Computer Assignment 0

part 1

First we import pandas library then read .csv file and store it into a data frame.

head(n): This function will returns n rows from begining of data frame.

tail(n): This function will returns n rows from end of data frame.

describe(n): This function will returns n statistics for about each column.

In [1]:

```python
import pandas
initial_data = pandas.read_csv('FuelConsumptionCo2.csv')
print("head(n) :")
print(initial_data.head(5))
print("tail(n) :")
print(initial_data.tail(5))
print("describe() :")
print(initial_data.describe())
```

```
head(n) :
   Unnamed: 0  MODELYEAR   MAKE       MODEL VEHICLECLASS  ENGINESIZE  \
0           0       2014  ACURA         ILX      COMPACT         2.0
1           1       2014  ACURA         ILX      COMPACT         2.4
2           2       2014  ACURA  ILX HYBRID      COMPACT         1.5
3           3       2014  ACURA     MDX 4WD  SUV - SMALL         3.5
4           4       2014  ACURA     RDX AWD  SUV - SMALL         3.5

   CYLINDERS TRANSMISSION FUELTYPE  FUELCONSUMPTION_CITY  FUELCONSUMPTION_HW
Y  \
0        4.0          AS5        Z                   9.9                  6.
7
1        4.0           M6        Z                  11.2                  7.
7
2        4.0          AV7        Z                   6.0                  5.
8
3        6.0          AS6        Z                  12.7                  9.
1
4        6.0          AS6        Z                  12.1                  8.
7

   FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  CO2EMISSIONS
0                   8.5                        33         196.0
1                   9.6                        29         221.0
2                   5.9                        48         136.0
3                  11.1                        25           NaN
4                  10.6                        27         244.0
tail(n) :
      Unnamed: 0  MODELYEAR   MAKE    MODEL    VEHICLECLASS  ENGINESIZE  \
1062        1062       2014  VOLVO  XC60 AWD     SUV - SMALL         3.0
1063        1063       2014  VOLVO  XC60 AWD     SUV - SMALL         3.2
1064        1064       2014  VOLVO  XC70 AWD     SUV - SMALL         3.0
1065        1065       2014  VOLVO  XC70 AWD     SUV - SMALL         3.2
1066        1066       2014  VOLVO  XC90 AWD  SUV - STANDARD         3.2

      CYLINDERS TRANSMISSION FUELTYPE  FUELCONSUMPTION_CITY  \
1062        6.0          AS6        X                  13.4
1063        6.0          AS6        X                  13.2
1064        6.0          AS6        X                  13.4
1065        6.0          AS6        X                  12.9
1066        6.0          AS6        X                  14.9

      FUELCONSUMPTION_HWY  FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  \
1062                  9.8                  11.8                        24
1063                  9.5                  11.5                        25
1064                  9.8                  11.8                        24
1065                  9.3                  11.3                        25
1066                 10.2                  12.8                        22

      CO2EMISSIONS
```

```
1062        271.0
1063        264.0
1064        271.0
1065        260.0
1066        294.0
describe() :
        Unnamed: 0  MODELYEAR   ENGINESIZE    CYLINDERS  FUELCONSUMPTION_CIT
Y  \
count  1067.000000     1067.0  1040.000000  1033.000000           1067.00000
0
mean    533.000000     2014.0     3.324038     5.797677             13.29653
2
std     308.160672        0.0     1.411400     1.807262              4.10125
3
min       0.000000     2014.0     1.000000     3.000000              4.60000
0
25%     266.500000     2014.0     2.000000     4.000000             10.25000
0
50%     533.000000     2014.0     3.300000     6.000000             12.60000
0
75%     799.500000     2014.0     4.200000     8.000000             15.55000
0
max    1066.000000     2014.0     8.400000    12.000000             30.20000
0

       FUELCONSUMPTION_HWY  FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG
\
count          1067.000000           1067.000000               1067.000000
mean              9.474602             11.580881                 26.441425
std               2.794510              3.485595                  7.468702
min               4.900000              4.700000                 11.000000
25%               7.500000              9.000000                 21.000000
50%               8.800000             10.900000                 26.000000
75%              10.850000             13.350000                 31.000000
max              20.500000             25.800000                 60.000000

       CO2EMISSIONS
count    964.000000
mean     256.741701
std       63.265308
min      108.000000
25%      209.000000
50%      251.000000
75%      294.000000
max      437.000000
```

Part 2

As we see FUELTYPE is not a numerical type so we assign numbers to each type:

In [2]:

```
print(initial_data.info())
initial_data['FUELTYPE'] = initial_data['FUELTYPE'].astype('category').cat.codes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1067 entries, 0 to 1066
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Unnamed: 0                1067 non-null   int64  
 1   MODELYEAR                 1067 non-null   int64  
 2   MAKE                      1067 non-null   object 
 3   MODEL                     1058 non-null   object 
 4   VEHICLECLASS              1067 non-null   object 
 5   ENGINESIZE                1040 non-null   float64
 6   CYLINDERS                 1033 non-null   float64
 7   TRANSMISSION              1067 non-null   object 
 8   FUELTYPE                  996 non-null    object 
 9   FUELCONSUMPTION_CITY      1067 non-null   float64
 10  FUELCONSUMPTION_HWY       1067 non-null   float64
 11  FUELCONSUMPTION_COMB      1067 non-null   float64
 12  FUELCONSUMPTION_COMB_MPG  1067 non-null   int64  
 13  CO2EMISSIONS              964 non-null    float64
dtypes: float64(6), int64(3), object(5)
memory usage: 95.9+ KB
None
```

Part 3

First we print number of NaN data in each column using number of rows subtract by number of not null data in each column.

Then replace NaN data with mean value of each column.

In the end we extract rows with NaN data in CO2EMITION column and store them in new data frame.

Number of NaN data before replacing with mean value:

In [3]:

```
print(len(initial_data.index)-initial_data.count())
initial_data['ENGINESIZE'].fillna(value=initial_data['ENGINESIZE'].mean(),inplace=True)
initial_data['CYLINDERS'].fillna(value=initial_data['CYLINDERS'].mean(),inplace=True)
nan_data = initial_data[initial_data.CO2EMISSIONS.isna()]
initial_data.dropna(subset=['CO2EMISSIONS'], inplace=True)
```

```
Unnamed: 0                 0
MODELYEAR                  0
MAKE                       0
MODEL                      9
VEHICLECLASS               0
ENGINESIZE                27
CYLINDERS                 34
TRANSMISSION               0
FUELTYPE                   0
FUELCONSUMPTION_CITY       0
FUELCONSUMPTION_HWY        0
FUELCONSUMPTION_COMB       0
FUELCONSUMPTION_COMB_MPG   0
CO2EMISSIONS             103
dtype: int64
```

Number of NaN data after replacing with mean value:

MODEL column is not a numerical value so we cannot assign mean value to NaN data.

Advantages : Easy to implement - Mean and variance will not change

Disadvantages : Results may not be accurate - For large amount of NaN data this method can increase mod value significantly and cause error in results

In [4]:

```
print(len(initial_data.index)-initial_data.count())
```

```
Unnamed: 0                 0
MODELYEAR                  0
MAKE                       0
MODEL                      9
VEHICLECLASS               0
ENGINESIZE                 0
CYLINDERS                  0
TRANSMISSION               0
FUELTYPE                   0
FUELCONSUMPTION_CITY       0
FUELCONSUMPTION_HWY        0
FUELCONSUMPTION_COMB       0
FUELCONSUMPTION_COMB_MPG   0
CO2EMISSIONS               0
dtype: int64
```

Part 4

First we filter CO2EMISSIONS then calculate mean value.

In [5]:

```
import time
start = time.time()
print(f"CO2 Emotion < 240 : {initial_data[initial_data['CO2EMISSIONS'] < 240].FUELCONSU
print(f"CO2 Emotion > 300 : {initial_data[initial_data['CO2EMISSIONS'] > 300].FUELCONSU
print(f"Executed in {time.time() - start} seconds")
```

```
CO2 Emotion < 240 : 10.03781902552204
CO2 Emotion > 300 : 18.663255813953487
Executed in 0.00899505615234375 seconds
```

Part 5

As we can see, using loop will take longer time to execute.

We must expect longer execution time in loop method but for some reason it's taking less time than vectorization.

In [6]:

```
co2_240_sum = 0
co2_240_count = 0
co2_300_sum = 0
co2_300_count = 0
for index,row in initial_data.iterrows():
    if row['CO2EMISSIONS'] < 240:
        co2_240_sum += row['FUELCONSUMPTION_CITY']
        co2_240_count += 1
    elif row['CO2EMISSIONS'] > 300:
        co2_300_sum += row['FUELCONSUMPTION_CITY']
        co2_300_count += 1
start = time.time()
print(f"CO2 Emotion < 240 : {co2_240_sum/co2_240_count}")
print(f"CO2 Emotion > 300 : {co2_300_sum/co2_300_count}")
print(f"Executed in {time.time() - start} seconds")
```

```
CO2 Emotion < 240 : 10.037819025522042
CO2 Emotion > 300 : 18.663255813953487
Executed in 0.0010004043579101562 seconds
```

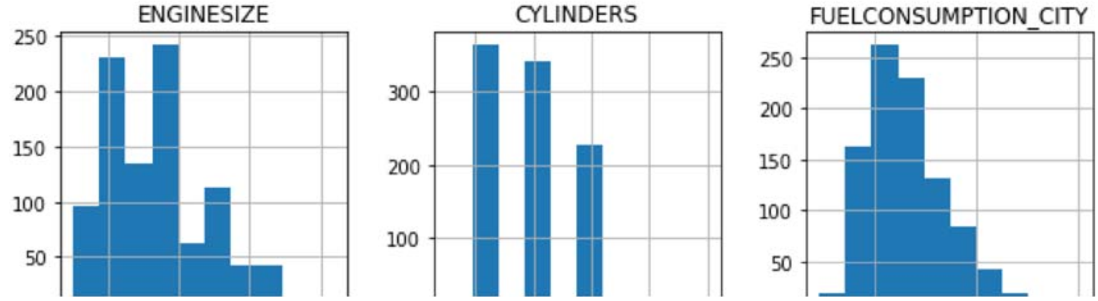
Part 6

In [7]:

```
1  initial_data.hist(column=['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPT
2                             'FUELCONSUMPTION_COMB','FUELCONSUMPTION_COMB_MPG','CO2EMISSIO
```

Out[7]:

```
array([[<AxesSubplot:title={'center':'ENGINESIZE'}>,
        <AxesSubplot:title={'center':'CYLINDERS'}>,
        <AxesSubplot:title={'center':'FUELCONSUMPTION_CITY'}>],
       [<AxesSubplot:title={'center':'FUELCONSUMPTION_HWY'}>,
        <AxesSubplot:title={'center':'FUELCONSUMPTION_COMB'}>,
        <AxesSubplot:title={'center':'FUELCONSUMPTION_COMB_MPG'}>],
       [<AxesSubplot:title={'center':'CO2EMISSIONS'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



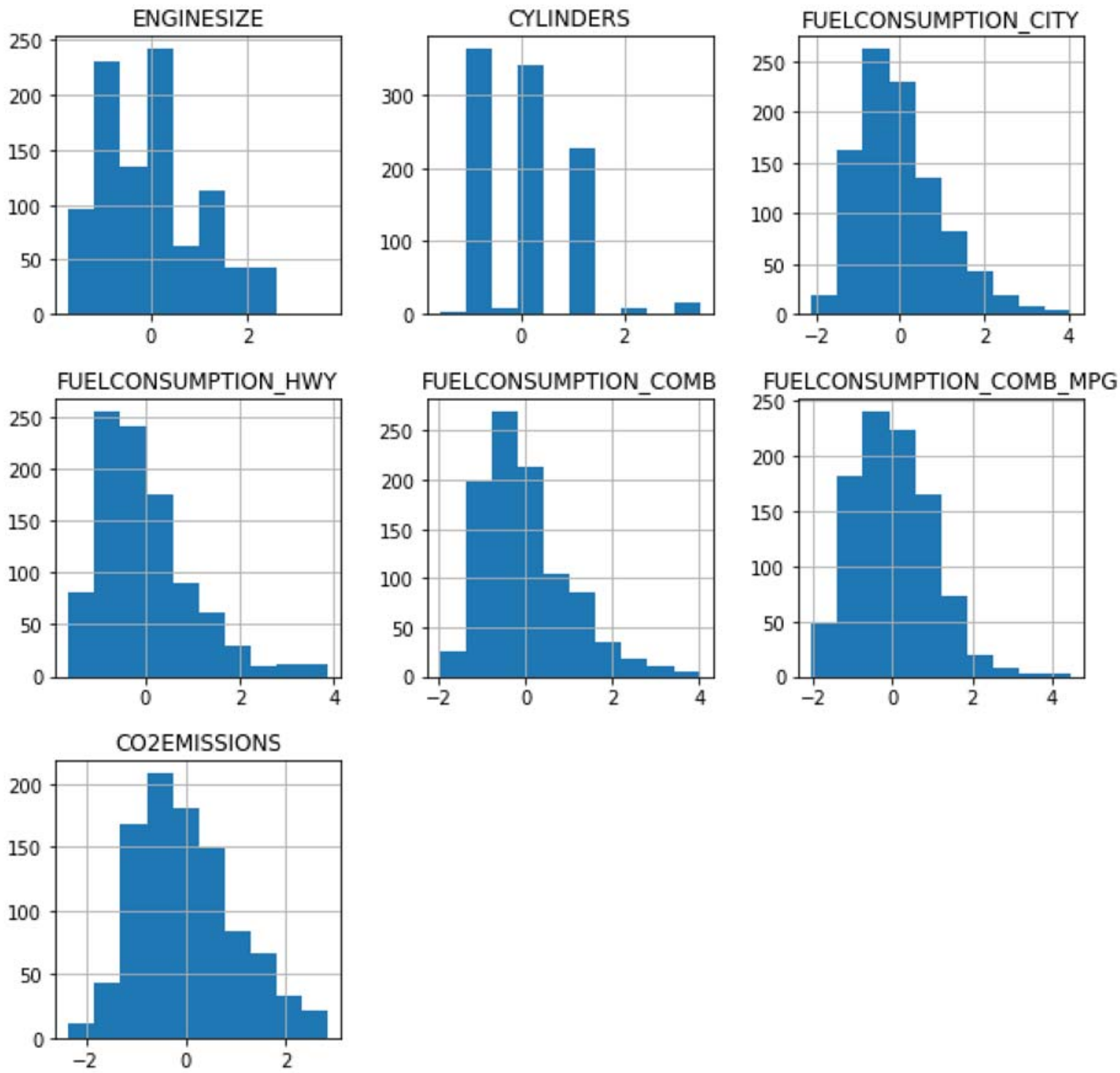
Part 7

In [8]:

```
std_co2 = initial_data['CO2EMISSIONS'].std()
mean_co2 = initial_data['CO2EMISSIONS'].mean()

normalized_data = initial_data
normalized_data['ENGINESIZE'] = \
    (initial_data['ENGINESIZE']-initial_data['ENGINESIZE'].mean())/initial_data['ENGINE
normalized_data['CYLINDERS'] = \
    (initial_data['CYLINDERS']-initial_data['CYLINDERS'].mean())/initial_data['CYLINDEF
normalized_data['FUELCONSUMPTION_CITY'] = \
    (initial_data['FUELCONSUMPTION_CITY']-initial_data['FUELCONSUMPTION_CITY'].mean())\
    /initial_data['FUELCONSUMPTION_CITY'].std()
normalized_data['FUELCONSUMPTION_HWY'] = \
    (initial_data['FUELCONSUMPTION_HWY']-initial_data['FUELCONSUMPTION_HWY'].mean())\
    /initial_data['FUELCONSUMPTION_HWY'].std()
normalized_data['FUELCONSUMPTION_COMB'] = \
    (initial_data['FUELCONSUMPTION_COMB']-initial_data['FUELCONSUMPTION_COMB'].mean())\
    /initial_data['FUELCONSUMPTION_COMB'].std()
normalized_data['FUELCONSUMPTION_COMB_MPG'] = \
    (initial_data['FUELCONSUMPTION_COMB_MPG']-initial_data['FUELCONSUMPTION_COMB_MPG'].
    /initial_data['FUELCONSUMPTION_COMB_MPG'].std()
normalized_data['CO2EMISSIONS'] = \
    (initial_data['CO2EMISSIONS']-initial_data['CO2EMISSIONS'].mean())\
    /initial_data['CO2EMISSIONS'].std()
normalized_data.hist(column=['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSU
                          'FUELCONSUMPTION_COMB','FUELCONSUMPTION_COMB_MPG','CO2EMISSIO
```

Out[8]:

```
array([[<AxesSubplot:title={'center':'ENGINESIZE'}>,
        <AxesSubplot:title={'center':'CYLINDERS'}>,
        <AxesSubplot:title={'center':'FUELCONSUMPTION_CITY'}>],
       [<AxesSubplot:title={'center':'FUELCONSUMPTION_HWY'}>,
        <AxesSubplot:title={'center':'FUELCONSUMPTION_COMB'}>,
        <AxesSubplot:title={'center':'FUELCONSUMPTION_COMB_MPG'}>],
       [<AxesSubplot:title={'center':'CO2EMISSIONS'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



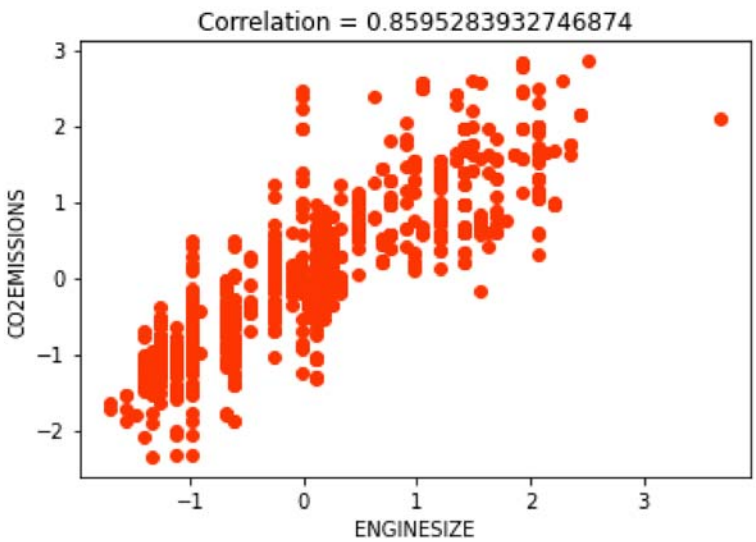
Part 8

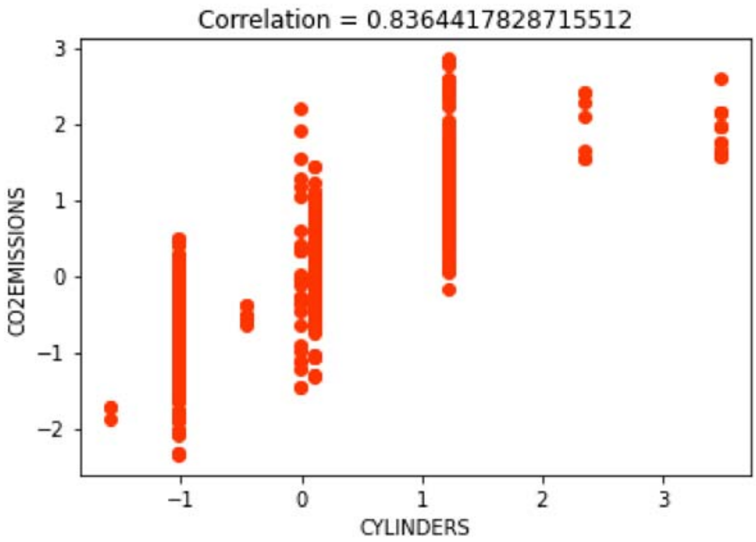
B: As we can see FUELCONSUMPTION_CITY is having the most linear relevant to CO2 emission.

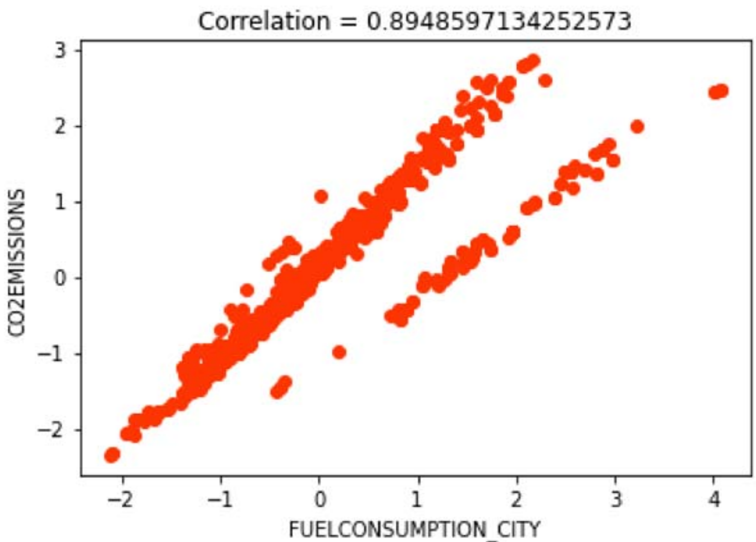
FUELCONSUMPTION_COMB_MPG is having the most correlation but it relevance in not linear.

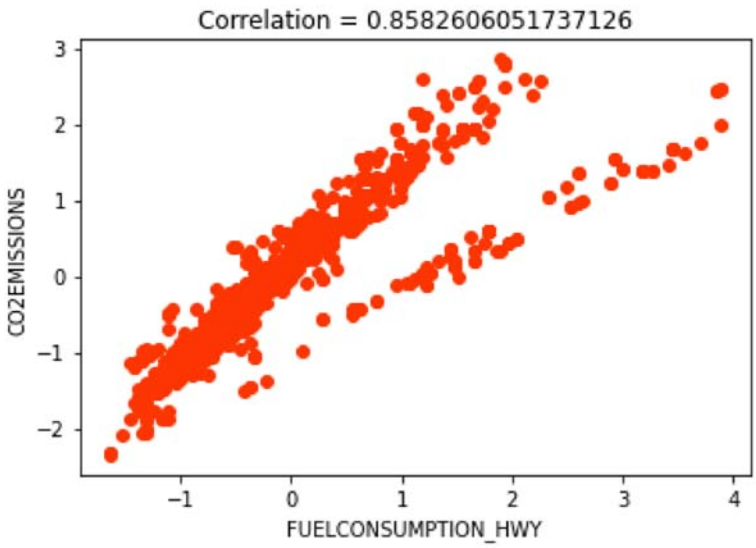
In [9]:

```
from matplotlib import pyplot

pyplot.scatter(normalized_data['ENGINESIZE'],normalized_data['CO2EMISSIONS'],c='#fc3503
pyplot.title(f"Correlation = {normalized_data['ENGINESIZE'].corr(normalized_data['CO2EM
pyplot.xlabel('ENGINESIZE')
pyplot.ylabel('CO2EMISSIONS')
pyplot.show()

pyplot.scatter(normalized_data['CYLINDERS'],normalized_data['CO2EMISSIONS'],c='#fc3503'
pyplot.title(f"Correlation = {normalized_data['CYLINDERS'].corr(normalized_data['CO2EMI
pyplot.xlabel('CYLINDERS')
pyplot.ylabel('CO2EMISSIONS')
pyplot.show()

pyplot.scatter(normalized_data['FUELCONSUMPTION_CITY'],normalized_data['CO2EMISSIONS'],
pyplot.title(f"Correlation = {normalized_data['FUELCONSUMPTION_CITY'].corr(normalized_d
pyplot.xlabel('FUELCONSUMPTION_CITY')
pyplot.ylabel('CO2EMISSIONS')
pyplot.show()

pyplot.scatter(normalized_data['FUELCONSUMPTION_HWY'],normalized_data['CO2EMISSIONS'],c
pyplot.title(f"Correlation = {normalized_data['FUELCONSUMPTION_HWY'].corr(normalized_da
pyplot.xlabel('FUELCONSUMPTION_HWY')
pyplot.ylabel('CO2EMISSIONS')
pyplot.show()

pyplot.scatter(normalized_data['FUELCONSUMPTION_COMB'],normalized_data['CO2EMISSIONS'],
pyplot.title(f"Correlation = {normalized_data['FUELCONSUMPTION_COMB'].corr(normalized_d
pyplot.xlabel('FUELCONSUMPTION_COMB')
pyplot.ylabel('CO2EMISSIONS')
pyplot.show()

pyplot.scatter(normalized_data['FUELCONSUMPTION_COMB_MPG'],normalized_data['CO2EMISSION
pyplot.title(f"Correlation = {normalized_data['FUELCONSUMPTION_COMB_MPG'].corr(normaliz
pyplot.xlabel('FUELCONSUMPTION_COMB_MPG')
pyplot.ylabel('CO2EMISSIONS')
pyplot.show()
```

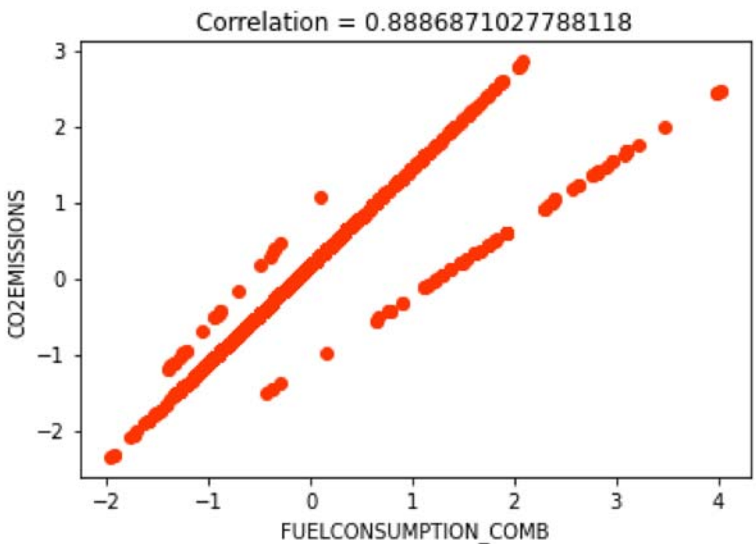



Correlation = 0.8364417828715512
CO2EMISSIONS
CYLINDERS

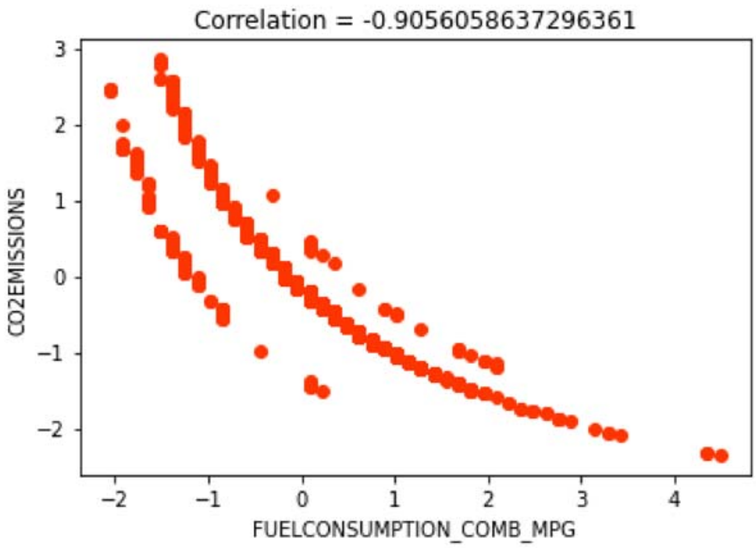

Correlation = 0.8948597134252573
CO2EMISSIONS
FUELCONSUMPTION_CITY



Correlation = 0.8582606051737126
CO2EMISSIONS
FUELCONSUMPTION_HWY



Correlation = 0.8886871027788118
CO2EMISSIONS
FUELCONSUMPTION_COMB



Part 9

In [10]:

```
data = [initial_data['FUELCONSUMPTION_CITY'], initial_data['CO2EMISSIONS']]
header = ['FUELCONSUMPTION_CITY', 'CO2EMISSIONS']
linear_data = pandas.concat(data, axis=1, keys=header)
```

Part 10

Fist we try to find $h_\theta(x)$ using formula below :

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ . & . \\ . & . \\ . & . \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_n \end{bmatrix}$$

By having $h_\theta(x)$ we calculate θ_0 and θ_1.

Then we calculate our hypothesize function $h_\theta(x)$.

In the end we calculate MSE wo verify that our error is in acceptable range.

In [11]:

```
import numpy

x_train, y_train = linear_data.values[:, 0], linear_data.values[:, 1]
x_train = x_train.reshape((len(x_train), 1))
theta = numpy.linalg.inv(x_train.T.dot(x_train)).dot(x_train.T).dot(y_train)
h = x_train.dot(theta)
theta_1 = ((x_train[0] - x_train[1]) / (h[0] - h[1]))[0]
theta_0 = h[0] - (theta_1 * x_train[0])[0]
print(f"theta_0 = {theta_0}")
print(f"theta_1 = {theta_1}")
print(f"MSE = {sum((h - y_train) ** 2) / len(h)}")
```

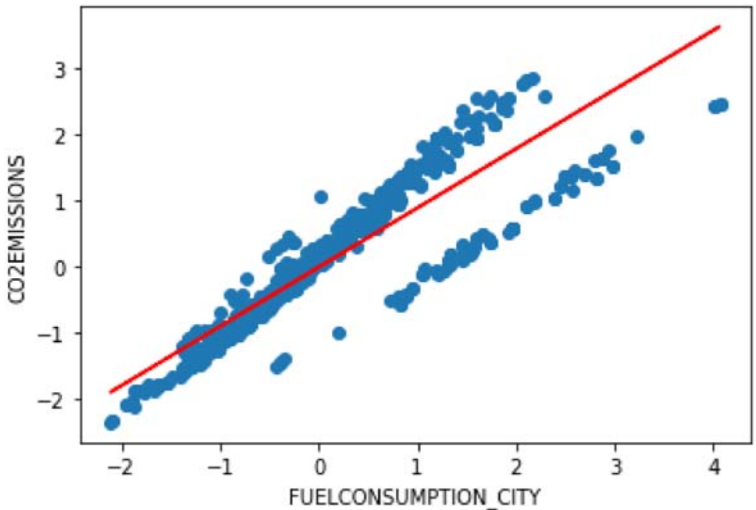
```
theta_0 = 0.1855886302194305
theta_1 = 1.1174935970380178
MSE = 0.19901942721659074
```

Part 11

As we can se in plot below, red line is our regression line. With this line we can predict how mutch a car emites CO2 by having it's cit fuel consumption.

In [12]:

```
pyplot.scatter(x_train, y_train)
pyplot.plot(x_train, h, color='red')
pyplot.xlabel('FUELCONSUMPTION_CITY')
pyplot.ylabel('CO2EMISSIONS')
pyplot.show()
```



Part 12

First we normalize FUELCONSUMPTION_CITY to match train data. Then we predict CO2EMISSIONS. After that we denormalize CO2EMISSIONS and FUELCONSUMPTION_CITY.

In [13]:

```python
import warnings
from pandas.core.common import SettingWithCopyWarning
warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)

def co2_emission(fuel_consumption):
    return (fuel_consumption*theta_1) + theta_0

std_fuel = nan_data['FUELCONSUMPTION_CITY'].std()
mean_fuel = nan_data['FUELCONSUMPTION_CITY'].mean()
nan_data['FUELCONSUMPTION_CITY'] = \
    (nan_data['FUELCONSUMPTION_CITY']-nan_data['FUELCONSUMPTION_CITY'].mean())\
    /nan_data['FUELCONSUMPTION_CITY'].std()
nan_data['CO2EMISSIONS'] = co2_emission(nan_data['FUELCONSUMPTION_CITY'])
nan_data['FUELCONSUMPTION_CITY'] = ((nan_data['FUELCONSUMPTION_CITY']*std_fuel)+mean_fu
nan_data['CO2EMISSIONS'] = (nan_data['CO2EMISSIONS']*std_co2)+mean_co2
print(nan_data.head(5))
nan_data.to_csv('Prediction_Data.csv')
```

```
    Unnamed: 0  MODELYEAR   MAKE                 MODEL    VEHICLECLASS  \
3            3       2014  ACURA               MDX 4WD     SUV - SMALL
20          20       2014   AUDI            A4 QUATTRO         COMPACT
30          30       2014   AUDI                    A8        MID-SIZE
42          42       2014   AUDI                    Q7  SUV - STANDARD
43          43       2014   AUDI  Q7 TDI CLEAN DIESEL  SUV - STANDARD

    ENGINESIZE  CYLINDERS TRANSMISSION  FUELTYPE  FUELCONSUMPTION_CITY  \
3          3.5   6.000000          AS6         3                  12.7
20         2.0   4.000000          AS8         3                  11.5
30         3.0   6.000000          AS8        -1                  13.1
42         3.0   5.797677          AS8         3                  15.1
43         3.0   6.000000          AS8         0                  12.9

    FUELCONSUMPTION_HWY  FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  \
3                   9.1                  11.1                        25
20                  8.1                  10.0                        28
30                  8.8                  11.2                        25
42                 10.9                  13.2                        21
43                  8.4                  10.9                        26

    CO2EMISSIONS
3     267.320218
20    244.139132
30    275.047246
42    313.682390
43    271.183732
```