```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

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

```
co.head()
```

```
co.describe()
```

```
co.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
```

```
co.count()
```

```
Make                                7385
Model                               7385
Vehicle Class                       7385
Engine Size(L)                      7385
Cylinders                           7385
Transmission                        7385
Fuel Type                           7385
Fuel Consumption City (L/100 km)    7385
Fuel Consumption Hwy (L/100 km)     7385
Fuel Consumption Comb (L/100 km)    7385
Fuel Consumption Comb (mpg)         7385
CO2 Emissions(g/km)                 7385
dtype: int64
```

Performing one hot encoding and removing some columns for linear regression

```
co["Make"].unique().size
```

```
42
```

```
column_names_to_one_hot = ["Make","Vehicle Class","Transmission", "Fuel Type","Mode
```

```
co = pd.get_dummies(co, columns = column_names_to_one_hot)
```

```
co.loc[:20,:]
```

Removing duplicates

```
co.index[co.duplicated()]
```

```
Int64Index([1075, 1076, 1081, 1082, 1084, 1086, 1104, 1105, 1107, 1110,
            ...
            7350, 7351, 7352, 7353, 7354, 7356, 7365, 7366, 7367, 7368],
           dtype='int64', length=1103)
```

```
co.duplicated().sum()
```

```
1103
```

```
co.drop(axis="rows", labels=co.index[co.duplicated()], inplace=True)
```

```
co.duplicated().sum()
co.count()
```

```
Engine Size(L)                      6282
Cylinders                           6282
Fuel Consumption City (L/100 km)    6282
Fuel Consumption Hwy (L/100 km)     6282
Fuel Consumption Comb (L/100 km)    6282
                                    ...
Model_iM                            6282
Model_iQ                            6282
Model_tC                            6282
Model_xB                            6282
Model_xD                            6282
Length: 2150, dtype: int64
```

```
y = co["CO2 Emissions(g/km)"]
co = co.drop(columns=["CO2 Emissions(g/km)"])
```

removing Skew

```
co
```

```
co["Engine Size(L)"].describe()
```

```
count    6282.000000
mean        3.161812
std         1.365201
min         0.900000
25%         2.000000
50%         3.000000
75%         3.700000
max         8.400000
Name: Engine Size(L), dtype: float64
```

```
plt.hist(co["Engine Size(L)"], bins=100)
plt.show()
co["Engine Size(L)"] = np.log(co["Engine Size(L)"])
plt.hist(co["Engine Size(L)"], bins=100)
plt.show()
```

```
co["Cylinders"].describe()
```

```
count    6282.000000
mean        5.618911
std         1.846250
min         3.000000
25%         4.000000
50%         6.000000
75%         6.000000
max        16.000000
Name: Cylinders, dtype: float64
```

```
plt.hist(co["Cylinders"], bins=100)
plt.show()
co["Cylinders"] = np.log(np.log(co["Cylinders"]))
plt.hist(co["Cylinders"], bins=100)
plt.show()
```

```
co["Fuel Consumption City (L/100 km)"].describe()
```

```
count    6282.000000
mean       12.610220
std         3.553066
min         4.200000
25%        10.100000
50%        12.100000
75%        14.700000
max        30.600000
Name: Fuel Consumption City (L/100 km), dtype: float64
```

```
plt.hist(co["Fuel Consumption City (L/100 km)"], bins=100)
plt.show()
co["Fuel Consumption City (L/100 km)"] = np.log(co["Fuel Consumption City (L/100 km
plt.hist(co["Fuel Consumption City (L/100 km)"], bins=100)
plt.show()
```

```
co["Fuel Consumption Hwy (L/100 km)"].describe()
```

```
count    6282.000000
mean        9.070583
std         2.278884
min         4.000000
25%         7.500000
50%         8.700000
75%        10.300000
max        20.600000
Name: Fuel Consumption Hwy (L/100 km), dtype: float64
```

```
plt.hist(co["Fuel Consumption Hwy (L/100 km)"], bins=100)
plt.show()
co["Fuel Consumption Hwy (L/100 km)"] = np.log(co["Fuel Consumption Hwy (L/100 km)'
plt.hist(co["Fuel Consumption Hwy (L/100 km)"], bins=100)
plt.show()
```

```
co["Fuel Consumption Comb (L/100 km)"].describe()
```

```
count    6282.000000
mean       11.017876
std         2.946876
min         4.100000
25%         8.900000
50%        10.600000
75%        12.700000
max        26.100000
Name: Fuel Consumption Comb (L/100 km), dtype: float64
```

```
plt.hist(co["Fuel Consumption Comb (L/100 km)"], bins=100)
plt.show()
co["Fuel Consumption Comb (L/100 km)"] = np.log(co["Fuel Consumption Comb (L/100 km
plt.hist(co["Fuel Consumption Comb (L/100 km)"], bins=100)
plt.show()
```

```
co["Fuel Consumption Comb (mpg)"].describe()
```

```
count    6282.000000
mean       27.411016
std         7.245318
min        11.000000
```

```
25%        22.000000
50%        27.000000
75%        32.000000
max        69.000000
Name: Fuel Consumption Comb (mpg), dtype: float64
```

```
plt.hist(co["Fuel Consumption Comb (mpg)"], bins=100)
plt.show()
co["Fuel Consumption Comb (mpg)"] = np.log(co["Fuel Consumption Comb (mpg)"])
plt.hist(co["Fuel Consumption Comb (mpg)"], bins=100)
plt.show()
```

```
y.describe()
```

```
count    6282.000000
mean      251.157752
std        59.290426
min        96.000000
25%       208.000000
50%       246.000000
75%       289.000000
max       522.000000
Name: CO2 Emissions(g/km), dtype: float64
```

```
plt.hist(y, bins=100)
plt.show()
y = np.log(y)
plt.hist(y, bins=100)
plt.show()
```

normalization for the data

```
co = (co - co.min())/(co.max()-co.min())
co.insert(len(co.columns),"CO2 Emissions(g/km)",y )
```

```
co
```

Outliers removal

```
co
```

```
sns.boxplot(x=co["Cylinders"])
```

```
plt.show()
```

```
sns.boxplot(x=co["Fuel Consumption City (L/100 km)"])
plt.show()
```

```
sns.boxplot(x=co["Fuel Consumption Hwy (L/100 km)"])
plt.show()
```

```
sns.boxplot(x=co["Fuel Consumption Comb (L/100 km)"])
plt.show()
```

```
sns.boxplot(x=co["Fuel Consumption Comb (mpg)"])
plt.show()
```

```
sns.boxplot(x=co["CO2 Emissions(g/km)"])
plt.show()
```

```
class OutlierRemoval:
    def __init__(self, lower_quartile, upper_quartile):
        self.lower_whisker = lower_quartile - 1.5*(upper_quartile - lower_quartile)
        self.upper_whisker = upper_quartile + 1.5*(upper_quartile - lower_quartile)
    def removeOutlier(self, x):
        return (x if x <= self.upper_whisker and x >= self.lower_whisker else (self

outlier1 = OutlierRemoval(co["Cylinders"].quantile(0.25), co["Cylinders"].quantile(
t1 = co["Cylinders"].apply(outlier1.removeOutlier)
co["Cylinders"] = t1
sns.boxplot(x=co["Cylinders"])
plt.show()
```

```
outlier2 = OutlierRemoval(co["Fuel Consumption City (L/100 km)"].quantile(0.25), co
t2 = co["Fuel Consumption City (L/100 km)"].apply(outlier2.removeOutlier)
co["Fuel Consumption City (L/100 km)"] = t2
sns.boxplot(x=co["Fuel Consumption City (L/100 km)"])
plt.show()
```

```
outlier3 = OutlierRemoval(co["Fuel Consumption Hwy (L/100 km)"].quantile(0.25), co[
t3 = co["Fuel Consumption Hwy (L/100 km)"].apply(outlier3.removeOutlier)
co["Fuel Consumption Hwy (L/100 km)"] = t3
sns.boxplot(x=co["Fuel Consumption Hwy (L/100 km)"])
plt.show()
```

```
outlier4 = OutlierRemoval(co["Fuel Consumption Comb (L/100 km)"].quantile(0.25), co
t4 = co["Fuel Consumption Comb (L/100 km)"].apply(outlier4.removeOutlier)
co["Fuel Consumption Comb (L/100 km)"] = t4
sns.boxplot(x=co["Fuel Consumption Comb (L/100 km)"])
plt.show()
```

```
outlier5 = OutlierRemoval(co["Fuel Consumption Comb (mpg)"].quantile(0.25), co["Fue
t5 = co["Fuel Consumption Comb (mpg)"].apply(outlier5.removeOutlier)
co["Fuel Consumption Comb (mpg)"] = t5
sns.boxplot(x=co["Fuel Consumption Comb (mpg)"])
plt.show()
```

```
outlier5 = OutlierRemoval(co["CO2 Emissions(g/km)"].quantile(0.25), co["CO2 Emissio
t5 = co["CO2 Emissions(g/km)"].apply(outlier5.removeOutlier)
co["CO2 Emissions(g/km)"] = t5
sns.boxplot(x=co["CO2 Emissions(g/km)"])
plt.show()
```

correlation

```
co
```

```
c=co.corr()
ct = abs(c["CO2 Emissions(g/km)"])
rf = ct[ct>0.15]
rf
```

```
Engine Size(L)                          0.848637
Cylinders                               0.823469
Fuel Consumption City (L/100 km)        0.944993
Fuel Consumption Hwy (L/100 km)         0.917499
Fuel Consumption Comb (L/100 km)        0.946614
Fuel Consumption Comb (mpg)             0.946031
Make_GMC                                0.181000
Make_HONDA                              0.183049
Make_LAMBORGHINI                        0.160338
```

```
Make_MAZDA                                  0.150304
Make_MINI                                   0.171338
Make_ROLLS-ROYCE                            0.173562
Vehicle Class_COMPACT                       0.244838
Vehicle Class_MID-SIZE                      0.225906
Vehicle Class_PICKUP TRUCK - STANDARD       0.250090
Vehicle Class_STATION WAGON - SMALL         0.158419
Vehicle Class_SUV - STANDARD                0.303154
Vehicle Class_VAN - PASSENGER               0.210784
Transmission_A6                             0.163857
Transmission_A8                             0.154587
Transmission_AM6                            0.152499
Transmission_AV                             0.305265
Transmission_M5                             0.166074
Fuel Type_X                                 0.264564
Fuel Type_Z                                 0.230026
CO2 Emissions(g/km)                         1.000000
Name: CO2 Emissions(g/km), dtype: float64
```

functions

```
def mean_square_error_calculated(y_true, y_pred):
    return np.square(np.subtract(y_true,y_pred)).mean()
```

```
def mean_absolute_error_calculated(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred))
```

Done with the preprocessing, can proceed with the regression.

```
def train_test_data(hd):

    shuffle_df = hd.sample(frac=1)
    train_set = shuffle_df[:int(0.9 * len(hd))]
    X_train = train_set.drop(columns=["CO2 Emissions(g/km)"])
    Y_train = train_set["CO2 Emissions(g/km)"]

    shuffle_df = hd.sample(frac=1)
    test_set = shuffle_df[int(0.1*len(hd)):]
    X_test = test_set.drop(columns=["CO2 Emissions(g/km)"])
    Y_test = test_set["CO2 Emissions(g/km)"]


    return X_train, Y_train, X_test, Y_test
```

```
X_train, Y_train, X_test, Y_test = train_test_data(co)
```

```
ones = np.ones([X_train.shape[0],1])
X_train.insert(0,'ones', ones )
ones = np.ones([X_test.shape[0],1])
X_test.insert(0,'ones', ones )
```

```
X_train
```

```
Y_train
```

```
6064    5.416100
5421    5.293305
637     6.042633
1467    5.575949
4546    5.393628
          ...
6536    5.420535
3137    5.605802
6231    5.214936
4297    4.844205
737     5.730100
Name: CO2 Emissions(g/km), Length: 5653, dtype: float64
```

```
W = np.linalg.pinv(X_train.T@X_train)@(X_train.T@Y_train)
Y_pred = X_train @ W
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
MSE1 = mean_squared_error(Y_train, Y_pred)
MAE1 = mean_absolute_error(Y_train, Y_pred)
MSE = mean_square_error_calculated(Y_train, Y_pred)
MAE = mean_absolute_error_calculated(Y_train, Y_pred)
Y_p = X_test @ W
mse1 = mean_squared_error(Y_test,Y_p)
mae1 = mean_absolute_error(Y_test,Y_p)
mse = mean_square_error_calculated(Y_test,Y_p)
mae = mean_absolute_error_calculated(Y_test,Y_p)
```

```
mse
```

```
0.00011445128935177108
```

```
mse1
```

```
0.00011445128935177108
```

```
W.max()
```

```
3.514225144357148
```

Gradient descent

```
Y_test.shape
```

```
(5654,)
```

```
X_train.shape
```

```
(5653, 2150)
```

```
learning_rate = 0.5 #0.9
n = 1000

parameters = W[:]+0.1
nodp = co["CO2 Emissions(g/km)"].count()

costs = []
print(W)

for _ in range(n):
    y = X_train@parameters
    cost = np.mean((Y_train - y)**2)
    costs.append(cost)
    gradient_matrix = (X_train.T)@(y - Y_train)/nodp
    parameters = parameters - (learning_rate * gradient_matrix)/nodp
print(parameters)
```

```
[3.51422514e+00 2.56715457e-02 3.29256309e-04 ... 1.52825975e-02
 1.99288348e-03 1.34586891e-02]
ones                                  3.556740
Engine Size(L)                        0.094891
Cylinders                             0.074321
Fuel Consumption City (L/100 km)      0.455077
Fuel Consumption Hwy (L/100 km)       0.276576
                                        ...
Model_iM                              0.118236
Model_iQ                              0.109229
Model_tC                              0.115225
Model_xB                              0.101953
Model_xD                              0.113440
Length: 2150, dtype: float64
```

```
plt.plot(costs)
```

```
Y_pred1 = X_train@parameters
```

```
from sklearn.metrics import mean_squared_error
```

```
MSE = mean_square_error_calculated(Y_train, Y_pred1)
MAE = mean_absolute_error_calculated(Y_train, Y_pred1)
Y_p = X_test @ parameters
mse = mean_square_error_calculated(Y_test,Y_p)
mae = mean_absolute_error_calculated(Y_test,Y_p)
```

```
mse
```

```
0.5125577516024277
```

```
mae
```

```
0.7143500736144022
```

Univariate linear regression

Closed form

```
train_X = X_train.iloc[:,:4]
train_X = train_X.drop(columns=["Engine Size(L)","Cylinders"])
test_X = X_test.iloc[:,:4]
test_X = test_X.drop(columns=["Engine Size(L)","Cylinders"])
```

```
train_X.iloc[:,1:]
```

```
w0, w1 = np.linalg.pinv(train_X.T@train_X)@(train_X.T@Y_train)
Yp_train = np.array(train_X.iloc[:,:1]*w0) + np.array(train_X.iloc[:,1:]*w1)
```

```
Y_train
```

```
6064    5.416100
5421    5.293305
637     6.042633
1467    5.575949
4546    5.393628
          ...
6536    5.420535
3137    5.605802
6231    5.214936
4297    4.844205
737     5.730100
Name: CO2 Emissions(g/km), Length: 5653, dtype: float64
```

```
MSE = mean_square_error_calculated(Yp_train, np.array(Y_train))
MAE = mean_absolute_error_calculated(Yp_train, np.array(Y_train))
Yp_test = np.array(test_X.iloc[:,:1]*w0) + np.array(test_X.iloc[:,1:]*w1)
mse = mean_square_error_calculated(Yp_test, np.array(Y_test))
mae = mean_absolute_error_calculated(Yp_test, np.array(Y_test))
```

```
mse
```

```
0.1037119462279975
```

```
mae
```

```
0.2572230681583795
```

gradient descent

```
parameters = [w0-0.1, w1-0.1]

costs = []

for _ in range(n):
    y = train_X@parameters
    cost = np.mean((Y_train - y)**2)
    costs.append(cost)
    gradient_matrix = (train_X.T)@(y - Y_train)/nodp
    parameters = parameters - (learning_rate * gradient_matrix)/nodp
print(parameters)
```

```
ones                            4.550325
Fuel Consumption City (L/100 km)    1.511913
dtype: float64
```

```
plt.plot(costs)
```

```
Y_pred1 = train_X@parameters
from sklearn.metrics import mean_squared_error
```

```
MSE = mean_square_error_calculated(Y_train, Y_pred1)
MAE = mean_absolute_error_calculated(Y_train, Y_pred1)
Y_p = test_X @ parameters
mse = mean_square_error_calculated(Y_test,Y_p)
mae = mean_absolute_error_calculated(Y_test,Y_p)
```

```
mse
```

```
    0.025665877269205672

mae

    0.15084041061098277
```