```
# 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 | 100.00000 |
| **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 | 28.34799 |
| **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 |

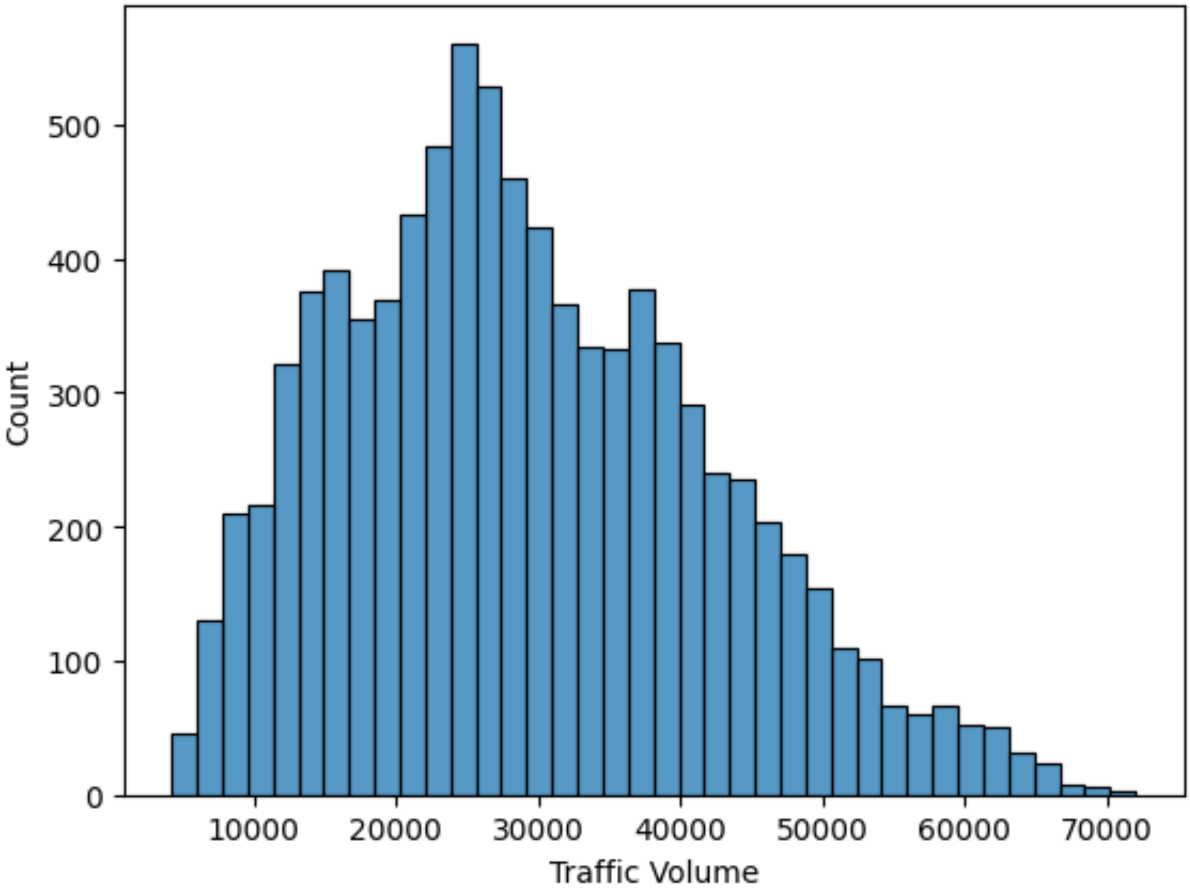
```
# lets find the individual column statistics
heading("Stats about non-numeric values")
print(dataSet.describe(include = "object"))

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])
```

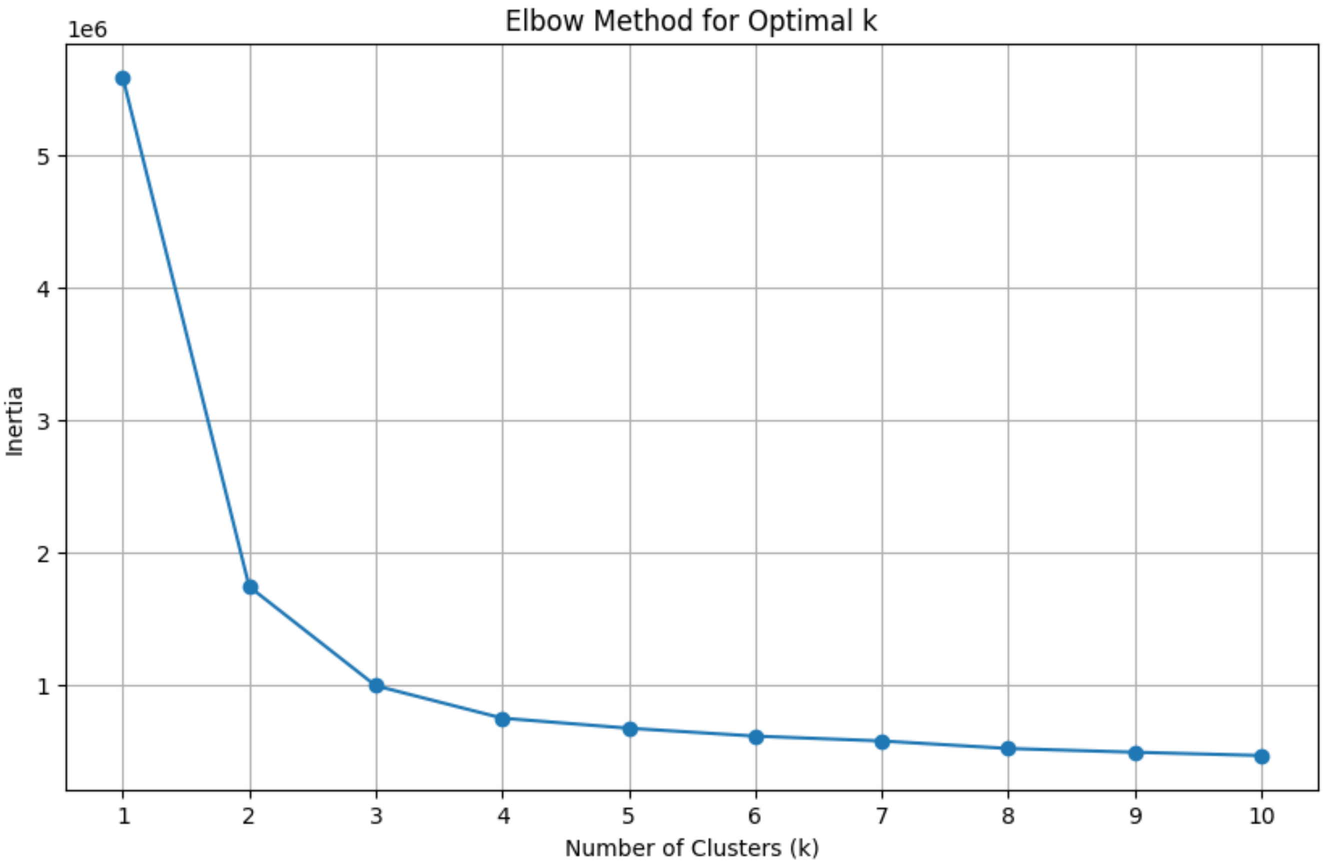
```
##### Printing X and Y variables for the model #####
[[  1.           0.           0.          50.23029856   1.5
  100.         100.           0.         151.18       111.        ]
 [  1.           0.           1.          29.37712471   1.5
  100.         100.           1.         111.65       100.        ]
 [  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
```

```
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.        ]]
```

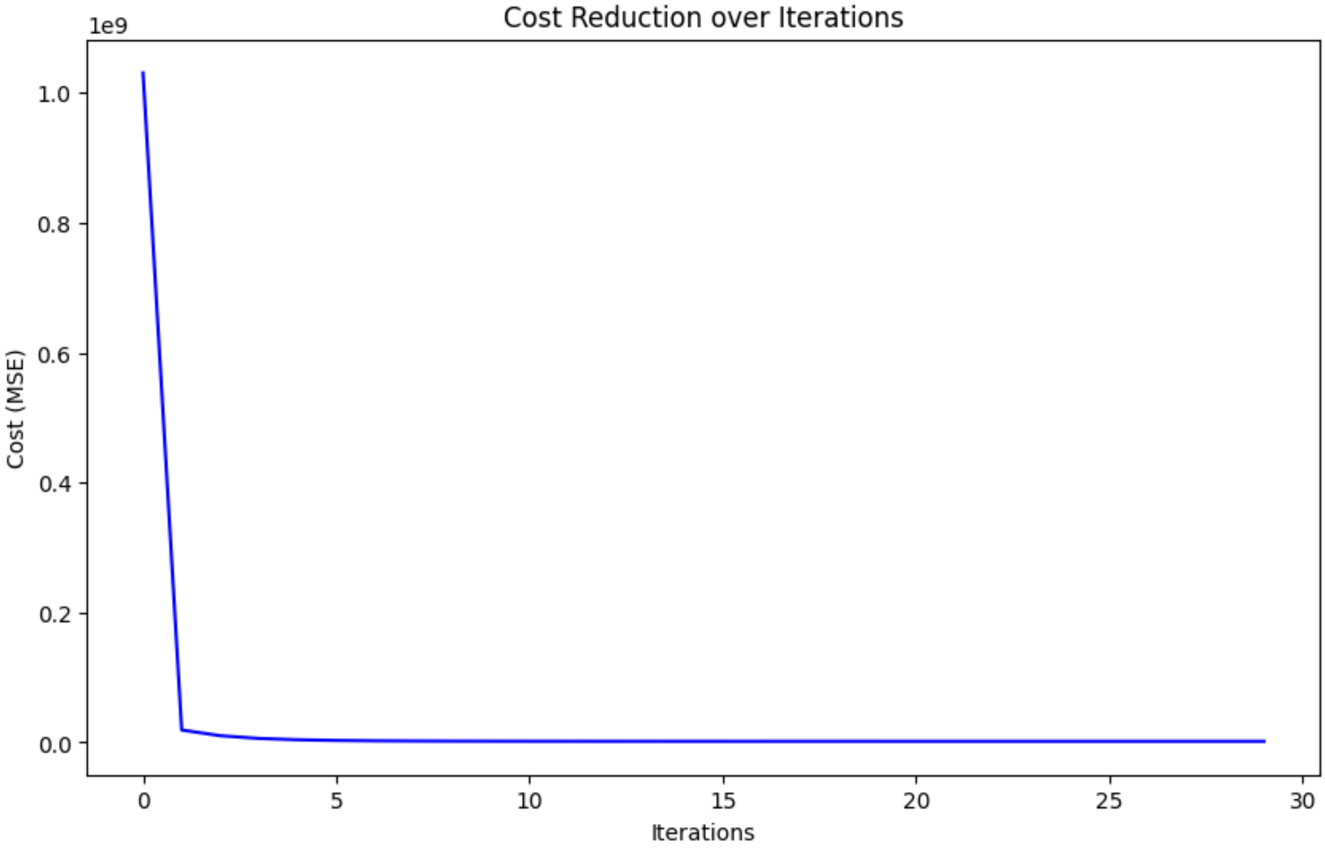
```
# for computing gradient descent, we use the the firmula new_weights = old_weig
# dJ/dW = -2X^T Y + 2X^T XW = 2X^T(XW-Y)

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
```