```
# 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	Congestio 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(

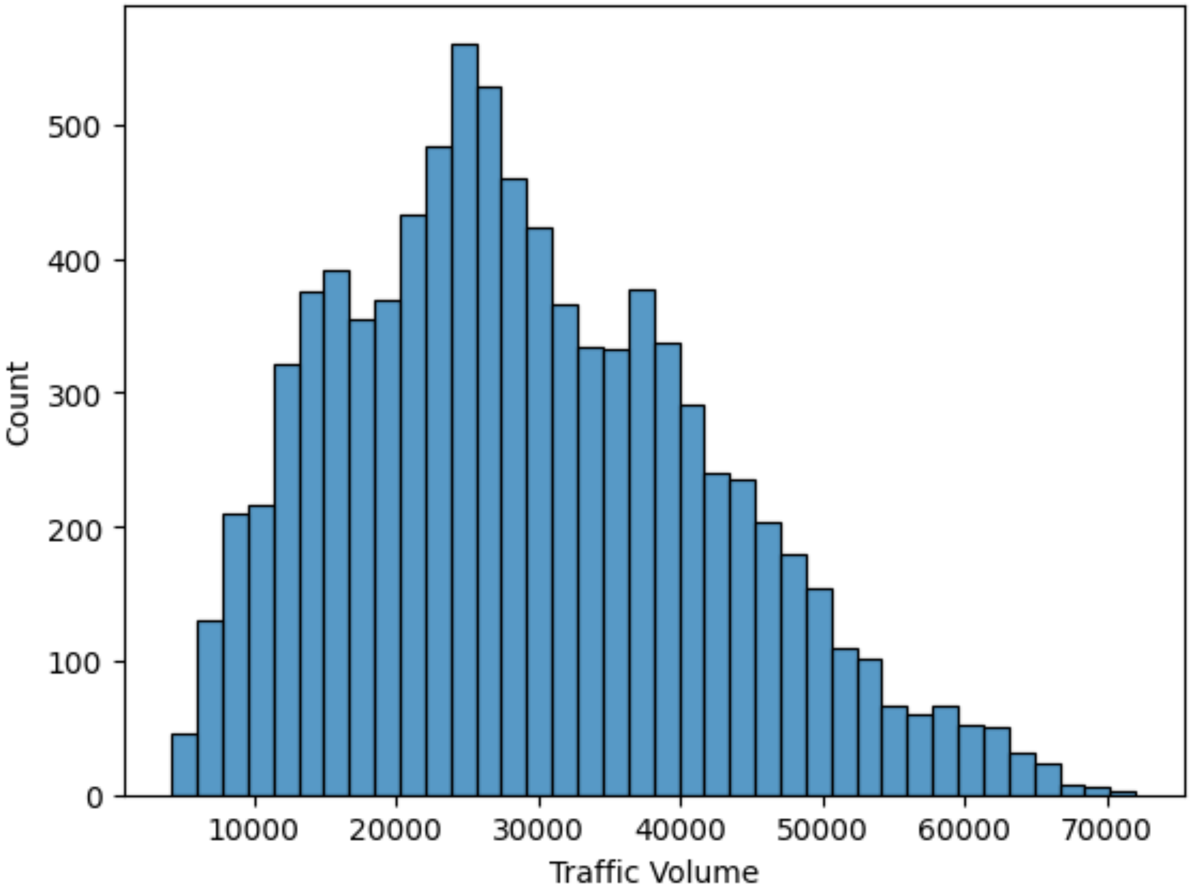
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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
```

```
def sigmoid(z):
    z = np.clip(z, -500, 500)  # Prevent overflow
    return 1 / (1 + np.exp(-z))

def binary_cross_entropy(y, y_hat):
    n = len(y)
    loss = -np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
    return loss
```

```
class LogisticRegression:
    def __init__(self, learning_rate=0.001, n_iters=100000):
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Initialize weights and bias
        self.weights = np.zeros(n_features)
        self.bias = 0

        # Gradient descent
        for _ in range(self.n_iters):
            # Linear model: X @ w + b
            linear_model = np.dot(X, self.weights) + self.bias
            # Apply sigmoid to get predictions
            y_hat = sigmoid(linear_model)

            # Compute gradients
            dw = (1 / n_samples) * np.dot(X.T, (y_hat - y))
            db = (1 / n_samples) * np.sum(y_hat - y)

            # Update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict_proba(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        return sigmoid(linear_model)

    def predict(self, X):
        y_hat = self.predict_proba(X)
        return np.where(y_hat >= 0.5, 1, 0)
```

```
class Norm:
    def normalise(self,X):
        # Remove nan columns
        X = X[:, ~np.isnan(X).any(axis=0)]

        self.mean = np.mean(X)
        self.std_dev = np.std(X)
        return (X - self.mean) / (self.std_dev + 1e-8) # Avoid division by 0

    def denormalize(self, X):
        return X*self.std_dev+ self.mean


model = LogisticRegression(learning_rate=0.1, n_iters=1000)

Xnorm = Norm()
Ynorm = Norm()

#Normalizing data
X = Xnorm.normalise(X_train)
y = Ynorm.normalise(Y_train).flatten()

print(X)
print(y)
model.fit(X,y)
```

```
[[0.         0.57142857 0.53333333 ... 0.14285714 0.32646079 0.12209302]
 [0.         1.         0.93333333 ... 0.14285714 0.2414978  0.11046512]
 [0.         0.         0.         ... 0.         0.11180427 0.83139535]
 ...
 [0.         0.28571429 0.2        ... 0.14285714 0.41230865 0.20930233]
 [0.         0.71428571 0.6        ... 0.28571429 0.36079403 0.15697674]
 [0.         0.28571429 0.26666667 ... 0.71428571 0.42329587 0.13953488]]
[0.32646079 0.2414978  0.11180427 0.34078105 0.09721854 0.73730938
 0.34570687 0.32314249 0.16725658 0.38002537 0.07070171 0.16551633
 0.2819367  0.19754889 0.04254786 0.36271126 0.7068401  0.32236085
 0.64930832 0.53535085 0.20747426 0.4118957  0.15066513 0.34364216
 0.54183996 0.06565791 0.57587824 0.39966965 0.39222193 0.38296021
 0.32991181 0.41124679 0.45137598 0.2812288  0.06792909 0.18066248
 0.26562546 0.09789694 0.29919181 0.62324868 0.45976757 0.33842138
 0.85325782 0.72980267 0.48423443 0.18830192 0.03170811 0.23617379
 0.58747014 0.53532136 0.45849925 0.16287644 0.43571365 0.13136006
 0.84468926 0.72248769 0.38897738 0.61363301 0.08670324 0.58398962
 0.24396071 0.16578179 0.02375896 0.49780255 0.49380586 0.729257
 0.24963868 0.17575141 0.63178775 0.09606819 0.12942807 0.64691915
 0.42586202 0.86256378 0.21433207 0.40767779 0.42214553 0.26513878
 0.15727222 0.36120697 0.51871516 0.63366074 0.34477775 0.36628027
 0.63162552 0.22509807 0.58452054 0.17752116 0.33961596 0.08255907
 0.38309294 0.4938796  0.17868625 0.20163407 0.36514468 0.39832758
```

```
0.02368522 0.67635607 0.30357196 0.43033065 0.44010854 0.15752293
0.06555467 0.19946612 0.34719641 0.29823319 0.16290594 0.43152523
0.2765242  0.19664926 0.17281656 0.68554405 0.42800047 0.58391588
0.35409846 0.31975046 0.36017462 0.46332183 0.60083178 0.25075952
0.20296139 0.51162139 0.49561986 0.20654514 0.61494558 0.49867268
0.18943751 0.51558859 0.34797806 0.70368404 0.56602661 0.3195145
0.34566263 0.42805946 0.30163997 0.19091231 0.46013627 0.12833673
0.34865646 0.39077663 0.61693655 0.30264283 0.39412441 0.38720762
0.30314426 0.53629472 0.76763118 0.64206707 0.65063564 0.35771171
0.5721765  0.22496534 0.4704156  0.32700646 0.31163909 0.49100375
0.20467215 0.1628027  0.27469545 0.10764534 0.67025042 0.29845441
0.40188184 0.47752411 0.29224552 0.53661918 0.18439371 0.72682358
0.49101849 0.72838687 0.32497124 0.31777424 0.32535469 0.20840339
0.29391204 0.20807893 0.31731705 0.31068047 0.39643984 0.46702357
0.57947674 0.8086895  0.06346046 0.18743179 0.18172433 0.666062
0.37703153 0.19318349 0.41792762 0.51870041 0.05874111 0.30568091
0.43269032 0.33163732 0.75606878 0.43534495 0.64172787 0.63317406
0.32582662 0.4707843  0.22835737 0.66904109 0.50355426 0.35744624
0.20741527 0.68514586 0.04011444 0.40158688 0.2606259  0.44783647
0.49334867 0.34165118 0.23710291 0.18184231 0.52098634 0.20840339
0.74878329 0.18066248 0.47495797 0.50184349 0.67616435 0.05613073
0.49783205 0.25183612 0.43990207 0.39195646 0.17595788 0.40198508
0.50445388 0.46941274 0.28602189 0.12146418 0.46832139 0.0739905
0.56592337 0.35875881 0.1822995  0.19343421 0.26817686 0.49581158
0.1393387  0.42328113 0.27381058 0.46873433 0.64045955 0.18547031
0.82275905 0.69790284 0.42732207 0.72524555 0.43242486 0.30274607
0.58727841 0.61696605 0.13702327 0.50016223 0.03723859 0.14132968
0.8584786  0.34533817 0.32107778 0.07337109 0.19123676 0.36408282
0.28758517 0.19269681 0.14221455 0.21005516 0.52760818 0.31379229
0.16758104 0.20167832 0.5873374  0.22782645 0.16169661 0.45711294
0.12512167 0.66004483 0.42218978 0.23691119 0.51828747 0.48106362
0.05436097 0.49088576 0.26010972 0.25794178 0.35927499 0.65762617
0.26477008 0.59064095 0.63162552 0.59002153 0.1747338  0.5177123
0.6981683  0.58745539 0.3967643  0.03133941 0.32085656 0.66952777
0.34769784 0.62279149 0.63908799 0.51377459 0.44528508 0.32551692
0.61047695 0.28248238 0.39023095 0.34032387 0.30541545 0.26220393
```

```
X_nt = Xnorm.normalise(X_test)

Y_pred = model.predict_proba(X_nt)
Y_pred = Ynorm.denormalize(Y_pred)
print("Predictions:", Y_pred)

#Error in terms of probabilities
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
```

```
Predictions: [12422.33504507 15768.9228721  36283.27024499 ... 37970.861782
 20606.22732013 29920.14654645]


##### Printing the MAPE and first 10 predictions with actual values #####
MAPE: 16.81712872415161 %

Predicted value: 12422.335045071459      Actual value: 12973.0

Predicted value: 15768.922872096924      Actual value: 9272.0

Predicted value: 36283.27024498963       Actual value: 36313.0

Predicted value: 43321.16507847105       Actual value: 47890.0

Predicted value: 45759.81734744462       Actual value: 50330.0

Predicted value: 39945.41896939413       Actual value: 37482.0

Predicted value: 38439.589876817         Actual value: 35875.0

Predicted value: 18823.14782435385       Actual value: 16942.0

Predicted value: 13108.465452439883      Actual value: 7382.0

Predicted value: 11749.400955343903      Actual value: 7929.0
```