+ Code + Text

```
# importing required packages

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.manifold import Isomap
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
%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
```

Github—
https://github.com/nitin-rajesh/MLProject-Traffic-Prediction

```
# 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(

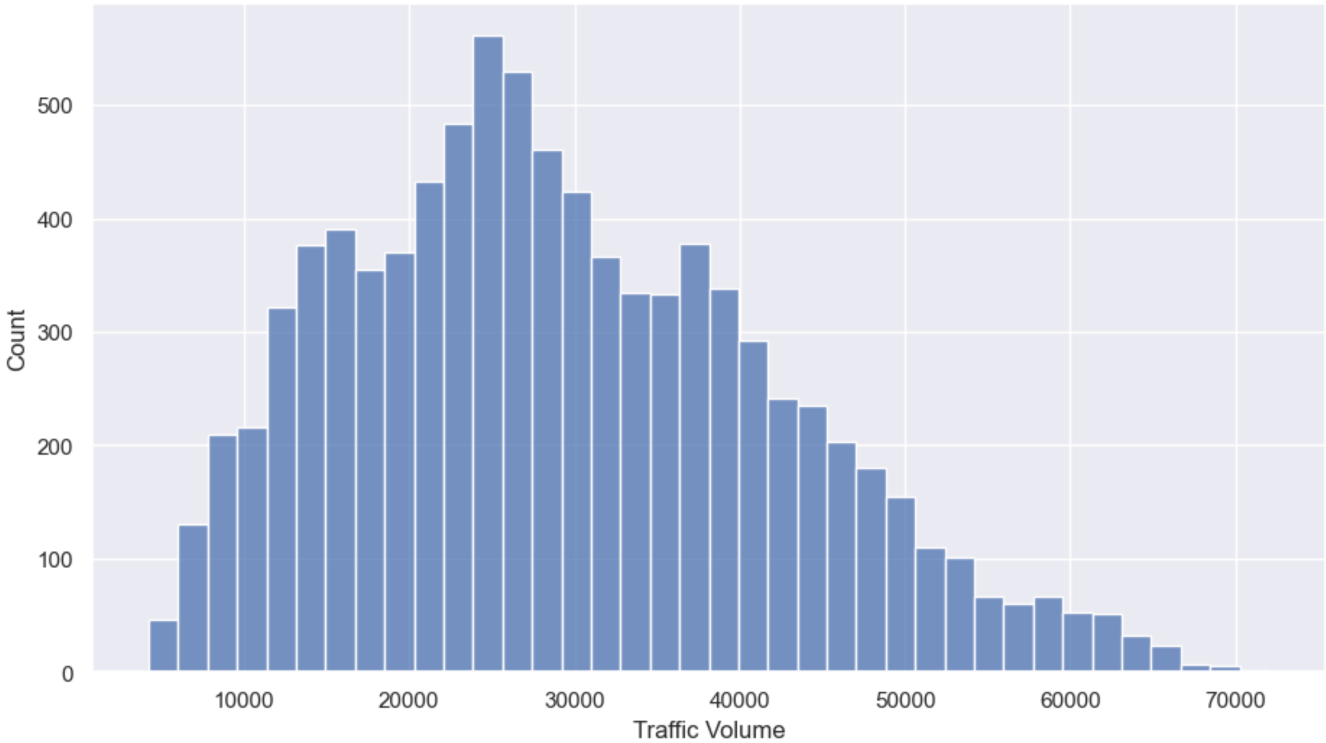
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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
```

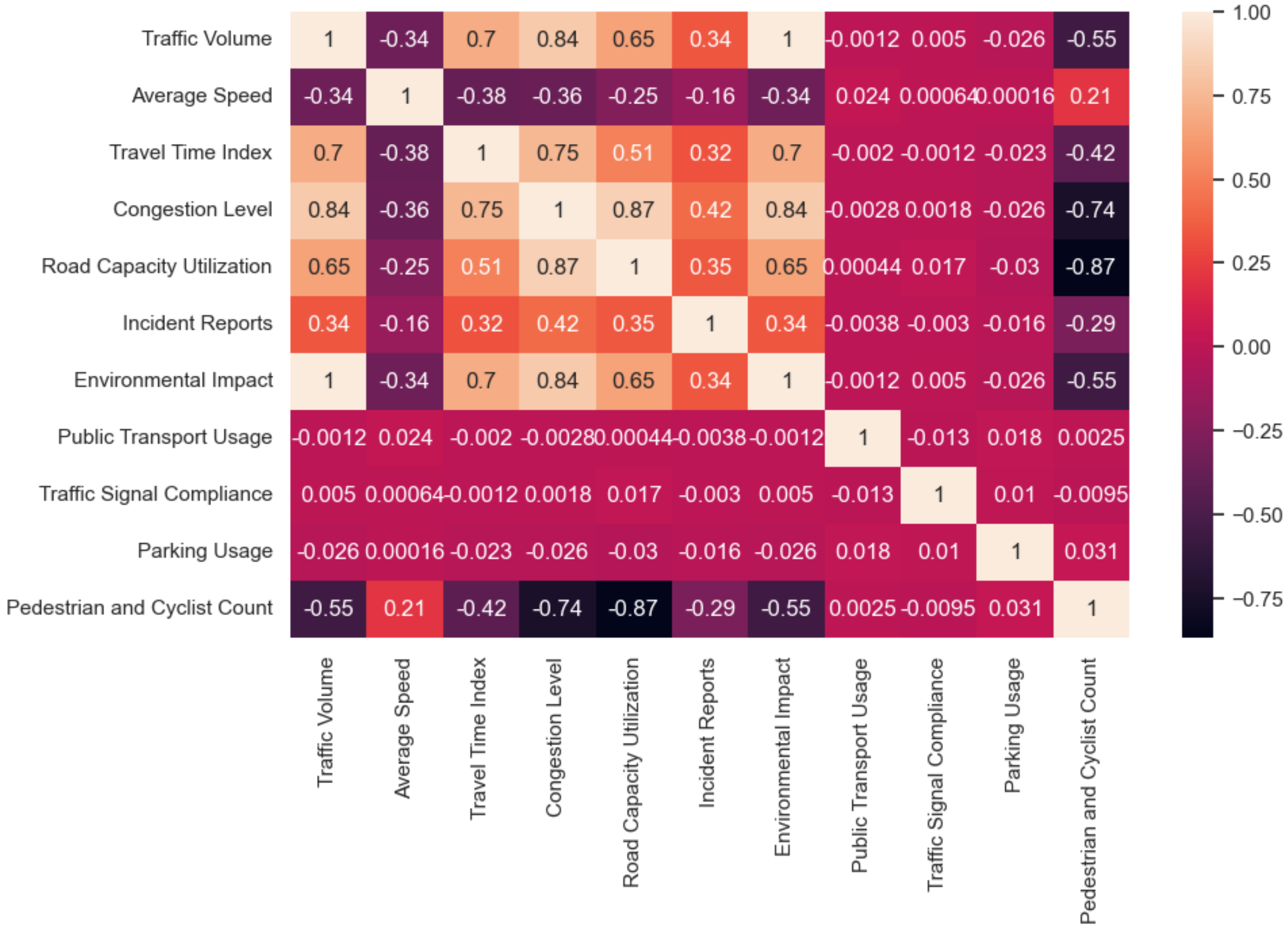
```
# lets first verify how to target variable is distributed
heading("Target variable \"Traffic volume\" distribution")
sns.histplot(data = dataSet, x = "Traffic Volume")
```

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



```
# lets also plot the corelation heatmap to analyze how to variables are co-rela
sns.set(rc = {'figure.figsize': (10,6)})
corelations = dataSet.select_dtypes(include = "number").corr()
sns.heatmap(corelations, annot = True)
```

<Axes: >



Some observations

general dataset

- total 8936 values
- 15 input variables, 1 target variable
- none of them have nan/null values
- 5 non numeric variables, 3 integer variables, 8 floats

non numeric

- date has 952 unique values
- rest have 8,16,5 and2 unique values
- need to be converted into numeric type

misc

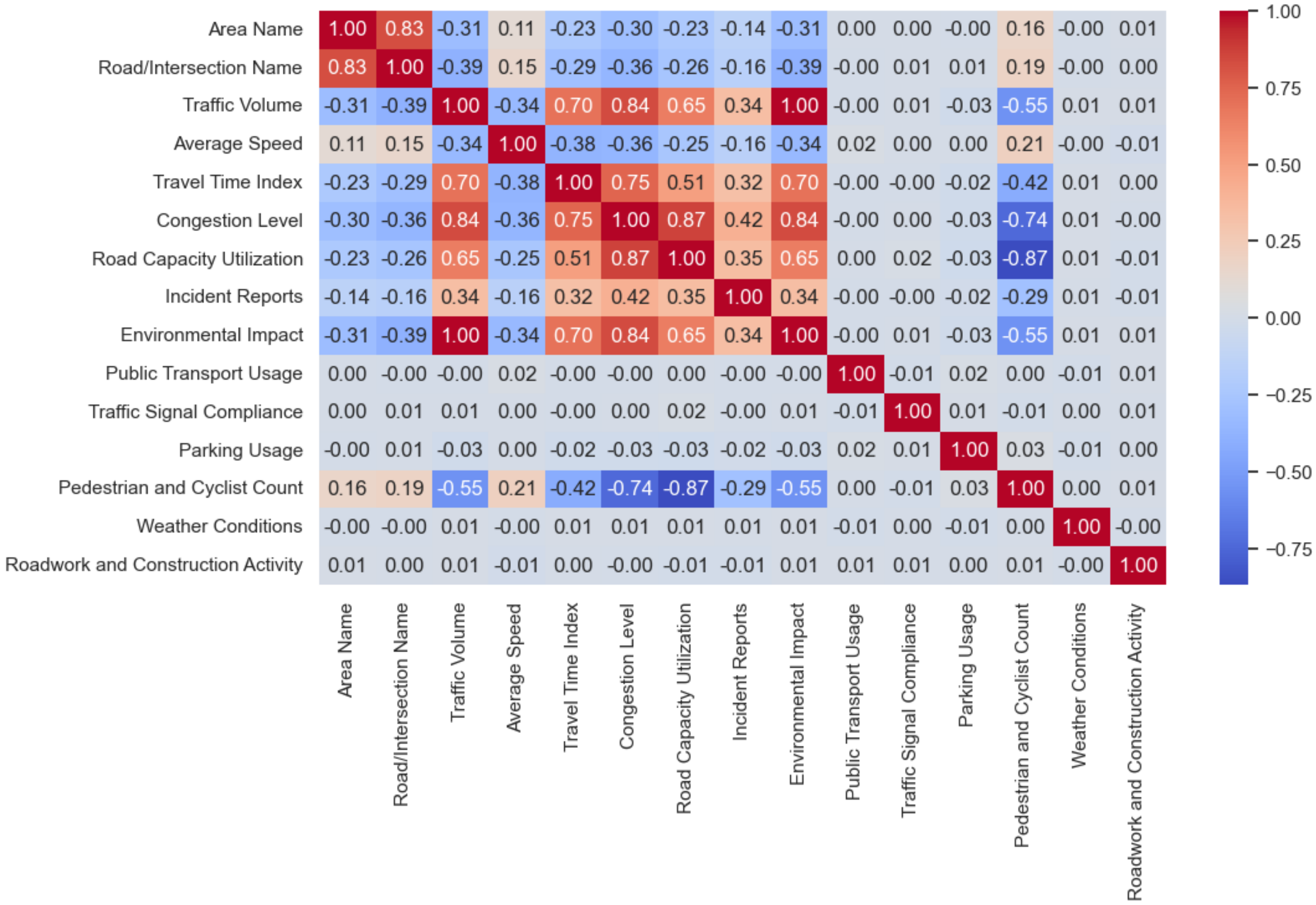
- target variable is little skewd version of normal distribution
- From the corelation heatmap above: Public Transport Usage, Traffic Signal Compliance, Parking Usage have very less corelation with the target variable

```
# 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
```

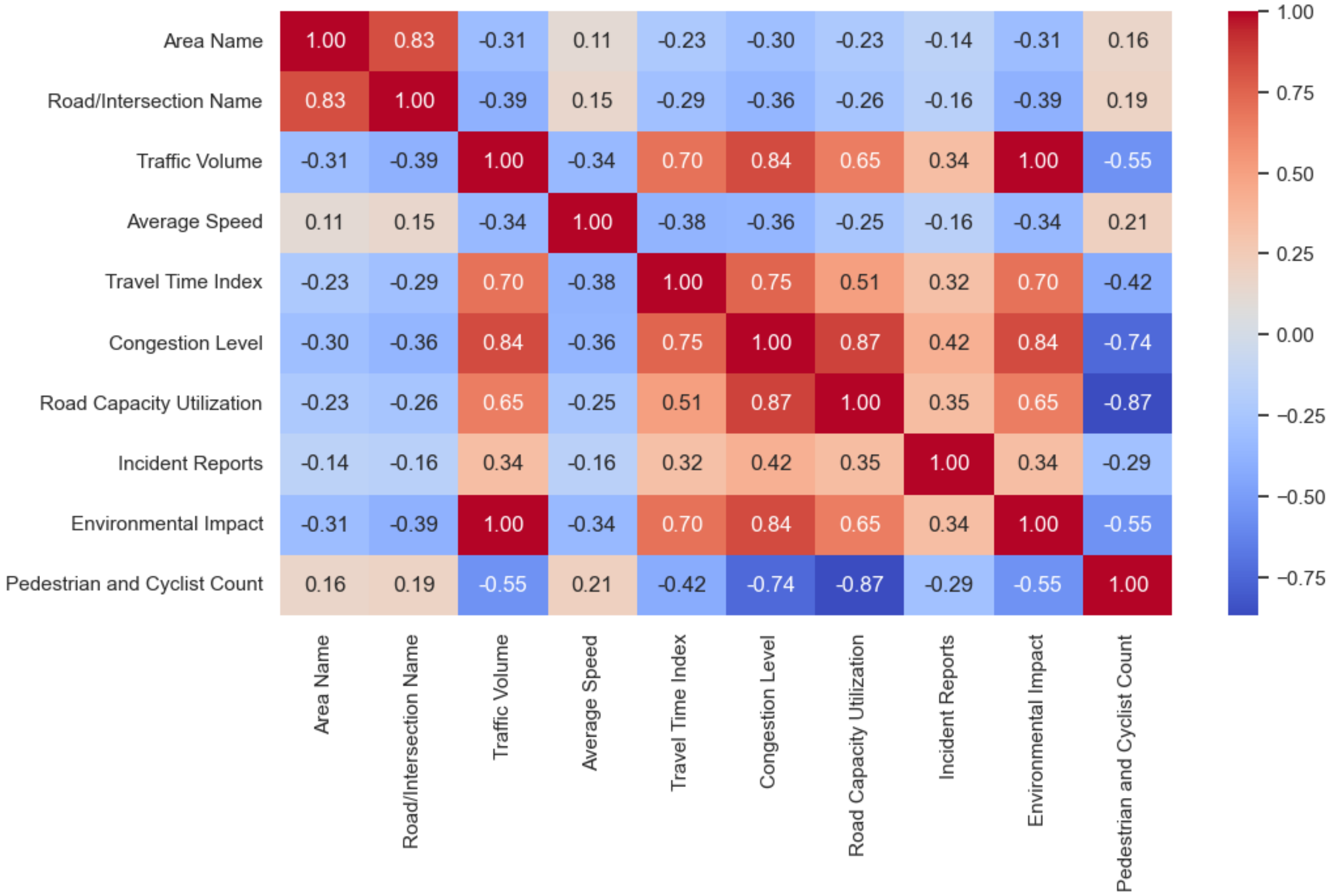
```
# lets again plot the corelation heatmap to analyze how all the variables are c
sns.set(rc = {'figure.figsize': (11,6)})
corelations = dataSet.select_dtypes(include = "number").corr()
sns.heatmap(corelations, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
```

```
<Axes: >
```



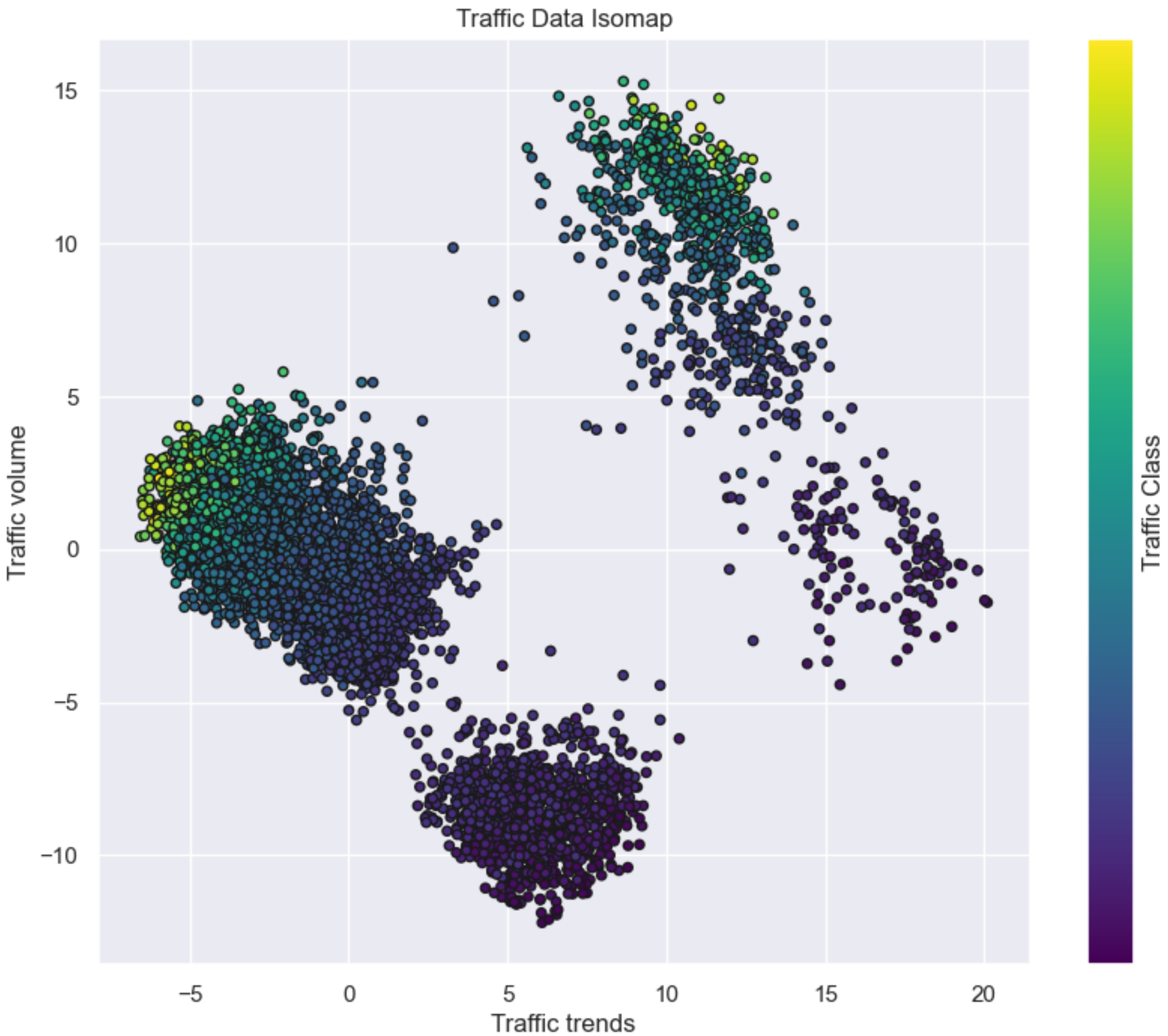
```
# we will drop certain columns that do not corelate to our target variable
dropThem = ["Public Transport Usage", "Traffic Signal Compliance", "Parking Usa
corelations = corelations.drop(columns=dropThem, index=dropThem)
sns.heatmap(corelations, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
```

```
<Axes: >
```




```
y = dataSet["Traffic Volume"]
dataSet = dataSet.drop(columns=dropThem)


dataSet = convert_categorical_to_numeric(dataSet,['Date'])
X = dataSet.drop(columns=["Traffic Volume"])
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

reduction_map = Isomap(n_neighbors=X.shape[1], n_components=2)
X_reduced = reduction_map.fit_transform(X_std)
```

```
plt.figure(figsize=(10, 8))
scatterplot = plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis'
plt.title('Traffic Data Isomap')
plt.xlabel('Traffic trends')
plt.ylabel('Traffic volume')
plt.colorbar(scatterplot, ticks=np.arange(10), label='Traffic Class')

plt.show()
```


Traffic Data Isomap
Traffic volume
Traffic trends
Traffic Class
15
10
5
0
−5
−10
−5
0
5
10
15
20

```
# 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	100.00000
1	2022-01-01	Indiranagar	CMH Road	30825	29.377125	1.500000	100.00000
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	100.00000
4	2022-01-01	Koramangala	Sarjapur Road	57292	41.116763	1.500000	100.00000

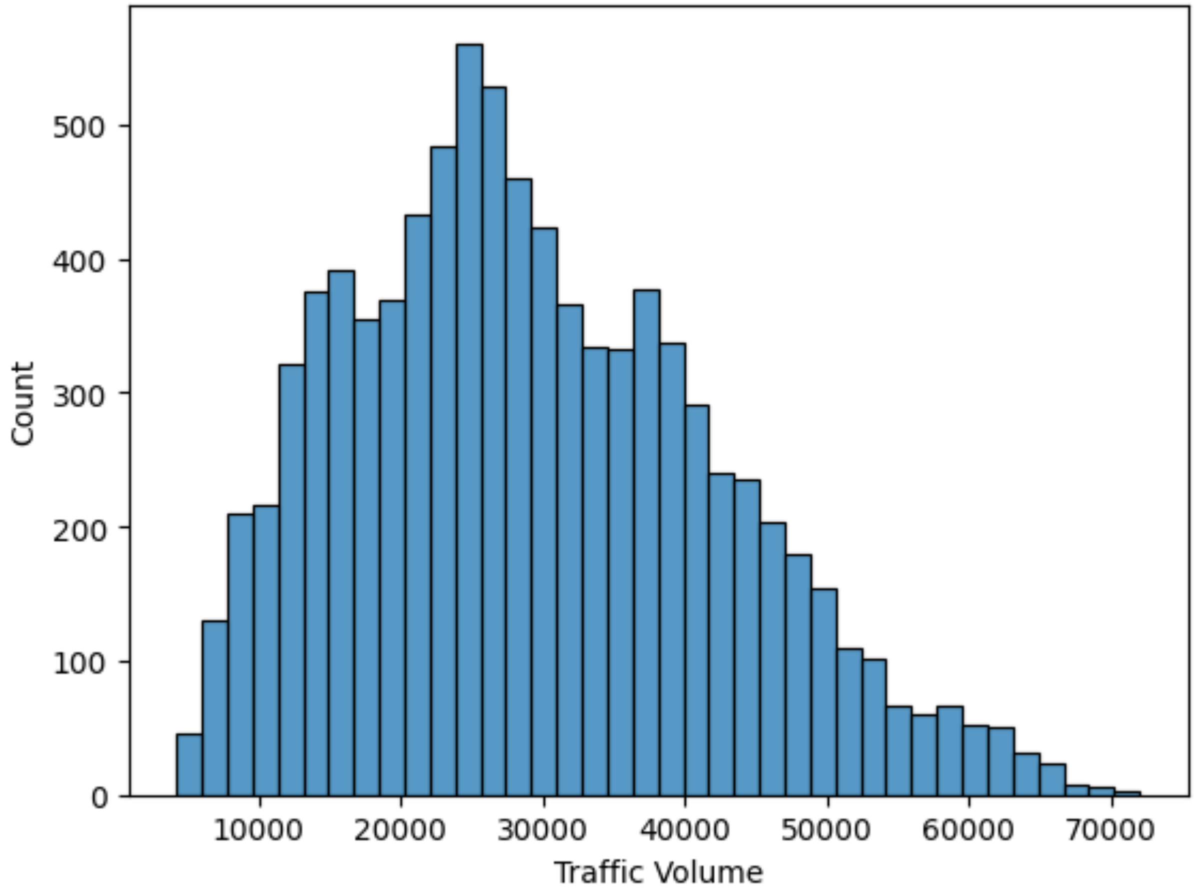
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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
```

```
# Mean square error function
def mse(y):
    return np.mean((y - np.mean(y)) ** 2)

# Compute weighted MSE on both sides
def mse_split(left, right):
    n_left, n_right = len(left), len(right)
    total = n_left + n_right
    return (n_left / total) * mse(left) + (n_right / total) * mse(right)

# Split data based on feature
def split_data(data, feature, value):
    left = data[data[feature] <= value]
    right = data[data[feature] > value]
    return left, right


class DecisionTreeRegression:
    def __init__(self, max_depth=3):
        self.max_depth = max_depth
        self.tree = None

    def fit(self, data, depth=0):
        # Stop splitting if max depth is reached or data is too small
```

```
        if depth >= self.max_depth or len(data) <= 1:
            return np.mean(data['y'])  # Leaf node

        # Initialize variables to track the best split
        best_feature, best_value = None, None
        best_mse = float('inf')
        best_left, best_right = None, None

        # Iterate over all features and values to find the best split
        for feature in data.columns[1:]:  # Exclude 'y'
            for value in data[feature].unique():
                left, right = split_data(data, feature, value)
                if len(left) == 0 or len(right) == 0:
                    continue  # Skip invalid splits

                current_mse = mse_split(left['y'], right['y'])

                # Update the best split if the current one is better
                if current_mse < best_mse:
                    best_mse = current_mse
                    best_feature, best_value = feature, value
                    best_left, best_right = left, right

        # Create a decision node with the best split
        self.tree = {
            'feature': best_feature,
            'value': best_value,
            'left': self.fit(best_left, depth + 1),
            'right': self.fit(best_right, depth + 1)
        }
        return self.tree

    def predict_one(self, x, node=None):
        # Predict the value for a single example by traversing the tree
        if node is None:
            node = self.tree

        if isinstance(node, (int, float)):  # If it's a leaf node
            return node

        if x[node['feature']] <= node['value']:
            return self.predict_one(x, node['left'])
        else:
            return self.predict_one(x, node['right'])

    def predict(self, X):
        # Predict for all examples in the dataset
        return X.apply(lambda row: self.predict_one(row), axis=1)
```

```
# Initialize the model
model = DecisionTreeRegression(max_depth=11)

decisionTreeData = pd.DataFrame({'y':Y_train})

for i in range(X_train.shape[1]):
    decisionTreeData['X'+str(i)] = X_train[:,i]

print(decisionTreeData.columns[1:])
print(decisionTreeData.shape)
# Train the model
model.fit(decisionTreeData)
print("Trained Tree:", model.tree)
```

```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9'], dtype='
(893, 11)
Trained Tree: {'feature': 'X8', 'value': np.float64(114.598), 'left': {'fea
```

```
# Make predictions on the dataset
decisionTreeTest = pd.DataFrame()

for i in range(X_test.shape[1]):
    decisionTreeTest['X'+str(i)] = X_test[:,i]

Y_pred = model.predict(decisionTreeTest)
print("Predictions:", predictions.values)

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: [50933. 58808. 36368. ... 26241. 28878. 44414.]


##### Printing the MAPE and first 10 predictions with actual values #####
MAPE: 0.5898035372180306 %

Predicted value: 50933.0        Actual value: 50896.0

Predicted value: 58808.0        Actual value: 59040.0

Predicted value: 36368.0        Actual value: 36281.0

Predicted value: 50908.0        Actual value: 51290.0

Predicted value: 41757.0        Actual value: 41705.0

Predicted value: 11172.0        Actual value: 11347.0

Predicted value: 21337.0        Actual value: 20987.0

Predicted value: 7525.0         Actual value: 7358.0

Predicted value: 20807.0        Actual value: 20737.0

Predicted value: 32901.0        Actual value: 32632.0
```

Full decision tree:

Trained Tree: {'feature': 'X8', 'value': np.float64(114.598), 'left': {'feature': 'X8', 'value': np.float64(89.50999999999999), 'left': {'feature': 'X8', 'value': np.float64(75.51599999999999), 'left': {'feature': 'X8', 'value': np.float64(69.068), 'left': {'feature': 'X8',

'value': np.float64(63.966), 'left': {'feature': 'X6', 'value':
np.float64(26.86278697457706), 'left': {'feature': 'X3', 'value':
np.float64(46.82973360305262), 'left': {'feature': 'X2', 'value':
np.float64(14.0), 'left': np.float64(5657.0), 'right': np.float64(5435.0)},
'right': {'feature': 'X2', 'value': np.float64(14.0), 'left':
np.float64(4233.0), 'right': np.float64(4924.0)}}, 'right': {'feature':
'X8', 'value': np.float64(62.886), 'left': {'feature': 'X8', 'value':
np.float64(62.588), 'left': {'feature': 'X3', 'value':
np.float64(44.22517225668114), 'left': np.float64(6193.0), 'right':
{'feature': 'X1', 'value': np.float64(1.0), 'left': np.float64(6294.0),
'right': np.float64(6254.0)}}, 'right': {'feature': 'X3', 'value':
np.float64(20.0), 'left': np.float64(6443.0), 'right': {'feature': 'X3',
'value': np.float64(34.14044538702768), 'left': np.float64(6383.0), 'right':
{'feature': 'X1', 'value': np.float64(6.0), 'left': np.float64(6347.0),
'right': np.float64(6356.0)}}}}, 'right': {'feature': 'X2', 'value':
np.float64(11.0), 'left': np.float64(6983.0), 'right':
np.float64(6812.0)}}}, 'right': {'feature': 'X8', 'value':
np.float64(66.59), 'left': {'feature': 'X8', 'value': np.float64(65.832),
'left': {'feature': 'X8', 'value': np.float64(65.34), 'left': {'feature':
'X3', 'value': np.float64(41.29539622226181), 'left': np.float64(7670.0),
'right': {'feature': 'X1', 'value': np.float64(1.0), 'left':
np.float64(7524.0), 'right': np.float64(7525.0)}}, 'right': {'feature':
'X8', 'value': np.float64(65.592), 'left': {'feature': 'X2', 'value':
np.float64(2.0), 'left': np.float64(7796.0), 'right': {'feature': 'X1',
'value': np.float64(1.0), 'left': np.float64(7779.0), 'right':
np.float64(7772.0)}}, 'right': {'feature': 'X1', 'value': np.float64(1.0),
'left': np.float64(7916.0), 'right': np.float64(7862.0)}}}, 'right':
{'feature': 'X8', 'value': np.float64(66.094), 'left': {'feature': 'X5',
'value': np.float64(24.61318768406163), 'left': {'feature': 'X1', 'value':
np.float64(1.0), 'left': np.float64(8047.0), 'right': np.float64(8021.0)},
'right': np.float64(7976.0)}, 'right': {'feature': 'X3', 'value':
np.float64(43.06546567657216), 'left': {'feature': 'X1', 'value':
np.float64(1.0), 'left': np.float64(8285.0), 'right': np.float64(8295.0)},
'right': {'feature': 'X1', 'value': np.float64(1.0), 'left': {'feature':
'X3', 'value': np.float64(46.94455192136265), 'left': np.float64(8157.0),
'right': np.float64(8159.0)}, 'right': {'feature': 'X1', 'value':

np.float64(5.0), 'left': np.float64(8193.0), 'right': np.float64(8210.0)}}}}}, 'right': {'feature': 'X8', 'value': np.float64(67.824), 'left': {'feature': 'X8', 'value': np.float64(67.314), 'left': {'feature': 'X5', 'value': np.float64(28.22965523470745), 'left': np.float64(8423.0), 'right': {'feature': 'X8', 'value': np.float64(67.1), 'left': {'feature': 'X1', 'value': np.float64(5.0), 'left': np.float64(8541.0), 'right': np.float64(8550.0)}, 'right': {'feature': 'X1', 'value': np.float64(1.0), 'left': np.float64(8657.0), 'right': np.float64(8615.75)}}}, 'right': {'feature': 'X6', 'value': np.float64(44.46501618523303), 'left': np.float64(8912.0), 'right': {'feature': 'X3', 'value': np.float64(31.645347065518813), 'left': np.float64(8718.0), 'right': {'feature': 'X1', 'value': np.float64(0.0), 'left': np.float64(8796.0), 'right': np.float64(8799.0)}}}}, 'right': {'feature': 'X1', 'value': np.float64(4.0), 'left': {'feature': 'X4', 'value': np.float64(1.3197194568860413), 'left': {'feature': 'X8', 'value': np.float64(68.47), 'left': {'feature': 'X3', 'value': np.float64(20.0), 'left': np.float64(9235.0), 'right': np.float64(9219.0)}, 'right': np.float64(9254.0)}, 'right': np.float64(9054.0)}, 'right': np.float64(9534.0)}}}}, 'right': {'feature': 'X8', 'value': np.float64(72.69800000000001), 'left': {'feature': 'X8', 'value': np.float64(71.126), 'left': {'feature': 'X4', 'value': np.float64(1.3816488295693146), 'left': {'feature': 'X4', 'value': np.float64(1.340565811337875), 'left': {'feature': 'X3', 'value': np.float64(36.9910629412649), 'left': {'feature': 'X1', 'value': np.float64(5.0), 'left': np.float64(9992.0), 'right': np.float64(10001.0)}, 'right': np.float64(10097.0)}, 'right': np.float64(10241.0)}, 'right': np.float64(10563.0)}, 'right': {'feature': 'X8', 'value': np.float64(72.03), 'left': {'feature': 'X1', 'value': np.float64(3.0), 'left': {'feature': 'X1', 'value': np.float64(0.0), 'left': {'feature': 'X3', 'value': np.float64(43.6982128267945), 'left': np.float64(10852.0), 'right': np.float64(10833.0)}, 'right': np.float64(10736.0)}, 'right': {'feature': 'X1', 'value': np.float64(5.0), 'left': {'feature': 'X1', 'value': np.float64(4.0), 'left': np.float64(10952.0), 'right': np.float64(10914.0)}, 'right': np.float64(11015.0)}}, 'right': {'feature': 'X5', 'value': np.float64(38.26000095380565), 'left': {'feature': 'X1', 'value': np.float64(0.0), 'left': np.float64(11193.0), 'right': np.float64(11172.0)},

```
# 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(

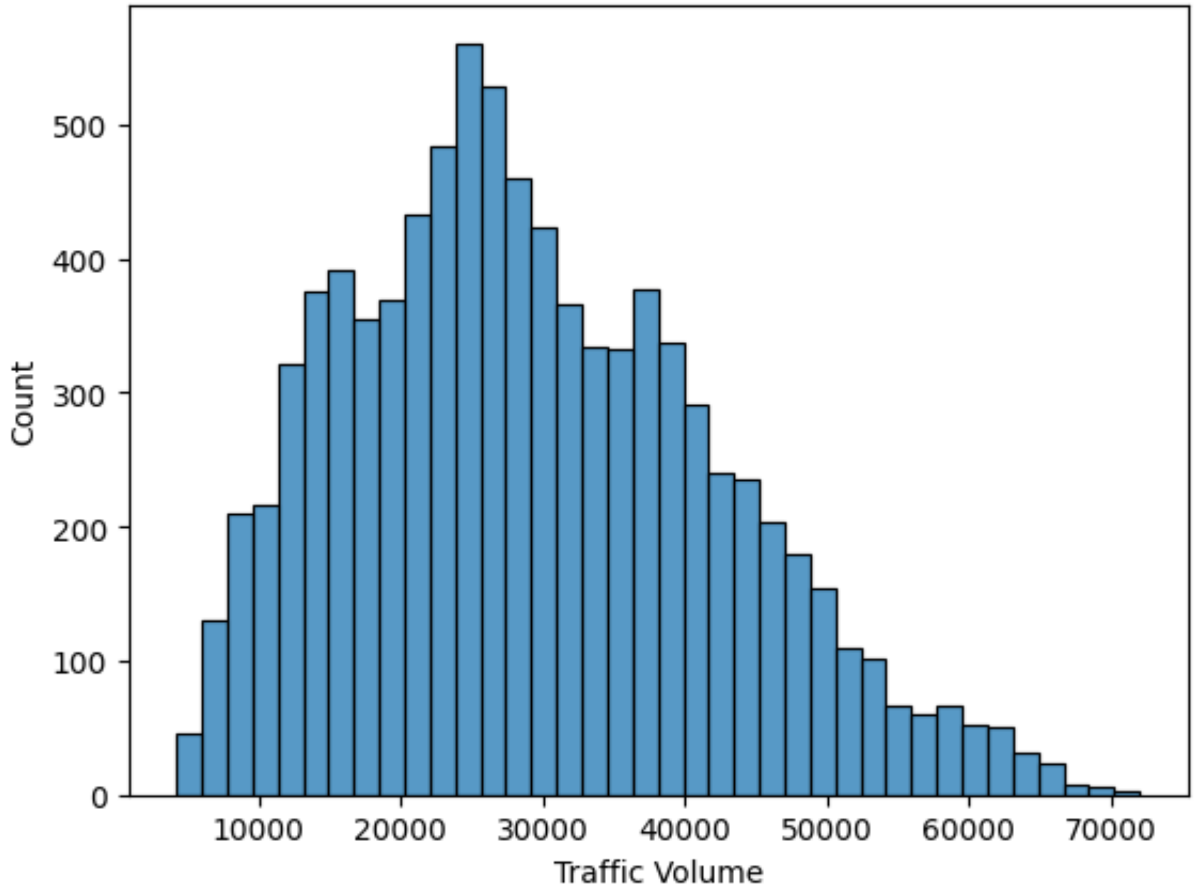
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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
```

```
# 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	100.00000
1	2022-01-01	Indiranagar	CMH Road	30825	29.377125	1.500000	100.00000
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	100.00000
4	2022-01-01	Koramangala	Sarjapur Road	57292	41.116763	1.500000	100.00000

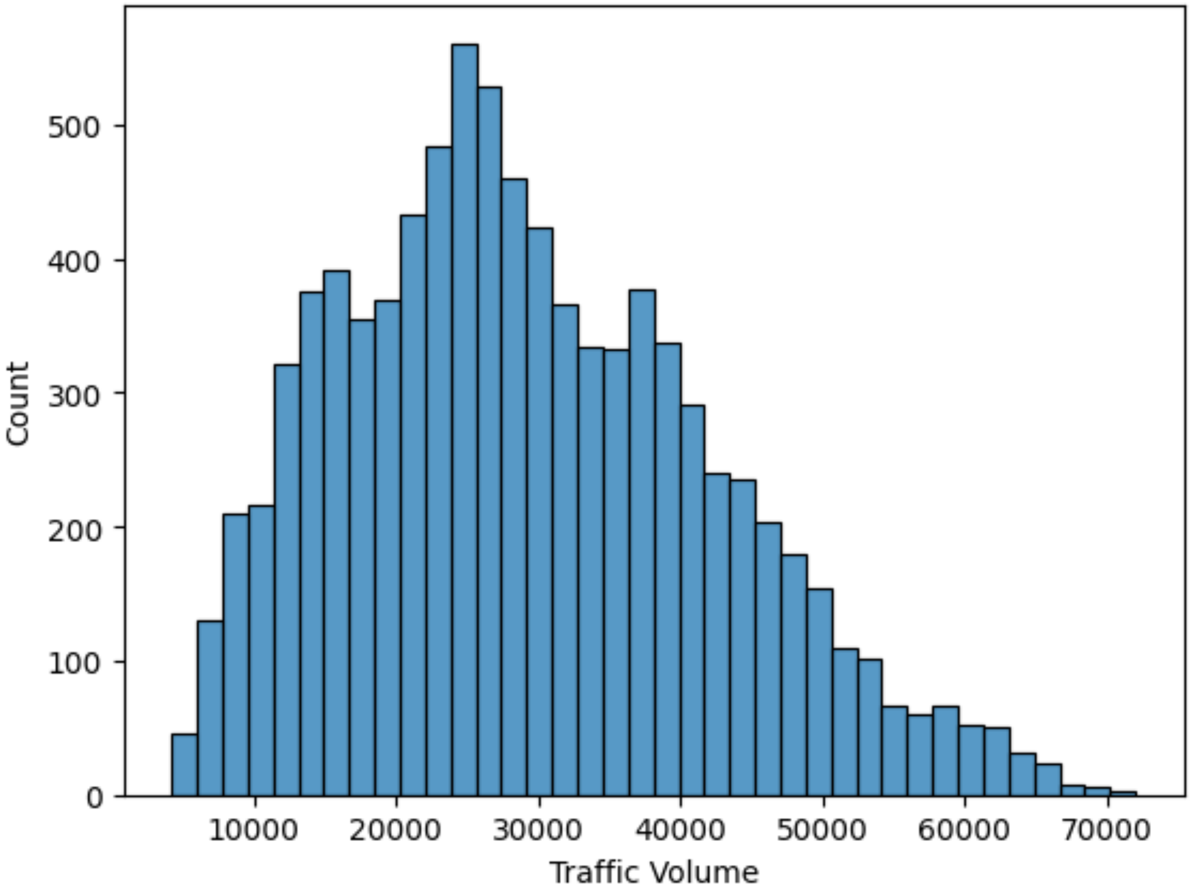
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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 = 0.2
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:  1787
Testing set samples:  7149
```

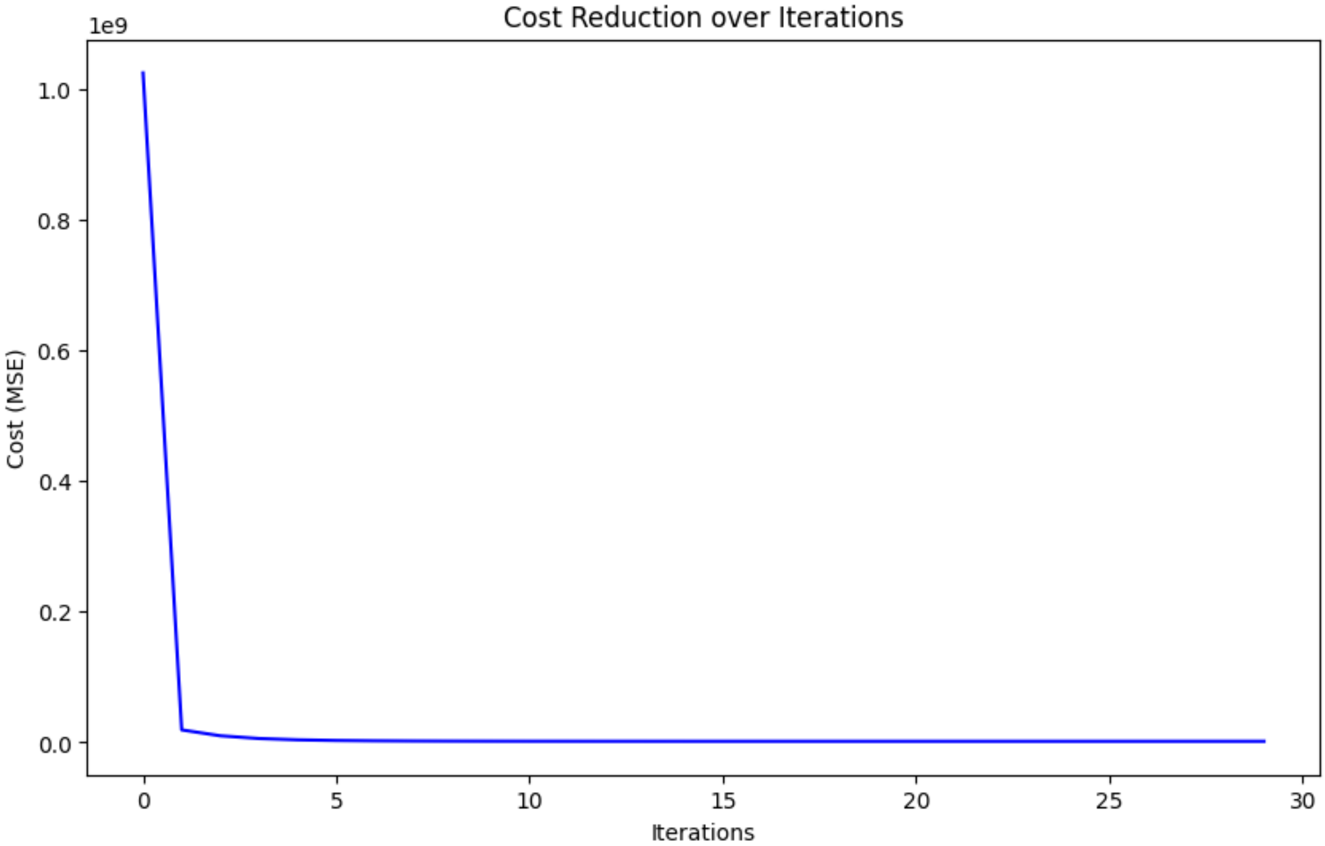
```
# for computing gradient descent, we use the the firmula new_weights = old_weig
# dJ/dW = -2X^T Y + 2X^T XW = 2X^T(XW-Y)

def gradient_descent(X, Y, learning_rate=0.01, iterations=1000):
    n_samples, n_features = X.shape

    # initialize weights to 0
    weights = np.zeros(n_features)
    # List to store cost at each iteration for plotting convergence
    cost_history = []

    for i in range(iterations):
        # Y_pred ->  XW
        Y_pred = X @ weights

        # compute the error -> XW-Y
        error = Y_pred - Y

        # compute gradient -> dJ/dW
        gradient = (2 / n_samples) * (X.T @ error)

        # Update weights
        weights -= learning_rate * gradient

        # Compute Mean Squared Error (Cost Function)
        cost = (1 / n_samples) * np.sum(error ** 2)

        # (Optional) Print cost at intervals
        if i % 50 == 0:
            print(f"Iteration {i+1}: Cost {cost}")
            cost_history.append(cost)

    return weights, cost_history
```

```
# we solve the least squares problem using the gradient descent algorithm
weights, cost_history = gradient_descent(X_train,Y_train,0.00002,1500)
heading("Training on linear regression with given dataset for 10 iterations")
print(weights[:5])
```

```
Iteration 1: Cost 1024440498.4297707
Iteration 51: Cost 19257093.994292814
Iteration 101: Cost 10232535.491425855
Iteration 151: Cost 6051920.219076406
Iteration 201: Cost 4053860.8857249906
Iteration 251: Cost 3060828.882377268
Iteration 301: Cost 2543862.3888015794
Iteration 351: Cost 2260861.998928928
Iteration 401: Cost 2098092.154895537
Iteration 451: Cost 2000241.9044550362
Iteration 501: Cost 1939238.3680511315
Iteration 551: Cost 1900124.1144997755
Iteration 601: Cost 1874519.4331545122
Iteration 651: Cost 1857503.5425434855
Iteration 701: Cost 1846068.2464918403
Iteration 751: Cost 1838315.207653634
Iteration 801: Cost 1833017.9106328539
Iteration 851: Cost 1829370.567145155
Iteration 901: Cost 1826837.6907410177
Iteration 951: Cost 1825060.655838286
Iteration 1001: Cost 1823798.0246972395
Iteration 1051: Cost 1822886.668808727
Iteration 1101: Cost 1822216.0840303628
Iteration 1151: Cost 1821711.262183427
Iteration 1201: Cost 1821321.1974394964
Iteration 1251: Cost 1821011.142274124
Iteration 1301: Cost 1820757.3758214433
Iteration 1351: Cost 1820543.6633198147
Iteration 1401: Cost 1820358.8571920535
Iteration 1451: Cost 1820195.2703071258


##### Training on linear regression with given dataset for 10 iterations ##
[ -5.4905183  -21.597431   -56.54143836 -35.13642342  -5.22527202]
```

```
# Plotting the cost history using seaborn
print(len(cost_history))
plt.figure(figsize=(10,6))
sns.lineplot(x=range(len(cost_history)), y=cost_history, color='blue')
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.title('Cost Reduction over Iterations')
plt.show()
```

```
30
```



```
# 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: 5.557551943610017 %

Predicted value: 34426.65982240805      Actual value: 35269.0

Predicted value: 27926.53382259589      Actual value: 28038.0

Predicted value: 27379.23702980667      Actual value: 27049.0

Predicted value: 23692.006650849784     Actual value: 24265.0

Predicted value: 40315.83882924316      Actual value: 41557.0

Predicted value: 10393.53185509519      Actual value: 8059.0

Predicted value: 9942.332775055342      Actual value: 8077.0

Predicted value: 11039.920832360887     Actual value: 14725.0

Predicted value: 11215.776744104967     Actual value: 12832.0

Predicted value: 35995.83136993581      Actual value: 36190.0
```

```
# 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(

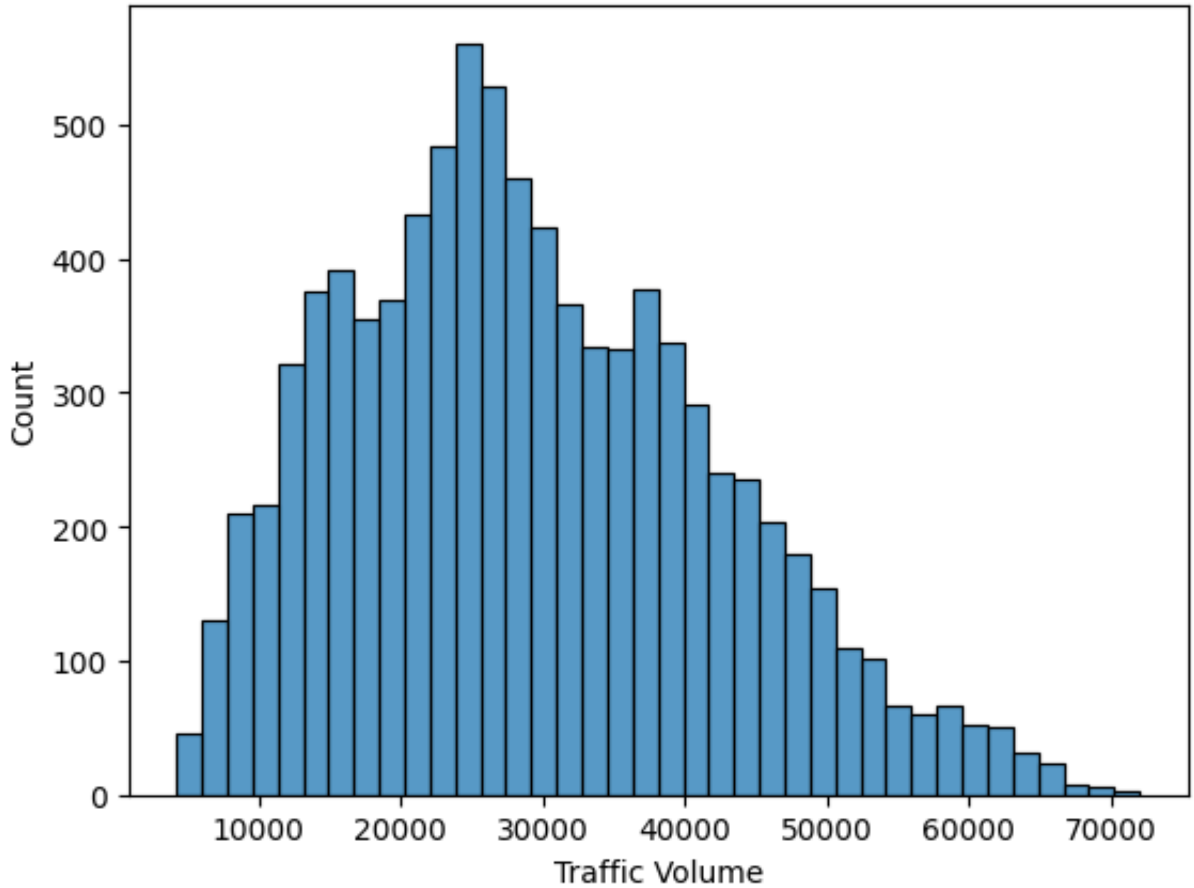
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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 #####
&lt;Axes: xlabel='Traffic Volume', ylabel='Count'&gt;
```



```
# 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])
```

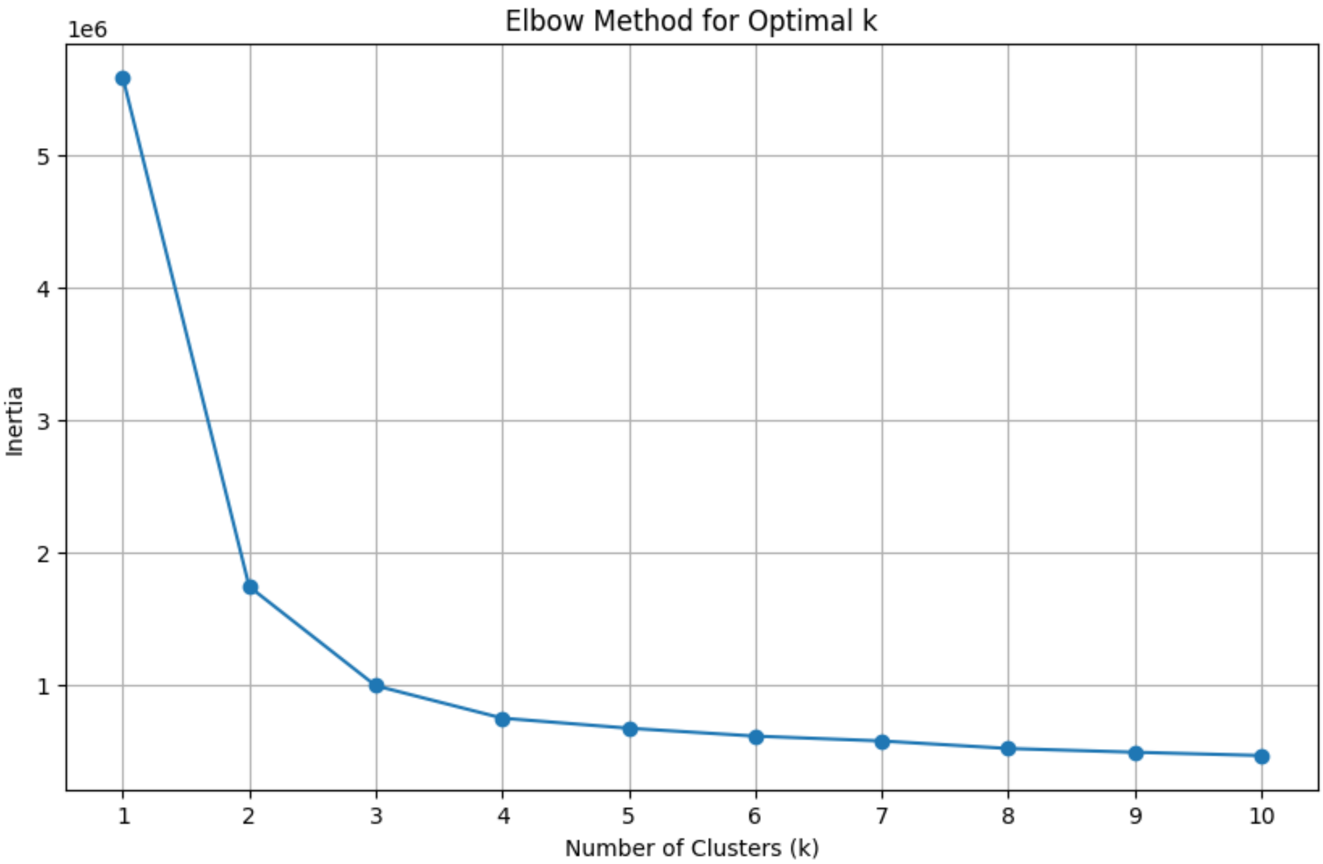
```
##### Printing X and Y variables for the model #####
[[  1.           0.           0.          50.23029856   1.5
  100.         100.           0.         151.18       111.        ]
 [  1.           0.           1.          29.37712471   1.5
  100.         100.           1.         111.65       100.        ]
 [  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 = 0.2
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:  1787
Testing set samples:  7149
```

```
def kmeans(X, k, n_iters=100):
    # Randomly initialize centroids
    np.random.seed(0)
    random_indices = np.random.choice(X.shape[0], k, replace=False)
    centroids = X[random_indices]

    for _ in range(n_iters):
        # Assign clusters
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)  # Cal
        labels = np.argmin(distances, axis=1)  # Assign clusters

        # Update centroids
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(k)

        # Check for convergence
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids

    # Calculate inertia
    inertia = np.sum((X - centroids[labels])**2)
    return labels, centroids, inertia
```

```
# Calculate inertia for different k values
inertia = []
k_vals = range(1, 11)

# Testing k from 1 to 10
for k in range(1, 11):
    _, _, inertia_value = kmeans(X_train, k)
    inertia.append(inertia_value)

# Plot elbow graph
plt.figure(figsize=(10, 6))
plt.plot(k_vals, inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_vals)
plt.grid()
plt.show()
```



```
def addClusters(X,k=3):
    labels, _, _ = kmeans(X,k)
    labels_v = labels[:,np.newaxis]
    return np.hstack((X,labels_v))


# Choosing 3 clusters

k_count = 3
X_train_clustered = addClusters(X_train,k_count)
print(X_train[:10])
```

```
[[  1.           0.           1.          37.37103217   1.5
  100.         100.           5.         138.108       96.        ]
 [  1.           5.          10.          29.35105081   1.22083068
   26.56832984  50.91059572   0.          68.428      167.        ]
 [  1.           3.           5.          22.71256237   1.5
  100.         100.           2.         131.066       96.        ]
 [  1.           1.          13.          56.7650234    1.37401091
   75.60291193 100.           3.          92.832      100.        ]
 [  1.           1.           2.          59.52684493   1.12717384
   66.01944166 100.           0.          92.748       93.        ]
 [  1.           0.           1.          25.92739549   1.5
   99.69170703 100.           1.         104.68       106.        ]
 [  1.           3.           5.          47.65180139   1.3423462
   89.44946886 100.           2.          98.53       108.        ]
 [  1.           2.           4.          24.43964816   1.5
  100.         100.           5.         170.822      104.        ]
 [  1.           7.          15.          30.51524769   1.5
  100.         100.           2.         111.576      120.        ]
 [  1.           6.          11.          47.80546602   1.06341905
   76.70233696 100.           3.          95.082      104.        ]]
```

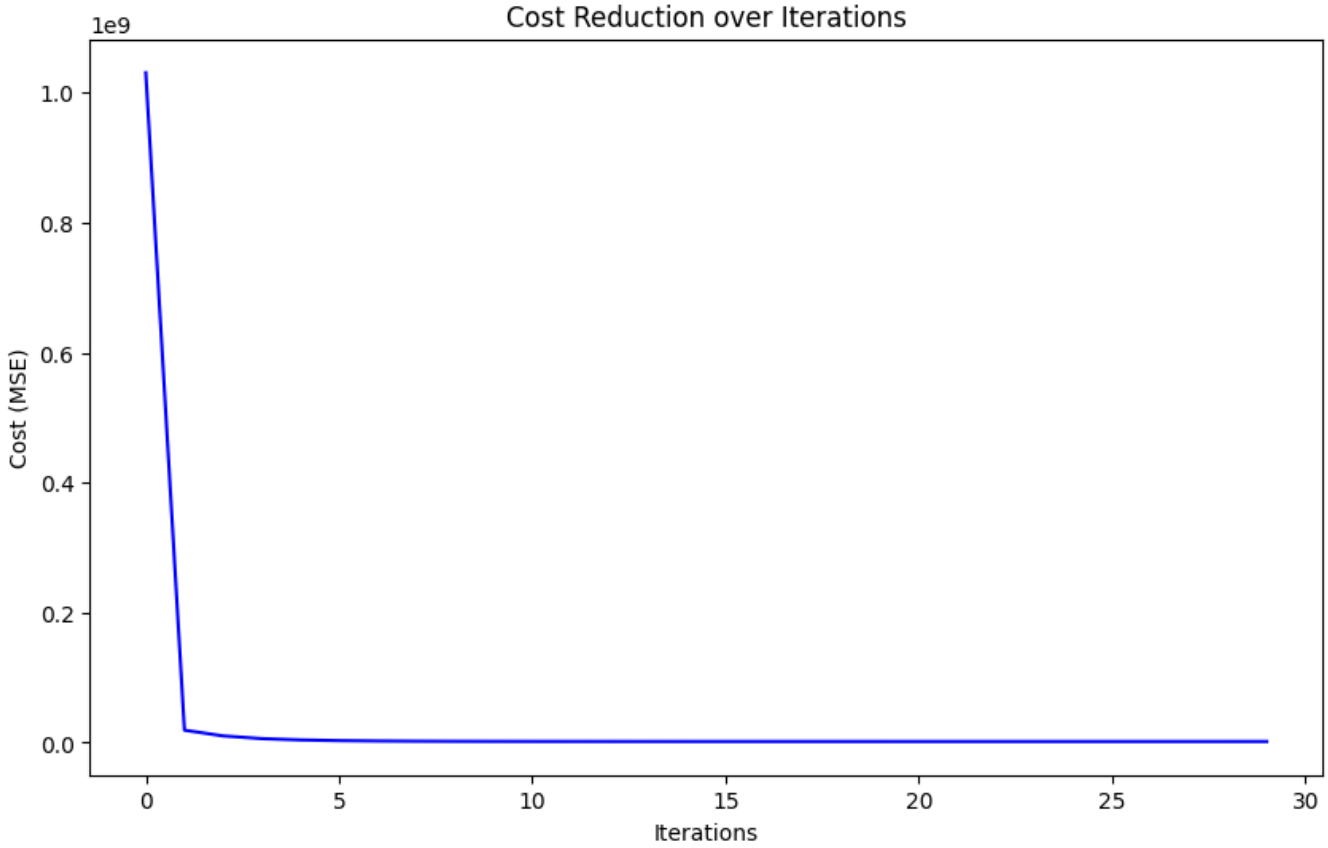
```
# for computing gradient descent, we use the the firmula new_weights = old_weig
# dJ/dW = -2X^T Y + 2X^T XW = 2X^T(XW-Y)

def gradient_descent(X, Y, learning_rate=0.01, iterations=1000):
    n_samples, n_features = X.shape

    # initialize weights to 0
    weights = np.zeros(n_features)
    # List to store cost at each iteration for plotting convergence
    cost_history = []

    for i in range(iterations):
        # Y_pred ->  XW
        Y_pred = X @ weights

        # compute the error -> XW-Y
        error = Y_pred - Y

        # compute gradient -> dJ/dW
        gradient = (2 / n_samples) * (X.T @ error)

        # Update weights
        weights -= learning_rate * gradient

        # Compute Mean Squared Error (Cost Function)
        cost = (1 / n_samples) * np.sum(error ** 2)

        # (Optional) Print cost at intervals
        if i % 50 == 0:
            print(f"Iteration {i+1}: Cost {cost}")
            cost_history.append(cost)

    return weights, cost_history
```

```
# we solve the least squares problem using the gradient descent algorithm
weights, cost_history = gradient_descent(X_train_clustered,Y_train,0.00002,150(
heading("Training on linear regression with given dataset for 10 iterations")
print(weights[:5])
```

```
Iteration 1: Cost 1030863311.8203694
Iteration 51: Cost 19209790.29673657
Iteration 101: Cost 10397071.409513244
Iteration 151: Cost 6208426.660506265
Iteration 201: Cost 4160200.239947096
Iteration 251: Cost 3123234.7223804705
Iteration 301: Cost 2576722.522288622
Iteration 351: Cost 2276031.3479443653
Iteration 401: Cost 2103414.146862621
Iteration 451: Cost 2000404.7585536845
Iteration 501: Cost 1936868.659267455
Iteration 551: Cost 1896614.8750633015
Iteration 601: Cost 1870564.926955068
Iteration 651: Cost 1853419.0142903784
Iteration 701: Cost 1841972.4034256407
Iteration 751: Cost 1834230.8211609973
Iteration 801: Cost 1828925.4912570387
Iteration 851: Cost 1825235.8847671105
Iteration 901: Cost 1822625.0647027805
Iteration 951: Cost 1820738.707754144
Iteration 1001: Cost 1819341.583840606
Iteration 1051: Cost 1818276.8278582876
Iteration 1101: Cost 1817439.4618013941
Iteration 1151: Cost 1816759.0127123976
Iteration 1201: Cost 1816188.0246262492
Iteration 1251: Cost 1815694.430766824
Iteration 1301: Cost 1815256.4727904287
Iteration 1351: Cost 1814859.3091331674
Iteration 1401: Cost 1814492.747360092
Iteration 1451: Cost 1814149.7262153195


##### Training on linear regression with given dataset for 10 iterations ##
[ -5.50793759 -28.35978491 -62.70057154 -35.48559764  -5.17234129]
```

```
# Plotting the cost history using seaborn
print(len(cost_history))
plt.figure(figsize=(10,6))
sns.lineplot(x=range(len(cost_history)), y=cost_history, color='blue')
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.title('Cost Reduction over Iterations')
plt.show()
```

```
30
```



```
# Predicting values with K means clustering over test dataset
X_test_clustered = addClusters(X_test,k_count)

Y_pred = X_test_clustered @ 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: 5.554205761949023 %

Predicted value: 29692.36974725272      Actual value: 29758.0

Predicted value: 10951.108997793792     Actual value: 11680.0

Predicted value: 20796.347250916715     Actual value: 20350.0

Predicted value: 11571.909192580497     Actual value: 11840.0

Predicted value: 34793.29390196996      Actual value: 33394.0

Predicted value: 38864.19583998721      Actual value: 40198.0

Predicted value: 33412.03555788865      Actual value: 34455.0

Predicted value: 22039.508470081484     Actual value: 22823.0

Predicted value: 20959.281868416234     Actual value: 21337.0

Predicted value: 37827.473477525935     Actual value: 37045.0
```

```
# 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(

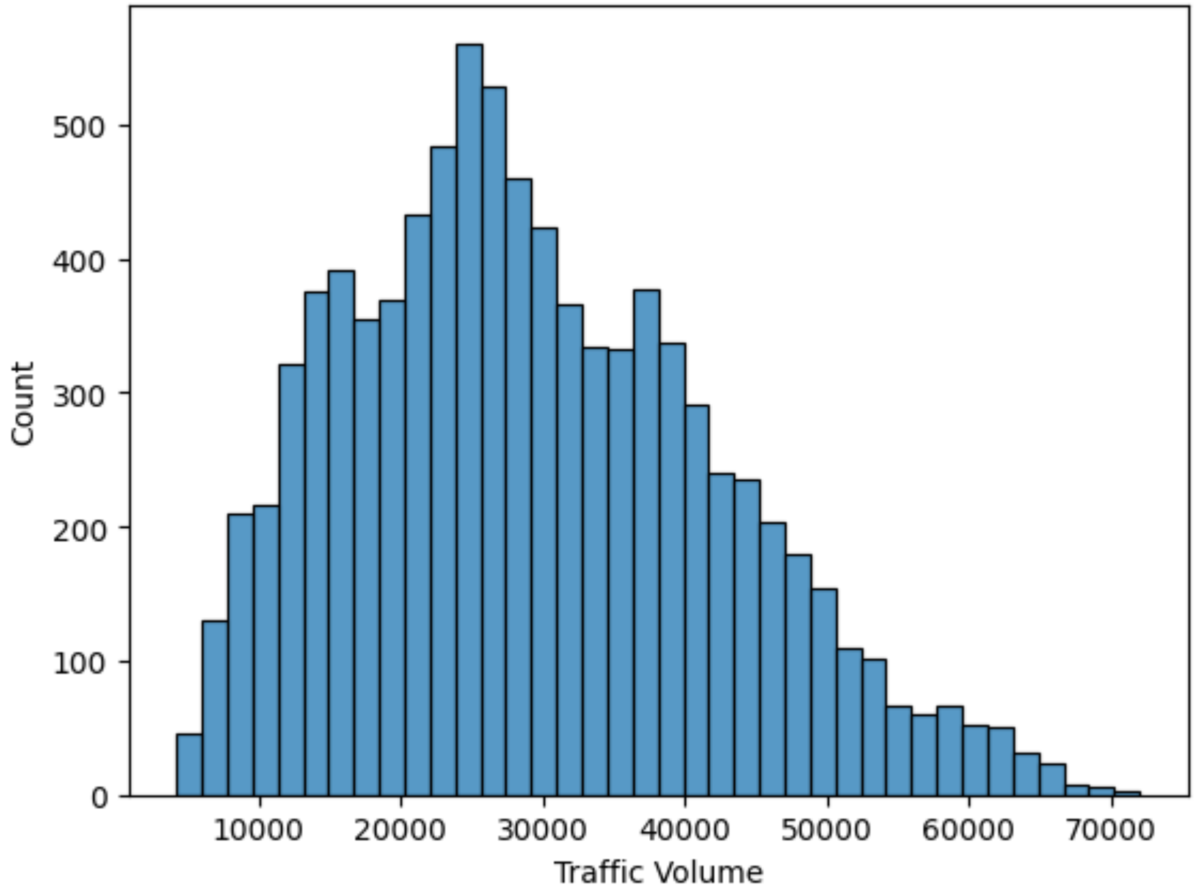
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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]
```

```
# we normalize our dataset

# we use standadization for input variables, since they are of different scales
# standardization helps set the mean to 0 and std dev to 1.
intercept_column = X_shuffled[:, 0]
features = X_shuffled[:, 1:]

X_mean = np.mean(features, axis=0)
X_std = np.std(features, axis=0)

# Normalize features
X_norm_temp = (features - X_mean) / X_std
X_norm = np.column_stack((intercept_column, X_norm_temp))

# we use min max normaliztion for the output variable, since it is in fixed rar
# also, standardization may yeild negative values which may not be as intuitive
Y_mean = np.mean(Y, axis=0)
Y_std = np.std(Y, axis=0)

# Normalize features
Y_norm = (Y - Y_mean) / Y_std
```

```
# split the dataset into 80:20
split_ratio = 0.8
split_index = int(len(X_shuffled) * split_ratio)

X_train = X_norm[:split_index]
Y_train = Y_norm[:split_index]

X_test = X_norm[split_index:]
Y_test = Y_norm[split_index:]

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

print("Training set samples: ", X_train[:5])
print("Testing set samples: ", Y_train[:5])
```

```
Training set samples size:  7148
Testing set samples size:  1788
Training set samples:  [[ 1.          0.04836864 -0.26557885 -0.24502419  0
   0.48067702 -1.10593338  0.98437596 -0.39481515]
 [ 1.          0.51563584  0.42430854  0.61332466 -0.82137174 -2.20155768
  -3.06246808 -1.10593338 -1.58425655  2.21313545]
 [ 1.          0.04836864 -0.26557885  0.38672395 -2.10567337  0.17118203
   0.48067702 -1.10593338 -0.3993551   0.23000635]
 [ 1.         -1.35343297 -1.41539116 -1.81638862  0.75280715  0.81514828
   0.48067702  0.30254958  0.48295613 -0.17748593]
 [ 1.          0.04836864 -0.26557885  1.95049546  0.13711301 -1.99925888
  -1.5518211  -1.10593338 -1.28358925  1.85997547]]
Testing set samples:  [ 1.64247507  0.1222169  -1.67963323  2.4334862   2.1
```

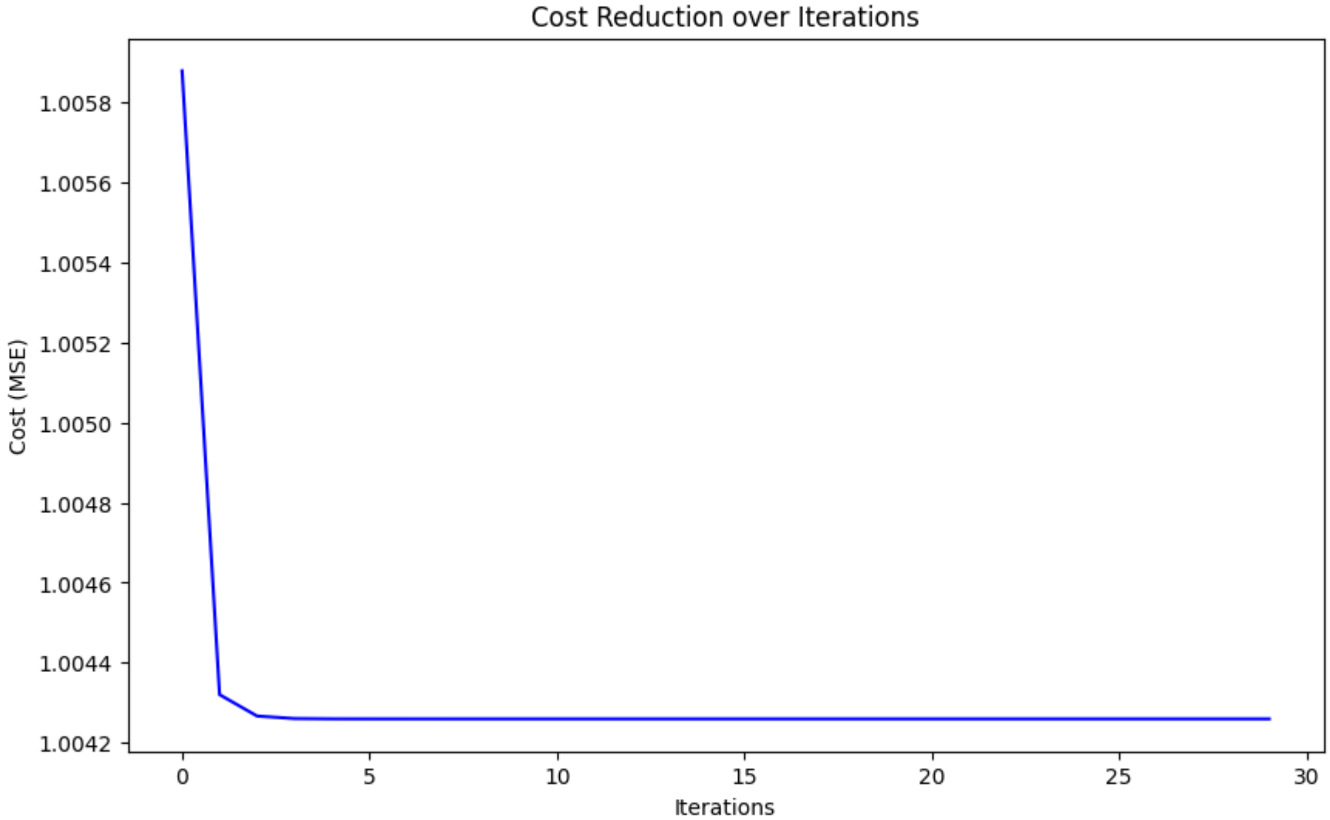
```
# for computing gradient descent, we use the the firmula new_weights = old_weig
# dJ/dW = -2X^T Y + 2X^T XW = 2X^T(XW-Y)

def gradient_descent(X, Y, learning_rate=0.01, iterations=1000):
    n_samples, n_features = X.shape

    # initialize weights to 0
    weights = np.zeros(n_features)
    # List to store cost at each iteration for plotting convergence
    cost_history = []

    for i in range(iterations):
        # Y_pred ->  XW
        Y_pred = X @ weights

        # compute the error -> XW-Y
        error = Y_pred - Y

        # compute gradient -> dJ/dW
        gradient = (2 / n_samples) * (X.T @ error)

        # Update weights
        weights -= learning_rate * gradient

        # Compute Mean Squared Error (Cost Function)
        cost = (1 / n_samples) * np.sum(error ** 2)

        # (Optional) Print cost at intervals
        if i % 100 == 0:
            print(f"Iteration {i+1}: Cost {cost}")
            cost_history.append(cost)

    return weights, cost_history
```

```
# we solve the least squares problem using the gradient descent algorithm
weights, cost_history = gradient_descent(X_train,Y_train,0.1,3000)
heading("Training on linear regression with given dataset for 10 iterations")
print(weights[:5])
```

```
Iteration 1: Cost 1.0058789972297144
Iteration 101: Cost 1.0043202614022586
Iteration 201: Cost 1.0042667705344606
Iteration 301: Cost 1.0042605008369565
Iteration 401: Cost 1.0042597657038155
Iteration 501: Cost 1.0042596795076422
Iteration 601: Cost 1.004259669400927
Iteration 701: Cost 1.0042596682158895
Iteration 801: Cost 1.0042596680769411
Iteration 901: Cost 1.0042596680606488
Iteration 1001: Cost 1.0042596680587388
Iteration 1101: Cost 1.0042596680585145
Iteration 1201: Cost 1.0042596680584883
Iteration 1301: Cost 1.0042596680584854
Iteration 1401: Cost 1.0042596680584848
Iteration 1501: Cost 1.004259668058485
Iteration 1601: Cost 1.0042596680584848
Iteration 1701: Cost 1.0042596680584848
Iteration 1801: Cost 1.0042596680584848
Iteration 1901: Cost 1.0042596680584848
Iteration 2001: Cost 1.0042596680584848
Iteration 2101: Cost 1.0042596680584848
Iteration 2201: Cost 1.004259668058485
Iteration 2301: Cost 1.0042596680584848
Iteration 2401: Cost 1.0042596680584848
Iteration 2501: Cost 1.0042596680584848
Iteration 2601: Cost 1.0042596680584848
Iteration 2701: Cost 1.0042596680584848
Iteration 2801: Cost 1.0042596680584848
Iteration 2901: Cost 1.0042596680584848


##### Training on linear regression with given dataset for 10 iterations ##
[-0.00058856 -0.01247141  0.00304069 -0.00042342  0.00373874]
```

```
# Plotting the cost history using seaborn
print(len(cost_history))
plt.figure(figsize=(10,6))
sns.lineplot(x=range(len(cost_history)), y=cost_history, color='blue')
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.title('Cost Reduction over Iterations')
plt.show()
```

```
30
```



```
# lets predict values
Y_pred = X_test @ weights
# we denormalize the predictions and actual testing data
Y_pred_original = (Y_pred * Y_std) + Y_mean
Y_test_denorm = (Y_test * Y_std) + Y_mean

# we will use mean absolute percentage error to calculate the error percentage
MAPE = np.mean(np.abs((Y_test_denorm - Y_pred_original) / Y_test_denorm)) * 10(

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_original|
```

```
##### Printing the MAPE and first 10 predictions with actual values #####
MAPE: 51.3920872695371 %

Predicted value: 29112.69893525145      Actual value: 51665.0

Predicted value: 28845.33853487075      Actual value: 36384.0

Predicted value: 29363.322605982026     Actual value: 41692.0

Predicted value: 29185.897419801313     Actual value: 29831.0

Predicted value: 29258.280767655135     Actual value: 10915.0

Predicted value: 29134.03138423169      Actual value: 25909.0

Predicted value: 29060.315434471842     Actual value: 37720.0

Predicted value: 29040.760707823174     Actual value: 26712.0

Predicted value: 28838.098747263044     Actual value: 33467.0

Predicted value: 29915.88538897134      Actual value: 27426.0
```

```
# 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(

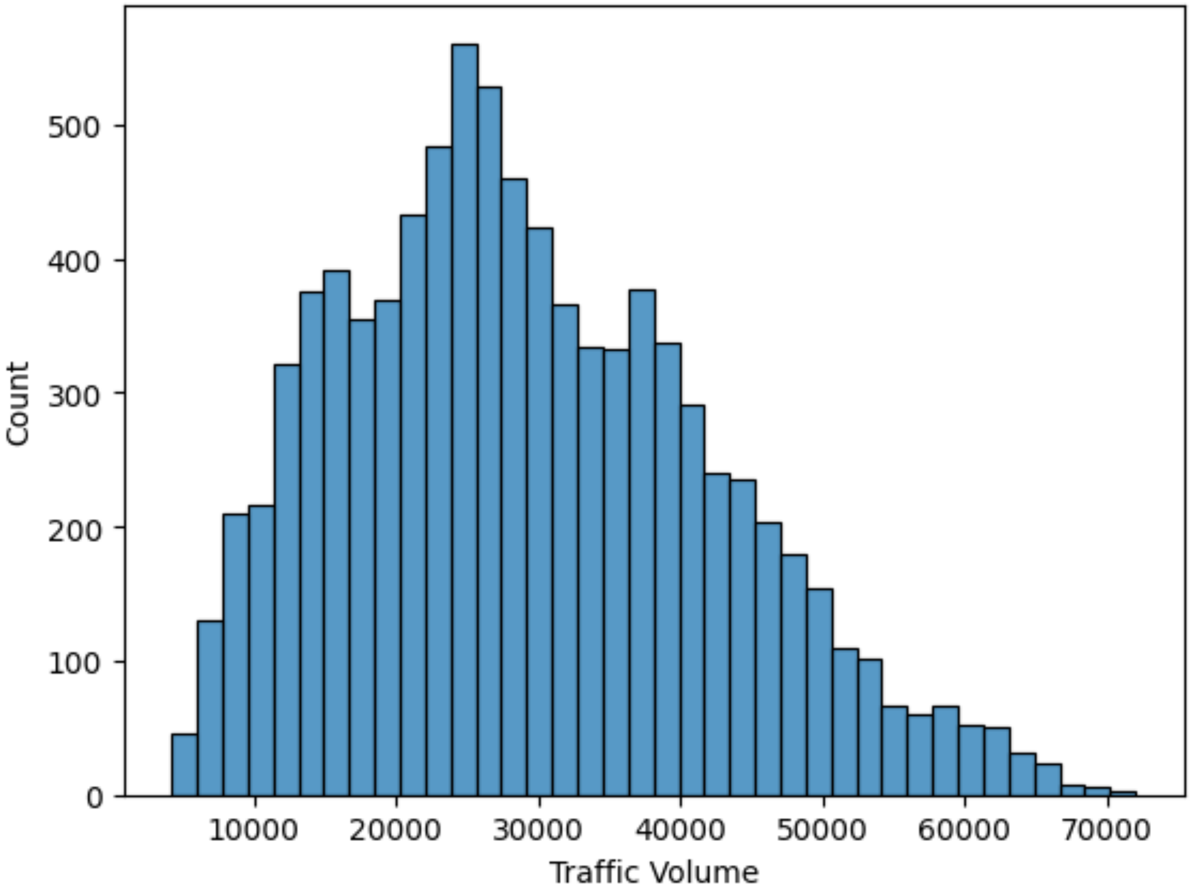
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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
```