```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


df = pd.read_csv("./CO2_Emissions_Canada.csv")
```

Pre-Proceccsing

```
df.head()
```

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	C
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	

```
df.describe()
```

```
df.corr()
```

	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)
Engine Size(L)	1.000000	0.927653	0.831379	0.761526	0.817060	-0.757
Cylinders	0.927653	1.000000	0.800702	0.715252	0.780534	-0.719
Fuel Consumption City (L/100 km)	0.831379	0.800702	1.000000	0.948180	0.993810	-0.927
Fuel Consumption Hwy (L/100 km)	0.761526	0.715252	0.948180	1.000000	0.977299	-0.890
Fuel Consumption Comb (L/100 km)	0.817060	0.780534	0.993810	0.977299	1.000000	-0.925
Fuel Consumption Comb (mpg)	-0.757854	-0.719321	-0.927059	-0.890638	-0.925576	1.000

Checking and Deleting duplicate rows:

```
df.isna().sum()
```

```
Make                              0
Model                             0
Vehicle Class                     0
Engine Size(L)                    0
Cylinders                         0
Transmission                      0
Fuel Type                         0
Fuel Consumption City (L/100 km)  0
Fuel Consumption Hwy (L/100 km)   0
Fuel Consumption Comb (L/100 km)  0
Fuel Consumption Comb (mpg)       0
CO2 Emissions(g/km)               0
dtype: int64
```

```
print(df.duplicated().sum())
df=df.drop_duplicates()
print(df.duplicated().sum())
```

```
1103
0
```

Handling numeric and non-numeric data

```
df.dtypes
```

```
Make                              object
Model                             object
Vehicle Class                     object
Engine Size(L)                   float64
Cylinders                          int64
Transmission                      object
Fuel Type                         object
Fuel Consumption City (L/100 km)  float64
Fuel Consumption Hwy (L/100 km)   float64
Fuel Consumption Comb (L/100 km)  float64
Fuel Consumption Comb (mpg)        int64
CO2 Emissions(g/km)                int64
dtype: object
```

```
print(df["Make"].nunique())
print(df["Model"].nunique())
print(df["Vehicle Class"].nunique())
print(df["Transmission"].nunique())
print(df["Fuel Type"].nunique())
```

```
42
2053
16
27
5
```

```
df=df.drop(['Model'], axis=1)
```

```
columns_To_1HotEncode=["Make","Vehicle Class","Transmission","Fuel Type"]
df = pd.get_dummies(df, columns=columns_To_1HotEncode)
df.head()
```

	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fuel Consumption Comb (mpg)	Emissions
0	2.0	4	9.9	6.7	8.5	33	
1	2.4	4	11.2	7.7	9.6	29	
2	1.5	4	6.0	5.8	5.9	48	
3	3.5	6	12.7	9.1	11.1	25	
4	3.5	6	12.1	8.7	10.6	27	

5 rows × 97 columns

Standardizing all numeric coulumns

```
columns_to_standerdize=["Engine Size(L)","Cylinders","Fuel Consumption City (L/100 km)","F
for column in columns_to_standerdize:
    df[column] = (df[column]-df[column].mean()) / df[column].std()
```

Spliting the data for testing and training purpose

```
def split_data(X,Y):
    np.random.seed(0)
    mask = np.random.rand(X.shape[0])<=0.80
    X_train = X[mask]
    X_test =  X[~mask]
    Y_train = Y[mask]
    Y_test = Y[~mask]

    return X_train,X_test,Y_train,Y_test
```

Linear Regression

```
def computeMSE(Y,X,weights):
    Y_pred = X@ weights
    MSE = 0
    for i in range(Y_pred.shape[0]):
        MSE += (Y[i] - Y_pred[i])**2
    MSE/= Y_pred.shape[0]
    return MSE

def computeMAE(Y,X,weights):
    Y_pred = X@ weights
    MAE = 0
    for i in range(Y_pred.shape[0]):
        MAE += abs(Y[i] - Y_pred[i])
    MAE/= Y_pred.shape[0]
    return MAE
```

Multivariate

```
# Making the variables ready
X = df.iloc[:,:]
X=X.drop(["CO2 Emissions(g/km)"],axis=1)
X["Constant"]=1
```

```
Y=df.iloc[:,6]

X_train,X_test,Y_train,Y_test=split_data(X,Y)

Y_train.reset_index(drop=True,inplace=True)
Y_test.reset_index(drop=True,inplace=True)
X_test.reset_index(drop=True,inplace=True)
X_train.reset_index(drop=True,inplace=True)
```

Closed Form

```
from numpy.linalg import pinv
def closed_form(X,Y):
    weights = pinv(X)@Y
    return weights



weights=closed_form(X_train,Y_train) #learning algorithm for training model
MSE=computeMSE(Y_test,X_test,weights)
MAE=computeMAE(Y_test,X_test,weights)
print("Mean Square Error: ", MSE)
print("Mean Absolute Error: ", MAE)
```

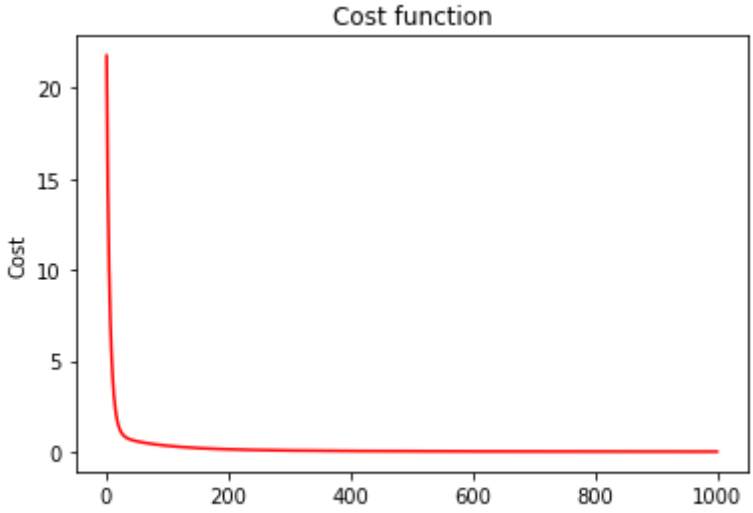
```
Mean Square Error:  0.008704893307890869
Mean Absolute Error:  0.05375495049165647
```

Gradient Descent

```
costs=[]
iterations=[]

learning_rate=.00001
itrns=1000
weights=np.ones((X_train.shape[1]))
for i in range(itrns):
    weights=weights-learning_rate*(X_train.T@(X_train@weights-Y_train))
    iterations.append(i+1)
    cost=computeMSE(Y_test,X_test,weights)
    costs.append(cost)
plt.plot(np.arange(1,itrns),costs[1:], color = 'red')
plt.title('Cost function')
plt.xlabel('Number of iterations')
plt.ylabel('Cost')
MSE=computeMSE(Y_test,X_test,weights)
MAE=computeMAE(Y_test,X_test,weights)
print("Mean Square Error: ", MSE)
print("Mean Absolute Error: ", MAE)
```

```
Mean Square Error:  0.017250752725954284
Mean Absolute Error:  0.09414584710911104
```



Univariate

```
# Making the variables ready
# X = df["Fuel Consumption City (L/100 km)"]
X = np.ones((len(df), 2))
X[:,1] = df.iloc[:,2]
X=pd.DataFrame(data=X)

Y=df.iloc[:,6]

X_train,X_test,Y_train,Y_test=split_data(X,Y)

Y_train.reset_index(drop=True,inplace=True)
Y_test.reset_index(drop=True,inplace=True)
X_test.reset_index(drop=True,inplace=True)
X_train.reset_index(drop=True,inplace=True)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(5039, 2)
(1243, 2)
(5039,)
(1243,)
```

Closed Form

```
from numpy.linalg import inv
weights = inv(X_train.T@X_train)@(X_train.T@Y_train)


MSE=computeMSE(Y_test,X_test,weights)
MAE=computeMAE(Y_test,X_test,weights)
```

```
print("Mean Square Error: ", MSE)
print("Mean Absolute Error: ", MAE)
```

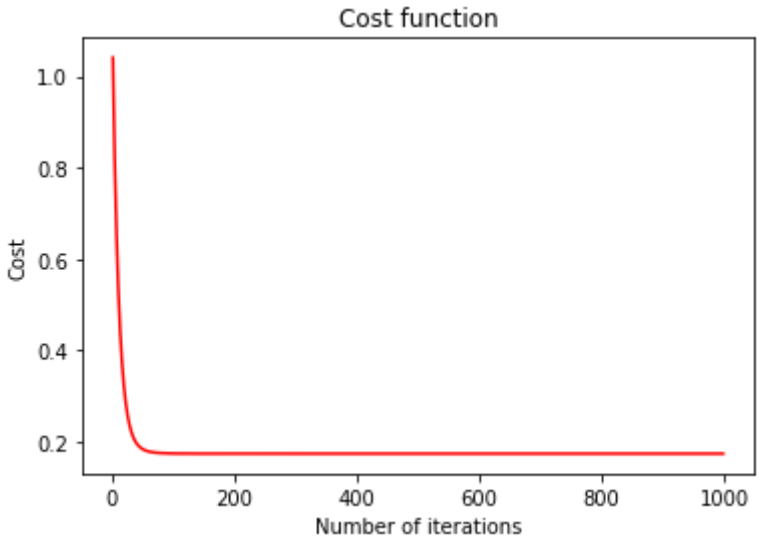
```
Mean Square Error:  0.1739664558480334
Mean Absolute Error:  0.25179601957712566
```

Gradient Descent

```
costs=[]
iterations=[]

learning_rate=.00001
itrns=1000
weights=np.ones((X_train.shape[1]))
for i in range(itrns):
    weights=weights-learning_rate*(X_train.T@(X_train@weights-Y_train))
    iterations.append(i+1)
    cost=computeMSE(Y_test,X_test,weights)
    costs.append(cost)
plt.plot(np.arange(1,itrns),costs[1:], color = 'red')
plt.title('Cost function')
plt.xlabel('Number of iterations')
plt.ylabel('Cost')
MSE=computeMSE(Y_test,X_test,weights)
MAE=computeMAE(Y_test,X_test,weights)
print("Mean Square Error: ", MSE)
print("Mean Absolute Error: ", MAE)
```

```
Mean Square Error:  0.1739664558480333
Mean Absolute Error:  0.2517960195771251
```



Colab paid products - Cancel contracts here

✓ 0s completed at 23:53