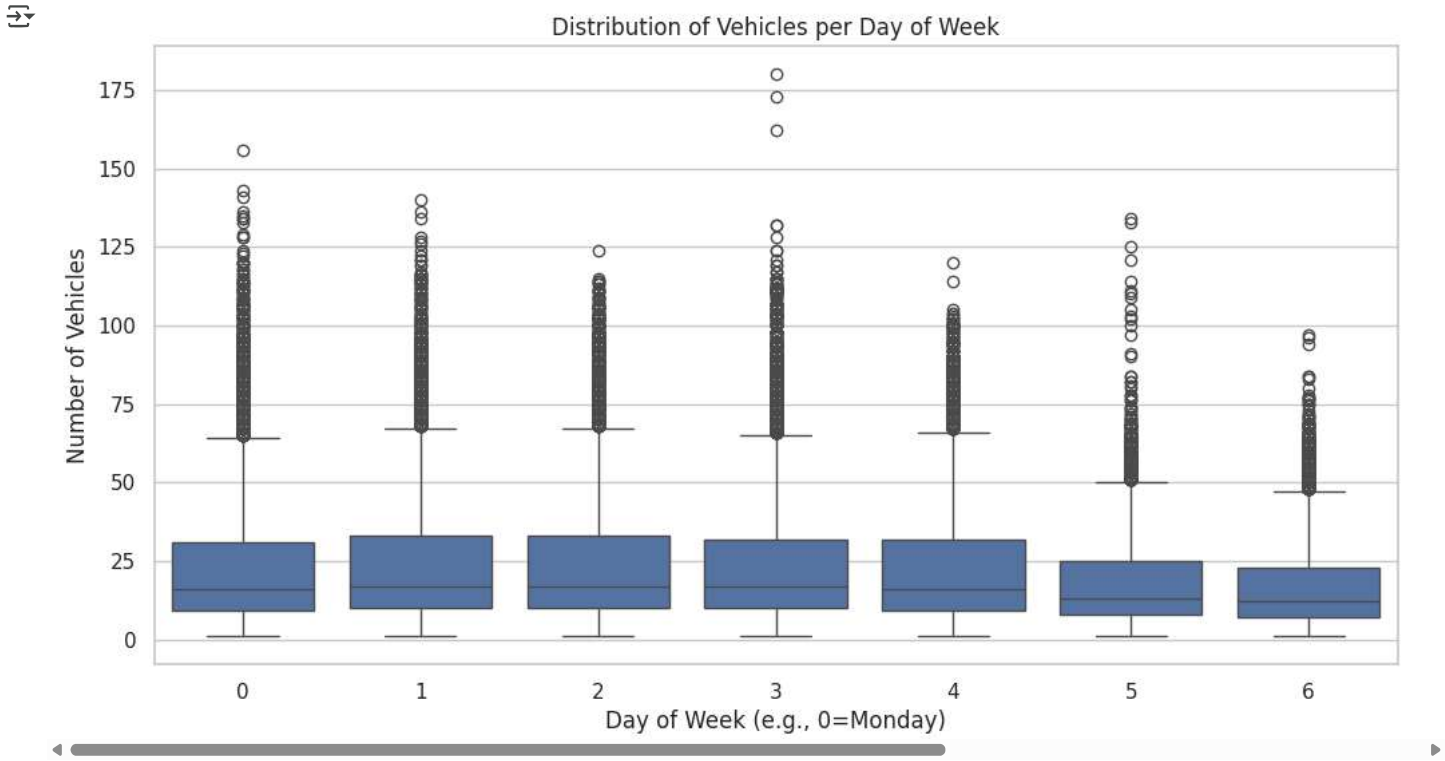



```
plt.figure(figsize=(7, 7))
plt.pie(df['Population'], labels=df['City'], autopct='%1.1f%%', startangle=140)
plt.title('Share of Total Population by City')
plt.show()
```

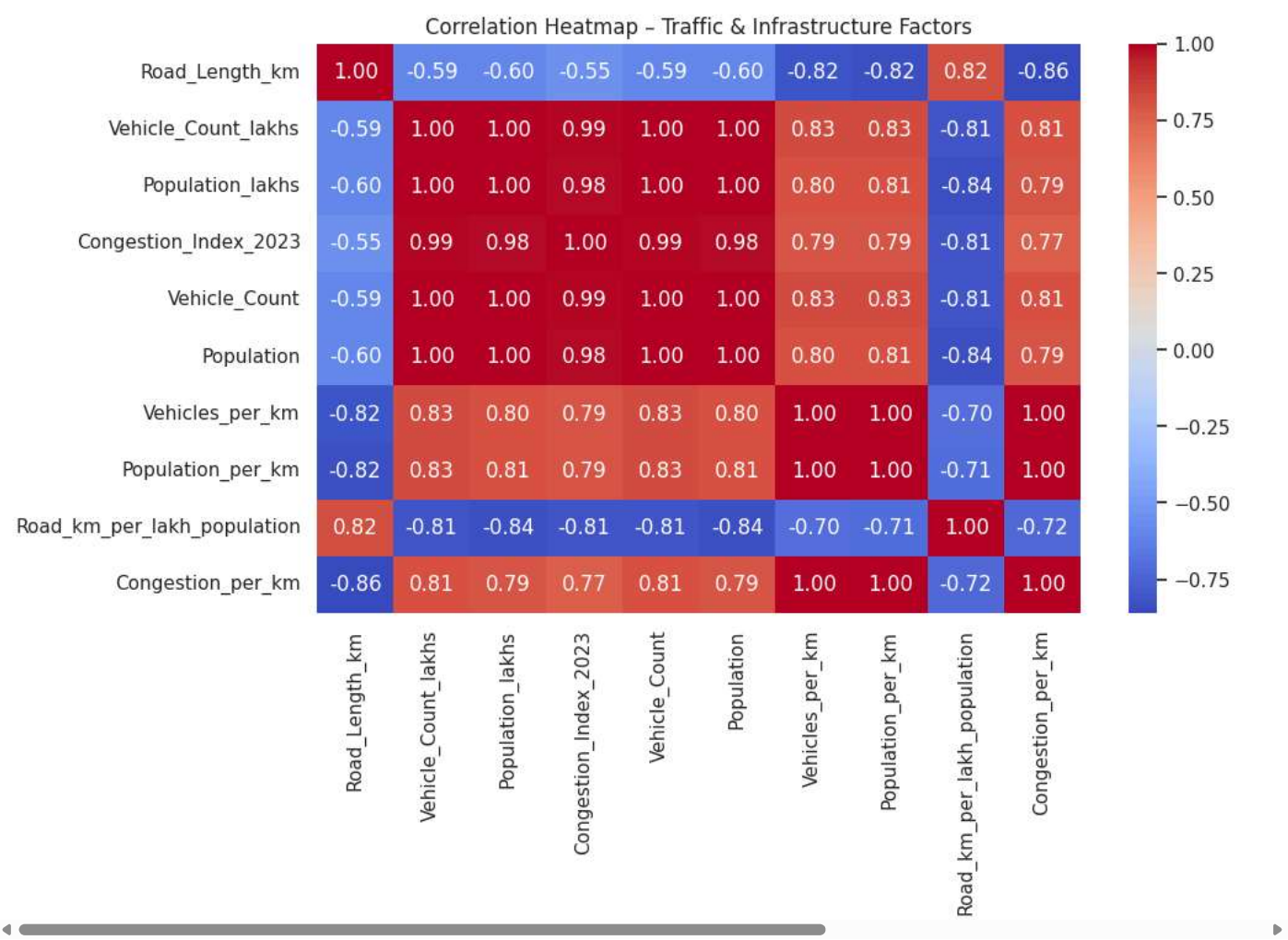


Double-click (or enter) to edit

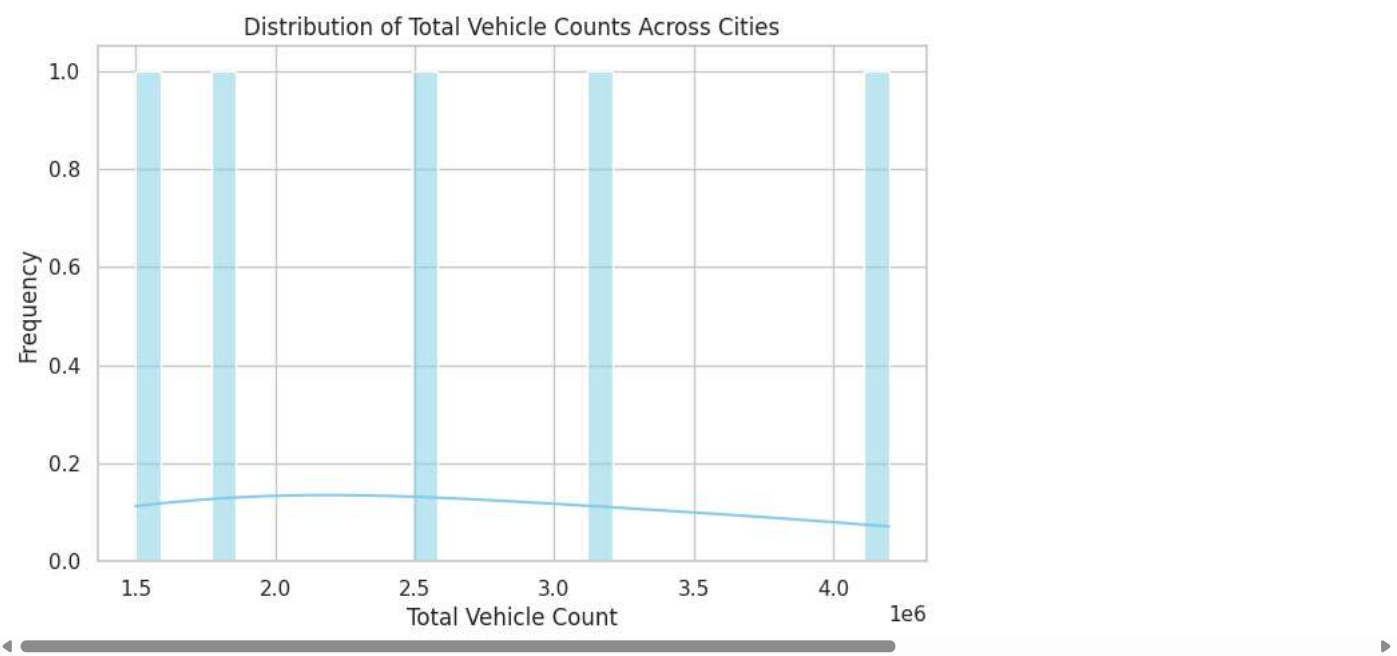
```
#Box and whiskers plot for each day
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='day_of_week', y='Vehicles')
plt.title('Distribution of Vehicles per Day of Week')
plt.xlabel('Day of Week (e.g., 0=Monday)')
plt.ylabel('Number of Vehicles')
plt.show()
```



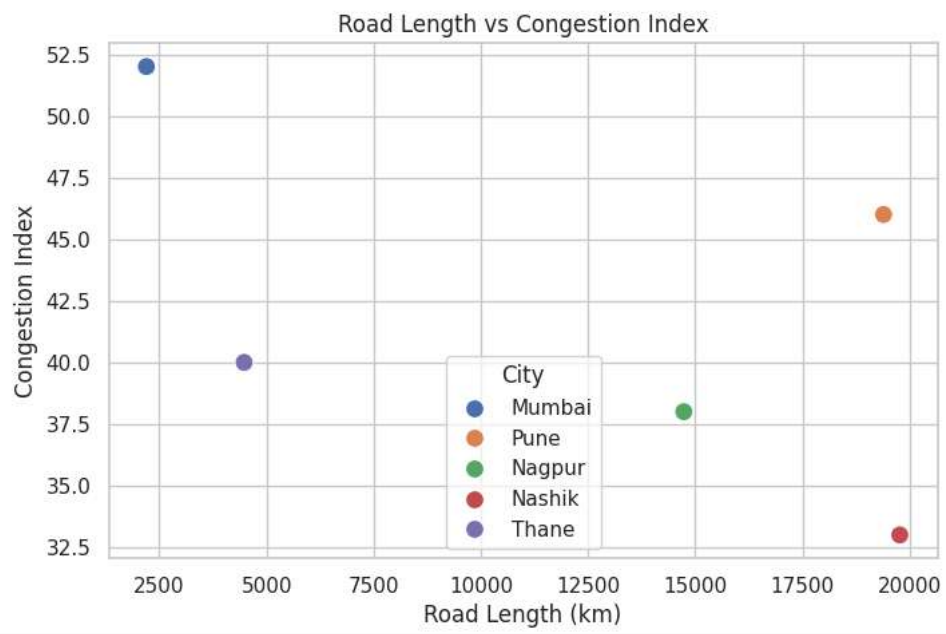

```
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap – Traffic & Infrastructure Factors')
plt.show()
```




```
#histogram
plt.figure(figsize=(8, 5))
sns.histplot(df['Vehicle_Count'], bins=30, kde=True, color='skyblue')
plt.title('Distribution of Total Vehicle Counts Across Cities')
plt.xlabel('Total Vehicle Count')
plt.ylabel('Frequency')
plt.show()
```



```
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='Road_Length_km', y='Congestion_Index_2023', hue='City', s=100)
plt.title('Road Length vs Congestion Index')
plt.xlabel('Road Length (km)')
plt.ylabel('Congestion Index')
plt.show()
```




```
# Features (independent variables)
X = df[['Road_Length_km', 'Vehicle_Count', 'Population', 'Vehicles_per_km',
        'Population_per_km', 'Road_km_per_lakh_population']]

# Target (dependent variable)
y = df['Congestion_Index_2023']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_pred = lr_model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error: {mae:.2f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Absolute Error: 24.98
Mean Squared Error: 624.14
```

```
R² Score: nan
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning:

R^2 score is not well-defined with less than two samples.
```