```
# importing required packages

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings(action = 'ignore')


def heading(info):
    print("\n\n##### {} #####".format(info))


# read the dataset
dataSet = pd.read_csv('Banglore_traffic_Dataset.csv', encoding = 'unicode_escape
```

```
# print info about the data
dataSet.info()
heading("Sample data points from the dataset")
dataSet.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8936 entries, 0 to 8935
Data columns (total 16 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Date                                  8936 non-null   object 
 1   Area Name                             8936 non-null   object 
 2   Road/Intersection Name                8936 non-null   object 
 3   Traffic Volume                        8936 non-null   int64  
 4   Average Speed                         8936 non-null   float64
 5   Travel Time Index                     8936 non-null   float64
 6   Congestion Level                      8936 non-null   float64
 7   Road Capacity Utilization             8936 non-null   float64
 8   Incident Reports                      8936 non-null   int64  
 9   Environmental Impact                  8936 non-null   float64
 10  Public Transport Usage                8936 non-null   float64
 11  Traffic Signal Compliance             8936 non-null   float64
 12  Parking Usage                         8936 non-null   float64
 13  Pedestrian and Cyclist Count          8936 non-null   int64  
 14  Weather Conditions                    8936 non-null   object 
 15  Roadwork and Construction Activity    8936 non-null   object 
dtypes: float64(8), int64(3), object(5)
memory usage: 1.1+ MB
```

```
##### Sample data points from the dataset #####
```

| | Date | Area Name | Road/Intersection Name | Traffic Volume | Average Speed | Travel Time Index | Congestic Leve |
|---|---|---|---|---|---|---|---|
| **0** | 2022-01-01 | Indiranagar | 100 Feet Road | 50590 | 50.230299 | 1.500000 | 100.0000( |
| **1** | 2022-01-01 | Indiranagar | CMH Road | 30825 | 29.377125 | 1.500000 | 100.0000( |
| **2** | 2022-01-01 | Whitefield | Marathahalli Bridge | 7399 | 54.474398 | 1.039069 | 28.3479( |
| **3** | 2022-01-01 | Koramangala | Sony World Junction | 60874 | 43.817610 | 1.500000 | 100.0000( |
| **4** | 2022-01-01 | Koramangala | Sarjapur Road | 57292 | 41.116763 | 1.500000 | 100.0000( |

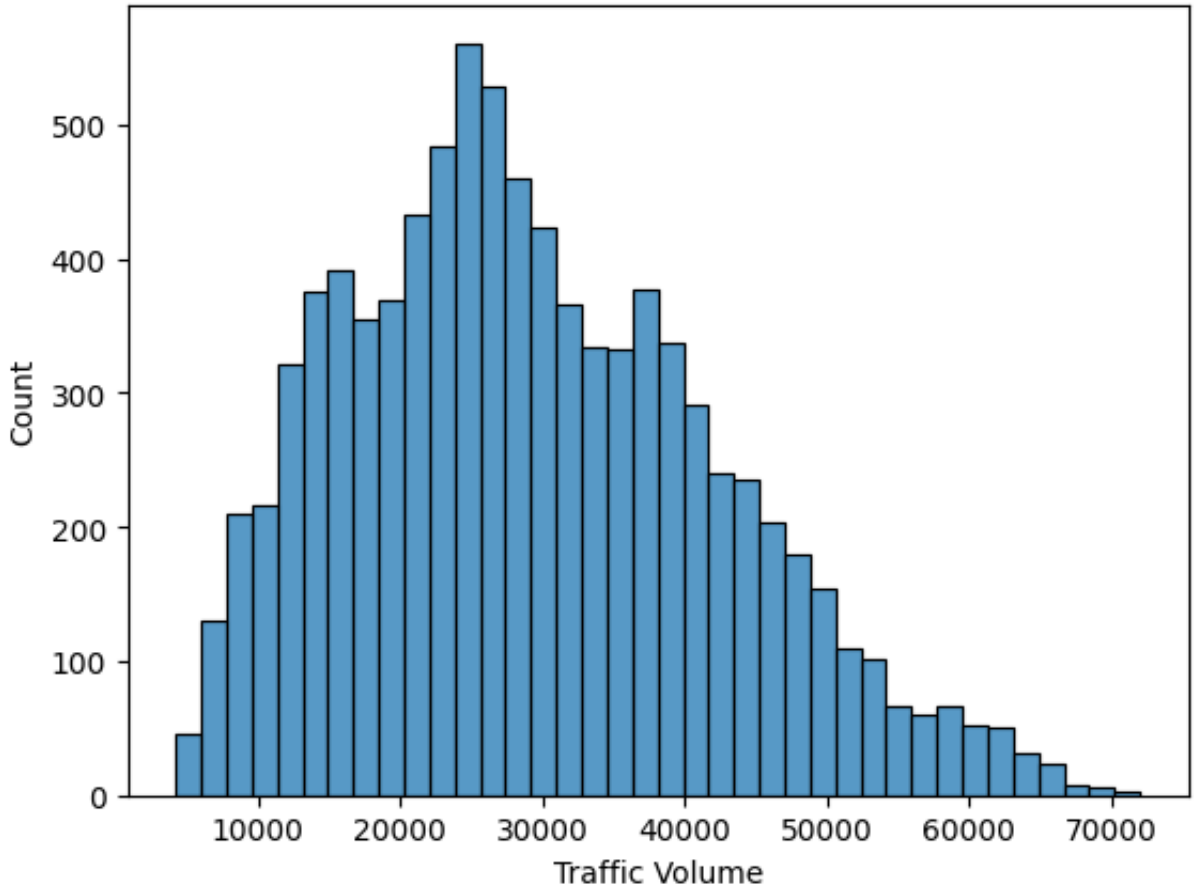
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

heading("Stats about numeric values")
print(dataSet.describe(include = "number"))
```

```
##### Stats about non-numeric values #####
              Date    Area Name Road/Intersection Name Weather Conditions
count         8936         8936                   8936                8936
unique         952            8                     16                   5
top     2023-01-24  Indiranagar          100 Feet Road               Clear
freq            15         1720                    860                5426

       Roadwork and Construction Activity
count                                8936
unique                                  2
top                                    No
freq                                 8054


##### Stats about numeric values #####
       Traffic Volume  Average Speed  Travel Time Index  Congestion Level
count     8936.000000    8936.000000        8936.000000       8936.000000
mean     29236.048120      39.447427           1.375554         80.818041
std      13001.808801      10.707244           0.165319         23.533182
min       4233.000000      20.000000           1.000039          5.160279
25%      19413.000000      31.775825           1.242459         64.292905
50%      27600.000000      39.199368           1.500000         92.389018
75%      38058.500000      46.644517           1.500000        100.000000
max      72039.000000      89.790843           1.500000        100.000000

       Road Capacity Utilization  Incident Reports  Environmental Impact  \
count                8936.000000       8936.000000           8936.000000
mean                   92.029215          1.570389            108.472096
std                    16.583341          1.420047             26.003618
min                    18.739771          0.000000             58.466000
25%                    97.354990          0.000000             88.826000
50%                   100.000000          1.000000            105.200000
75%                   100.000000          2.000000            126.117000
max                   100.000000         10.000000            194.078000

       Public Transport Usage  Traffic Signal Compliance  Parking Usage  \
count             8936.000000                8936.000000    8936.000000
mean                45.086651                  79.950243      75.155597
std                 20.208460                  11.585006      14.409394
min                 10.006853                  60.003933      50.020411
25%                 27.341191                  69.828270      62.545895
50%                 45.170684                  79.992773      75.317610
75%                 62.426485                  89.957358      87.518589
```

```
max                  79.979744                  99.993652      99.995049

       Pedestrian and Cyclist Count
count                   8936.000000
mean                     114.533348
std                       36.812573
min                       66.000000
25%                       94.000000
50%                      102.000000
75%                      111.000000
max                      243.000000
```

```
# lets first verify how to target variable is distributed
heading("Target variable \"Traffic volume\" distribution")
sns.histplot(data = dataSet, x = "Traffic Volume")
```

```
##### Target variable "Traffic volume" distribution #####
<Axes: xlabel='Traffic Volume', ylabel='Count'>
```



```
# lets convert the categorical values to numeric
def convert_categorical_to_numeric(dataframe, categorical_cols):

    for col in categorical_cols:
        if col in dataframe.columns:
            # create a mapping for the unique values in the column
            unique_values = dataframe[col].unique()
            value_mapping = {label: idx for idx, label in enumerate(unique_valu

            # apply the mapping to convert to numeric
            dataframe[col] = [value_mapping[val] for val in dataframe[col]]

    return dataframe

# leaving date column as of now and converting other columns

# we will backup the original dataset
originalDataset = dataSet.copy()

# select the relevant columns and convert them
columnsToConvert = ["Roadwork and Construction Activity","Weather Conditions","
dataSet = convert_categorical_to_numeric(dataSet, columnsToConvert)

heading("After conversion to numeric values")
print(dataSet[columnsToConvert].head())
```

```
##### After conversion to numeric values #####
   Roadwork and Construction Activity  Weather Conditions  Area Name  \
0                                   0                   0          0
1                                   0                   0          0
2                                   0                   0          1
3                                   0                   0          2
4                                   0                   0          2

   Road/Intersection Name
0                       0
1                       1
2                       2
3                       3
4                       4
```

```
# drop unrequired columns based on corelation matrix
dropThem = ["Public Transport Usage", "Traffic Signal Compliance", "Parking Usa
dataSet = dataSet.drop(columns=dropThem)

heading("Final dataset columns")
print(dataSet.head())
```

```
##### Final dataset columns #####
   Area Name  Road/Intersection Name  Traffic Volume  Average Speed  \
0          0                       0           50590      50.230299
1          0                       1           30825      29.377125
2          1                       2            7399      54.474398
3          2                       3           60874      43.817610
4          2                       4           57292      41.116763

   Travel Time Index  Congestion Level  Road Capacity Utilization  \
0           1.500000        100.000000                 100.000000
1           1.500000        100.000000                 100.000000
2           1.039069         28.347994                  36.396525
3           1.500000        100.000000                 100.000000
4           1.500000        100.000000                 100.000000

   Incident Reports  Environmental Impact  Pedestrian and Cyclist Count
0                 0               151.180                           111
1                 1               111.650                           100
2                 0                64.798                           189
3                 1               171.748                           111
4                 3               164.584                           104
```

```
# seperate the input and target columns into numpy arrays
if isinstance(dataSet, pd.DataFrame):
    dataSet = dataSet.to_numpy()
# print(dataset)
X = dataSet[:, [0, 1, 3, 4,5,6,7,8,9]]
Y = dataSet[:, 2]

# adding extra column for intercepts
X = np.hstack((np.ones((X.shape[0], 1)), X))
heading("Printing X and Y variables for the model")
print(X[:5])
print(Y[:5])
```

```
##### Printing X and Y variables for the model #####
[[  1.           0.           0.          50.23029856   1.5
  100.         100.           0.         151.18       111.        ]
 [  1.           0.           1.          29.37712471   1.5
  100.         100.           1.         111.65       100.        ]
 [  1.           1.           2.          54.47439821   1.03906885
   28.34799386  36.39652494   0.          64.798      189.        ]
 [  1.           2.           3.          43.81761039   1.5
  100.         100.           1.         171.748      111.        ]
 [  1.           2.           4.          41.11676289   1.5
  100.         100.           3.         164.584      104.        ]]
[50590. 30825.  7399. 60874. 57292.]
```

```
# shuffle the datasets
indices = np.arange(X.shape[0])
np.random.shuffle(indices)
#
X_shuffled = X[indices]
Y_shuffled = Y[indices]

# split the dataset into 80:20
split_ratio = 0.1
split_index = int(len(X_shuffled) * split_ratio)

X_train = X_shuffled[:split_index]
Y_train = Y_shuffled[:split_index]

X_test = X_shuffled[split_index:]
Y_test = Y_shuffled[split_index:]

print("Training set samples: ", X_train.shape[0])
print("Testing set samples: ", X_test.shape[0])
```

```
Training set samples:  893
Testing set samples:  8043
```

```
# we solve the least squares problem using the normal equation
# (X^T * X)^(-1) * (X^T * y)
weights = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ Y_train
heading("Training on linear regression with given dataset, ideal weights of the
print(weights[:5])
```

```
##### Training on linear regression with given dataset, ideal weights of th
[-2.50000000e+04  2.53820076e-09 -1.07382903e-09  1.37436729e-11
  7.53152563e-09]
```

```
# lets predict values
Y_pred = X_test @ weights

# we will use mean absolute percentage error to calculate the error percentage
MAPE = np.mean(np.abs((Y_test - Y_pred) / Y_test)) * 100

heading("Printing the MAPE and first 10 predictions with actual values")
print("MAPE: {} %".format(MAPE))
for i in range(10):
    print("\nPredicted value: {0} \t Actual value: {1}".format(Y_pred[i], Y_tes
```

```
##### Printing the MAPE and first 10 predictions with actual values #####
MAPE: 1.3428061131584133e-11 %

Predicted value: 40229.99999999769      Actual value: 40230.0

Predicted value: 19346.999999999585     Actual value: 19347.0

Predicted value: 18413.99999999702      Actual value: 18414.0

Predicted value: 43671.99999999871      Actual value: 43672.0

Predicted value: 27533.9999999988       Actual value: 27534.0

Predicted value: 42152.9999999955       Actual value: 42153.0

Predicted value: 52018.999999997235     Actual value: 52019.0

Predicted value: 25665.99999999832      Actual value: 25666.0

Predicted value: 22348.999999995027     Actual value: 22349.0

Predicted value: 20295.999999999753     Actual value: 20296.0
```