Objective: Predicting CO2 Emissions

Data Source: Kaggle

The ability to accurately monitor carbon emissions is a critical step in the fight against climate change. Precise carbon readings allow researchers and governments to understand the sources and patterns of carbon mass output. While Europe and North America have extensive systems in place to monitor carbon emissions on the ground, there are few available in Africa.

The objective of this challenge is to create a machine learning model using open-source CO2 emissions data from Sentinel-5P satellite observations to predict future carbon emissions.

These solutions may help enable governments, and other actors to estimate carbon emission levels across Africa, even in places where on-the-ground monitoring is not possible.

Dataset Description:

Approximately 497 unique locations were selected from multiple areas in Rwanda, with a distribution around farm lands, cities and power plants. The data is split by time; the years 2019 - 2021 are included in the training data, and the task is to predict the CO2 emissions data for year 2022 through November.

In [73]:
```python
import pandas as pd

df = pd.read_csv('train.csv')
```

In [74]:
```python
df.head()
```

Out[74]:

	ID_LAT_LON_YEAR_WEEK	latitude	longitude	year	week_no	SulphurDioxide_SO2_column_n
0	ID_-0.510_29.290_2019_00	-0.51	29.29	2019	0	
1	ID_-0.510_29.290_2019_01	-0.51	29.29	2019	1	
2	ID_-0.510_29.290_2019_02	-0.51	29.29	2019	2	
3	ID_-0.510_29.290_2019_03	-0.51	29.29	2019	3	
4	ID_-0.510_29.290_2019_04	-0.51	29.29	2019	4	

5 rows × 76 columns

In [75]:
```python
path = "train.csv"
f = open(path,'r')
dataset = []
header = f.readline().strip().split(',')
```

In [76]:
```python
f
```

Out[76]:
```
<_io.TextIOWrapper name='train.csv' mode='r' encoding='cp1252'>
```

In [77]:
```
for line in f:
    line = line.split(',')
    dataset.append(line)
```

In [78]:

```
line
```

Out[78]:
```
['ID_-3.299_30.301_2021_52',
 '-3.299',
 '30.301',
 '2021',
 '52',
 '-9.12E-05',
 '0.871951342',
 '-7.95E-05',
 '0',
 '76.82563782',
 '8.273741722',
 '-135.7662048',
 '29.16049767',
 '-3.91E-05',
 '0.031550779',
 '2262.703682',
 '3132.137194',
 '829985.8087',
 '71.14586894',
 '30.52617264',
 '-140.1333923',
 '27.03770256',
 '2.64E-05',
 '-9.82E-06',
 '3.62E-05',
 '6.07E-05',
 '9579.372471',
 '-1.102551937',
 '0',
 '830171.6875',
 '76.82563782',
 '8.273741722',
 '-135.7662048',
 '29.16049767',
 '-0.000147533',
 '1.176237464',
 '-0.000234808',
 '0',
 '29.16049767',
 '-135.7662048',
 '8.273741722',
 '76.82563782',
 '-0.942917689',
 '830135.5298',
 '-0.004557078',
 '42.26529738',
 '-136.2310573',
 '30.09219564',
 '0.113322741',
 '2.635400855',
 '0.303179994',
 '227.9950024',
 '0.68387064',
 '-0.004557078',
 '42.26529738',
 '-136.2310573',
 '30.09219564',
 '',
 '',
 '',
 '',
```

```
 '',
 '',
 '',
 '0.799366849',
 '41738.45198',
 '7553.295016',
 '47771.68189',
 '6553.295018',
 '19.46403218',
 '0.226276083',
 '-12.80852786',
 '47.92344082',
 '-136.2999838',
 '30.24638689',
 '27.239302\n']
```

In [79]:
```
type(dataset)
```

Out[79]:
```
list
```

In [80]:
```
y = [float(d[75]) for d in dataset] #emissions
y[1:10]
```

Out[80]:
```
[4.0251765,
 4.231381,
 4.3052855,
 4.347317,
 4.3108187,
 4.2693343,
 4.251361,
 4.2819366,
 4.3529334]
```

In [81]:
```
data1 = [d for d in dataset if (d[6] != '' and y != '')]
y = [ float(d[75]) for d in dataset if (d[6] != '' and d[75] != '')]
```

In [82]:
```
def feature(datum):
    f = [1, float(datum[6])]
    return f
```

In [83]:
```
X = [feature(d) for d in data1]
X[:10]
```

Out[83]:
```
[[1, 0.603019416],
 [1, 0.728213548],
 [1, 0.748198569],
 [1, 0.676295994],
 [1, 0.87171331],
 [1, 0.791955829],
 [1, 0.97631079],
 [1, 0.796940641],
 [1, 0.99854094],
 [1, 0.728321351]]
```

In [84]:
```
y[:10]
```

Out[84]:
```
[3.7509942,
 4.0251765,
 4.231381,
 4.347317,
 4.3108187,
 4.2693343,
 4.251361,
 4.2819366,
 4.3529334,
 4.305424]
```

In [85]:
```
import numpy
theta, residuals, rank, s = numpy.linalg.lstsq(X,y, rcond = None)
```

In [86]:
```
theta
# Least-squares solution is y = 82.4732 + 0.26351
# But, we don't know how good or bad this solution is, so let us calculat
```

Out[86]:
```
array([82.47324369,  0.26350563])
```

In [87]:
```
residuals,rank # Rank of matrix X.
```

Out[87]:
```
(array([1.27915804e+09]), 2)
```

In [88]:
```
s # Singular values of X
```

Out[88]:
```
array([332.00713652,  35.96635915])
```

In [89]:
```
# Adding more features to our model:
# this means changing the X matrix or the feature matrix.
dfnew = df.iloc[:,[7,9,10,75]]
dfnew
```

Out[89]:

	SulphurDioxide_SO2_slant_column_number_density	SulphurDioxide_sensor_azimuth_angle
0	-0.000065	-98.593887
1	0.000014	16.592861
2	0.000385	72.795837
3	NaN	NaN
4	-0.000048	4.121269
...	...	...
79018	0.000340	72.820518
79019	0.000063	-12.856753
79020	NaN	NaN
79021	-0.000028	-100.344827
79022	-0.000079	76.825638

79023 rows × 4 columns

In [90]:
```
# Drop columns with NaNs to analyze data better
dfnew = dfnew.dropna()
dfnew
```

Out[90]:

	SulphurDioxide_SO2_slant_column_number_density	SulphurDioxide_sensor_azimuth_angle
0	-0.000065	-98.593887
1	0.000014	16.592861
2	0.000385	72.795837
4	-0.000048	4.121269
5	0.000242	-13.453690
...	...	...
79017	-0.000172	71.891731
79018	0.000340	72.820518
79019	0.000063	-12.856753
79021	-0.000028	-100.344827
79022	-0.000079	76.825638

64414 rows × 4 columns

In [91]:
```
import statsmodels.api as sm
# Define the independent variables (features) and the dependent variable
X = dfnew.iloc[:,:-1]
# Add a column of 1s to the above dataframe to include an intercept term
X = sm.add_constant(X)
y = dfnew['emission']

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Get the summary of the regression model
summary = model.summary()
print(summary)
```

```
                            OLS Regression Results                            
==============================================================================
==
Dep. Variable:               emission   R-squared:                       0.0
01
Model:                            OLS   Adj. R-squared:                  0.0
01
Method:                 Least Squares   F-statistic:                     23.
86
Date:                Thu, 24 Aug 2023   Prob (F-statistic):            1.99e-
15
Time:                        11:13:06   Log-Likelihood:           -4.1010e+
05
No. Observations:               64414   AIC:                        8.202e+
05
Df Residuals:                   64410   BIC:                        8.202e+
05
Df Model:                           3                                         
Covariance Type:            nonrobust                                         
==============================================================================
======================================
                                                     coef    std err          
t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
--------------------------------------
const                                             84.4021      1.579     53.
464      0.000      81.308      87.496
SulphurDioxide_SO2_slant_column_number_density -1.473e+04   2699.838     -5.
456      0.000      -2e+04   -9437.402
SulphurDioxide_sensor_azimuth_angle               -0.0537      0.009     -6.
204      0.000      -0.071      -0.037
SulphurDioxide_sensor_zenith_angle                -0.0432      0.039     -1.
100      0.271      -0.120       0.034
==============================================================================
==
Omnibus:                   101071.854   Durbin-Watson:                   0.0
61
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         69669318.4
35
Skew:                          10.083   Prob(JB):                         0.
00
Kurtosis:                     162.848   Cond. No.                     3.16e+
05
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
[2] The condition number is large, 3.16e+05. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

In the above solution, R2 is really low, i.e. we do not have a good selection of relevant features in our model. So, we select significant features before proceeding.

In [92]:

```
# let us try a new feature matrix, some features with high correlation wi
dfnew = []
dfnew = df.iloc[:,[9,11,12,14,15,29,32,35,49,57,59,70,75]]
dfnew = dfnew.dropna()

# Define the independent variables (features) and the dependent variable
X = dfnew.iloc[:,:-1]
# Add a column of 1s to the above dataframe to include an intercept term
X = sm.add_constant(X)
y = dfnew.iloc[:,-1]

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Get the summary of the regression model
summary = model.summary()
print(summary)
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:               emission   R-squared:                       0.0
78
Model:                            OLS   Adj. R-squared:                  0.0
52
Method:                 Least Squares   F-statistic:                     2.9
97
Date:                Thu, 24 Aug 2023   Prob (F-statistic):           0.0004
82
Time:                        11:13:06   Log-Likelihood:                -255
1.7
No. Observations:                 438   AIC:                             512
9.
Df Residuals:                     425   BIC:                             518
3.
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
================================================
                                                            coef    std e
rr          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
------------------------------------------------
const                                                 -1.457e+04   2.69e+
04     -0.542      0.588   -6.74e+04    3.83e+04
SulphurDioxide_sensor_azimuth_angle                      -0.0422      0.1
01     -0.419      0.675      -0.240       0.155
SulphurDioxide_solar_azimuth_angle                       -1.8981      1.9
19     -0.989      0.323      -5.671       1.875
SulphurDioxide_solar_zenith_angle                        -0.0755      1.8
41     -0.041      0.967      -3.694       3.543
CarbonMonoxide_CO_column_number_density               -3668.7135   1151.7
28     -3.185      0.002   -5932.506   -1404.921
CarbonMonoxide_H2O_column_number_density                 -0.0159      0.0
12     -1.378      0.169      -0.039       0.007
NitrogenDioxide_sensor_altitude                           0.0179      0.0
32      0.552      0.581      -0.046       0.082
NitrogenDioxide_solar_azimuth_angle                       2.2021      1.9
71      1.117      0.265      -1.672       6.076
Formaldehyde_tropospheric_HCHO_column_number_density_amf    22.9704     29.6
08      0.776      0.438     -35.225      81.166
Ozone_O3_column_number_density_amf                      -56.4727     32.4
36     -1.741      0.082    -120.229       7.283
UvAerosolLayerHeight_aerosol_height                       0.0050      0.0
03      1.662      0.097      -0.001       0.011
UvAerosolLayerHeight_aerosol_optical_depth                5.5461     10.0
42      0.552      0.581     -14.191      25.283
Cloud_surface_albedo                                    275.2823    126.4
81      2.176      0.030      26.677     523.888
==============================================================================
==
Omnibus:                      279.902   Durbin-Watson:                   1.3
87
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3769.5
81
Skew:                           2.527   Prob(JB):                         0.
00
Kurtosis:                      16.454   Cond. No.                     5.61e+
09
```

```
===============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
[2] The condition number is large, 5.61e+09. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

In [93]:
```
X.shape
```
Out[93]:
```
(438, 13)
```

In [94]:
```
y.shape

```
Out[94]:
```
(438,)
```

The above solution gives the line of best fit, when the coefficients are used with the predictors in the feature matrix to predict emissions. The least squares equation will look like this: emissions = const + [coeffs] * X

where coeffs are the coefficients corresponding to the columns in X matrix.

Also, in the solution given by the table above, R2 value is low, a good feature selector is needed to improve the solution. Clearly, using some of the most correlated variables may not help. Note, the correlation values are also low.

Let us explore below if modeling important variables as time series will help. (Line no 45)

In [95]:
```
def feature(datacol, ind, windowSize):
    feat = [1]
    previousValues = [float(datacol[j]) for j in X[ind - windowSize: ind]
    return feat + previousValues
```

In [96]:
```
# X_ma has consecutive 10 values with '1' appended as the first value
# in the feature data array chosen below. The feature values are
# grouped into sizes = windowSize, we cannot predict the first 'windowsize
# of y values.
windowSize = 10
N = len(dataset)
X_ma = [feature(X.iloc[1], ind, windowSize) for ind in range(windowSize,
```

In [97]:
```
# so the emissions or the y value will be predicted as:
# y_predicted = theta0 + theta1*X_ma1 + theta2*X_ma2 +...
theta, residuals, rank, s = numpy.linalg.lstsq(X,y,rcond = 0)
theta
```
Out[97]:
```
array([-1.45671999e+04, -4.21716352e-02, -1.89813520e+00, -7.55370006e-02,
       -3.66871346e+03, -1.58922645e-02,  1.79035555e-02,  2.20206693e+00,
        2.29703811e+01, -5.64727425e+01,  5.03839034e-03,  5.54609791e+00,
        2.75282313e+02])
```

In [98]:
```
# Classification with SVMs
# Can we classify emissions into areas with high emission?
# Emissions greater than 100 are harmful and below 100 are ok.
# using a logistic regression model:
from sklearn import linear_model
model = linear_model.LogisticRegression()
```

In [99]:
```
# to fit the model, y has to be a 0/1 vector
ynew = y >= 100
ynew = [int(d) for d in ynew]
model.fit(X,ynew)
predictions = model.predict(X)
```

In [100]:
```
#ynew = [print(d) for d in ynew]
```

In [101]:
```
(ynew * y)[:16]
```

Out[101]:
```
155       0.00000
451       0.00000
453       0.00000
474       0.00000
1112      0.00000
1407      0.00000
1408      0.00000
1566      0.00000
1568      0.00000
1726      0.00000
1727      0.00000
1883      0.00000
1884      0.00000
2528      0.00000
2541      0.00000
2677    100.72777
Name: emission, dtype: float64
```

In [102]:
```
correctPredictions = predictions == ynew
correctPredictions_train = sum(correctPredictions)/len(correctPredictions
correctPredictions_train
```

Out[102]:
```
0.7397260273972602
```

The classifier is approximately 74 % accurate on training data. Let us now predict the emissions for test data. Note that we cannot check the accuracy in the test data, as emissions data is not available.

In [103]:
```
# upload the test data
test = pd.read_csv("test.csv")
# predict for the test data
Xtest = []
# Define the independent variables (features) as before
Xtest = test.iloc[:,[9,11,12,14,15,29,32,35,49,57,59,70]]
Xtest = Xtest.dropna()
# Add a column of 1s to the above dataframe to include an intercept term
Xtest = sm.add_constant(Xtest)
predictions_test = model.predict(Xtest)
sum(predictions_test)
```

Out[103]: 9

As the sum of predictions_test is 9, the model predicts > 100 emissions for 9 days in the test sample and less than 100 emissions for the remaining days.

Next, we model the data based on gradient descent in Python.

In [104]:
```
# This is the training dataset with 12 feature vectors and one emission ve
dfnew
```

Out[104]:

	SulphurDioxide_sensor_azimuth_angle	SulphurDioxide_solar_azimuth_angle	SulphurDioxid
155	-42.108062	-132.844392	
451	4.135166	-36.356835	
453	4.431898	-39.355122	
474	74.048912	-144.187874	
1112	74.313675	-144.471344	
...	...	...	
78045	4.671381	-37.555971	
78203	74.629140	-32.312084	
78205	-50.365193	-44.618462	
78216	75.047003	-105.262979	
78858	75.432152	-128.411514	

438 rows × 13 columns

In [105]:
```
X = dfnew.iloc[:,0:12]  # feature matrix
X = sm.add_constant(X)  # add a column of 1s to X
y = dfnew.iloc[:,12]   # labels
```

In [106]:
```
# feature dimension:
K = X.shape[1]
K
```

Out[106]: 13

In [107]:
```
# Initialize parameters
theta = [0.0]*K

# theta[0] is the intercept term in the model, so we calculate the mean o
# to initialize theta[0] to mean and then, try to improve our model
theta[0] = sum(y)/len(y)
theta[0]
```

Out[107]: 72.81695076772596

In [108]:
```
def inner(x,y):
    return sum([a*b for (a,b) in zip(x,y)])

def norm(x):
    return sum([a*a for a in x])
```

In [109]:
```
inner(X.iloc[1], theta)
```

Out[109]: 72.81695076772596

In [110]:
```
import numpy as np

# Compute the partial derivative:
def derivative(X, y, theta):
    dtheta = [0.0] * (len(theta))
    K = len(theta)
    N = X.shape[0]
    y = np.array(y)
    MSE = 0
    for i in range(N):
        error = inner(X.iloc[i].to_numpy(), np.array(theta)) - y[i]
        for k in range(K):
            dtheta[k] += 2*float(X.iloc[i][k])*error/N
        MSE += error/N
    return dtheta, MSE
```

In [111]:
```
#derivative(X,y,theta)
#error = inner(np.array(X.iloc[i].tolist()), np.array(theta)) # - y[i]

a = X.iloc[1].to_numpy()
b = np.array(theta)
inner(a,b)
#X.iloc[1]
N = X.shape[1]
N
```

Out[111]: 13

In [112]:
```
# learning rate is how much theta will be updated in the
# direction of the derivative.
learningRate = 0.003
```

In [113]:
```
len(theta)
```

Out[113]: 13

In [114]:

```
while(True):
    dtheta, MSE = derivative(X,y,theta)
    m = norm(dtheta)
    print("norm(dtheta) = " + str(m) + "MSE = " + str(MSE))
    for k in range(K):
        theta[k] -= learningRate*dtheta[k]
        # theta[k] changes positively or negatively
        # based on direction of derivative.
    if m < 0.001: break
```

```
norm(dtheta) = 368489065.1809829MSE = -7.189387973838279e-14
norm(dtheta) = 1.8104479071609532e+24MSE = -810557.9262026103
norm(dtheta) = 3.093138876900812e+43MSE = 3350363896565680.5
norm(dtheta) = 5.284608501736313e+62MSE = -1.384837068175093e+25
norm(dtheta) = 9.028720703483362e+81MSE = 5.724075845437148e+34
norm(dtheta) = 1.5425513075325398e+101MSE = -2.365985503803275e+44
norm(dtheta) = 2.635439299226882e+120MSE = 9.779547922429262e+53
norm(dtheta) = 4.5026316246294304e+139MSE = -4.042271493775085e+63
norm(dtheta) = 7.692718080458336e+158MSE = 1.670829670143666e+73
norm(dtheta) = 1.314296091683516e+178MSE = -6.906195664817269e+82
norm(dtheta) = 2.2454666849193196e+197MSE = 2.8546020826072486e+92
norm(dtheta) = 3.836365842512672e+216MSE = -1.1799192269541995e+102
norm(dtheta) = 6.554407142373946e+235MSE = 4.877069874707789e+111
norm(dtheta) = 1.119816377049831e+255MSE = -2.0158846486621047e+121
norm(dtheta) = 1.9131993040255756e+274MSE = 8.332443498064532e+130
norm(dtheta) = 3.2686890921948594e+293MSE = -3.4441263638034425e+140

C:\Users\trlal\AppData\Local\Temp\ipykernel_6492\3859210855.py:5: RuntimeW
arning: overflow encountered in double_scalars
  return sum([a*a for a in x])
```

The algorithm works if the model is a good fit to the data. If it is a good fit, we keep running this algorithm until the derivative is very small.

Let us check the gradient descent algorithm in Tensorflow library.

In [115]:

```
import tensorflow as tf
```

In [116]:

```
path
```

Out[116]: 'train.csv'

In [117]:
```
1 X
```

Out[117]:

	const	SulphurDioxide_sensor_azimuth_angle	SulphurDioxide_solar_azimuth_angle	Sulphu
155	1.0	-42.108062	-132.844392	
451	1.0	4.135166	-36.356835	
453	1.0	4.431898	-39.355122	
474	1.0	74.048912	-144.187874	
1112	1.0	74.313675	-144.471344	
...	...	...	...	
78045	1.0	4.671381	-37.555971	
78203	1.0	74.629140	-32.312084	
78205	1.0	-50.365193	-44.618462	
78216	1.0	75.047003	-105.262979	
78858	1.0	75.432152	-128.411514	

438 rows × 13 columns

In [118]:
```
1 y = tf.constant(y, shape = [len(y),1])
```

In [119]:
```
1 y
```

Out[119]:
```
<tf.Tensor: shape=(438, 1), dtype=float64, numpy=
array([[4.68789800e+00],
       [6.37902560e-01],
       [6.27023100e-01],
       [6.18268550e-01],
       [8.41614460e+01],
       [4.67755660e+01],
       [4.60246900e+01],
       [3.25936100e+01],
       [3.69537620e+01],
       [2.16543480e+01],
       [2.49670220e+01],
       [9.35163600e+01],
       [9.36049800e+01],
       [2.64713350e-01],
       [2.34416930e-01],
       [1.00727770e+02],
       [9.65314200e+01],
       [9.94578800e+01],
```

In [120]:
```
X
```

Out[120]:

	const	SulphurDioxide_sensor_azimuth_angle	SulphurDioxide_solar_azimuth_angle	Sulphu
155	1.0	-42.108062	-132.844392	
451	1.0	4.135166	-36.356835	
453	1.0	4.431898	-39.355122	
474	1.0	74.048912	-144.187874	
1112	1.0	74.313675	-144.471344	
...	...	...	...	
78045	1.0	4.671381	-37.555971	
78203	1.0	74.629140	-32.312084	
78205	1.0	-50.365193	-44.618462	
78216	1.0	75.047003	-105.262979	
78858	1.0	75.432152	-128.411514	

438 rows × 13 columns

In [121]:
```
theta = tf.cast(theta, dtype=tf.float64)
def MSE(X,y, theta):
    return tf.reduce_mean((tf.matmul(X,theta) - y)**2)
```

In [122]:
```
theta = tf.Variable(tf.constant([0.0]*K, shape = [K,1]))
```

In [123]:
```
# initializing values
init = tf.Variable(initial_value=0.0) # tf.global_variables_initializer()
```

In [124]:
```
# find the optimium
opt = tf.keras.optimizers.Adam(learning_rate = 0.01)
```

In [125]:
```
@tf.function
def MSE(X,y, theta):
    return tf.reduce_mean((tf.matmul(X,theta) - y)**2)

```

In [126]:
```
theta = tf.Variable(tf.constant([0.0*K], shape = [K,1]))
```

In [127]:

```
from tensorflow.keras.optimizers import Adam
optimizer = Adam(learning_rate=0.01)

trainable_vars = [theta]
epochs = 100

y = tf.cast(y, dtype=tf.float32)
X = tf.constant(X, dtype = tf.float32)
for _ in range(epochs):
    with tf.GradientTape() as tp:
        objective = MSE(X,y,theta)
    gradients = tp.gradient(objective, trainable_vars)
    optimizer.apply_gradients(zip(gradients, trainable_vars))
```

In [134]:

```
objective
```

Out[134]:

```
<tf.Tensor: shape=(), dtype=float32, numpy=9360.322>
```

In [128]:

```
y
```

Out[128]:

```
<tf.Tensor: shape=(438, 1), dtype=float32, numpy=
array([[4.68789816e+00],
       [6.37902558e-01],
       [6.27023101e-01],
       [6.18268549e-01],
       [8.41614456e+01],
       [4.67755661e+01],
       [4.60246887e+01],
       [3.25936089e+01],
       [3.69537621e+01],
       [2.16543484e+01],
       [2.49670219e+01],
       [9.35163574e+01],
       [9.36049805e+01],
       [2.64713347e-01],
       [2.34416932e-01],
       [1.00727768e+02],
       [9.65314178e+01],
       [9.94578781e+01],
```

In [129]:

```
type(X)
```

Out[129]:

```
tensorflow.python.framework.ops.EagerTensor
```

In [135]:
```
X
```

Out[135]:
```
<tf.Tensor: shape=(438, 13), dtype=float32, numpy=
array([[ 1.0000000e+00, -4.2108063e+01, -1.3284439e+02, ...,
          4.3619033e+03,  1.4891764e+00,  2.6723665e-01],
       [ 1.0000000e+00,  4.1351662e+00, -3.6356834e+01, ...,
          2.5273311e+03,  1.7923474e-01,  1.6723536e-01],
       [ 1.0000000e+00,  4.4318981e+00, -3.9355122e+01, ...,
          7.3255718e+03,  5.8069664e-01,  1.7466491e-01],
       ...,
       [ 1.0000000e+00, -5.0365192e+01, -4.4618462e+01, ...,
          2.3564241e+03,  9.7931260e-01,  1.7757662e-01],
       [ 1.0000000e+00,  7.5047005e+01, -1.0526298e+02, ...,
          3.3142532e+03,  7.8009897e-01,  2.3359303e-01],
       [ 1.0000000e+00,  7.5432152e+01, -1.2841151e+02, ...,
          3.1560845e+03,  1.0498621e+00,  2.5640440e-01]], dtype=float32)>
```

In [136]:
```
tf.print(theta)
```

```
[[0.000134186601]
 [0.00259684678]
 [-0.000800135895]
 ...
 [0.00103928975]
 [0.00108676369]
 [0.00098184275]]
```

The above solution of thetas can be obtained faster by using tensorflow library in Python.

In [ ]:
```

```