Question 1

Read the following highly-cited article by Fearon and Laitin (2003):
https://cisac.fsi.stanford.edu/publications/ethnicity_insurgency_and_civil_war

Answer the following:

a. What is the paper about? (Please write a paragraph – max. 250 words)

b. How many observations do authors have in their dataset? What does each observation represent? (=What is the unit of analysis?)

c. What is the identification strategy of the authors? (=What are the different regression equations they are running?) Please write down in form of equations and explain. Identify the independent and dependent variables.

d. What do the coefficient values listed in Table 1 represent? (theoretically speaking)

e. Which independent variables have positive coefficients? Which independent variables have negative coefficients? Which ones are statistically significant?

f. Thinking about the range of your independent variables, which variables do you think have a greater impact on the dependent variable(s)?

Part a

The authors of the paper, James D. Fearon and David D. Laitin study and evaluate the possible root causes of the many civil wars that took place in the United States after the Cold War between the mid to late 1900s. They perform various analyses using factors which might be possible influences on civil strife such as Per Capita Income, Ethnic and Religious Composition, Democracy and Civil Liberties, among many others. They also disprove a few commonly believed theories such as the civil wars taking place due to changes in the international system, more ethnic and religious diversity causing such civil conflicts and the ability to predict the occurence of a civil war based on the strength of political and ethnic grievances of the people of a nation. They lay out many such hypotheses based on common arguments and rebut them using solid evidence they gathered from their data. Moreover, they ellaborate on what common misconception might have led to such false notions regarding the matter. They conclude several interesting things at the end of the study such as the conditions favoring civil wars not being ethnic and religious diversity against popular belief. In fact, conditions that favour insurgency were found to be state weakness ["Decolonization from the 1940s through

the 1970s gave birth to a large number of financially, bureaucratically, and militarily weak states"] indicated by the population, poverty and instability within a state.

Part b

The authors had data on about 127 civil war starts in a sample of 6,610 country years.

The unit of analysis refers to the main parameter that one is investigating in a research project or study. The authors' objective was to study the susceptibility of a country to a civil war. Hence, the unit of analysis in the study are countries.

Part c

control variable: Prior war - x_1

independent variables:

1. Per capita income - x_2
2. log(population) - x_3
3. log(% mountainous)- x_4
4. Noncontiguous state - x_5
5. Oil exporter - x_6
6. New state - x_7
7. Instability - x_8
8. Democracy (Polity IV) - x_9
9. Ethnic fractionalization - x_{10}
10. Religious fractionalization - x_{11}
11. Anocracy - x_{12}
12. Democracy (Dichotomous) - x_{13}

dependent variable: onset - Y

The following regression equations were ran by the authors:

Model 1:

$$-6.731 - (0.954 * x_1) - (0.344 * x_2) + (0.263 * x_3) + (0.219 * x_4) + (0.443 * x_5) + (0.858 * x_6) + (1.709 * x_7) + (0.618 * x_8) + (0.021 * x_9) + (0.166 * x_{10}) + (0.285 * x_{11})$$

Model 2:

$$-8.450 - (0.849 * x_1) - (0.379 * x_2) + (0.389 * x_3) + (0.120 * x_4) + (0.481 * x_5) + (0.809 * x_6) + (1.777 * x_7) + (0.385 * x_8) + (0.013 * x_9) + (0.146 * x_{10}) + (1.533 * x_{11})$$

Model 3:

$$-7.019 - (0.916 * x_1) - (0.318 * x_2) + (0.272 * x_3) + (0.199 * x_4) + (0.426 * x_5) + (0.751 * x_6) + (1.658 * x_7) + (0.513 * x_8) + (0.164 * x_{10}) + (0.326 * x_{11}) + (0.521 * x_{12}) + (0.127 * x_{13})$$

Model 4:

$$-7.019 - (0.688 * x_1) - (0.305 * x_2) + (0.267 * x_3) + (0.192 * x_4) + (0.798 * x_5) + (0.548 * x_6) + (1.523 * x_7) + (0.548 * x_8) + (0.490 * x_{10})$$

Model 5:

$$-6.801 - (0.551 * x_1) - (0.309 * x_2) + (0.223 * x_3) + (0.418 * x_4) - (0.171 * x_5) + (1.269 * x_6) + (1.147 * x_7) + (0.584 * x_8) - (0.119 * x_9) + (1.176 * x_{11}) + (0.597 * x_{12}) + (0.219 * x_{13})$$

Part d

Generally speaking, the coefficients describe the relationship between the independent variables and the dependent variable.

In this scenario, a high positive coefficent means that an increase in the value of the variable with the high positive coefficient will increase the likelihood of the war occuring. Similarly, a high negative coefficient would mean that an increase in the value of the independent variable with the high negative coefficient would entail a decrease in the likelihood of the war occuring.

For example, the independent variable "Instability" has a high positive coefficient, this means that theoretically, an increase in the political instability of antion would increase the likelihood of a war occuring. And, the negative coefficient for the independent variable "Per Capita Income" suggests that an increase in the per capita income of the nation would decrease the likelihood of a war occuring.

Part e

Independent variables with positive coeffiecients:

1. log(population)

2. log(% mountainous)
3. Noncontiguous state
4. Oil exporter
5. New state
6. Instability
7. Democracy (Polity IV)
8. Ethnic fractionalization
9. Religious fractionalization
10. Anocracy
11. Democracy (Dichotomous)

Independent varaibles with negative coefficients:

1. Prior war
2. Per capita income

Consider the following models:

1 - Civil War 2 - Ethnic War 3 - Civil War 4 - Civil War (Plus Empires) 5 - Civil War(COW)

The confidence levels were set at 95%. The following variables are statistically significant for each model respectively:

Model 1:

1. Prior war
2. Per capita income
3. log(population)
4. log(% mountainous)
5. Oil exporter
6. New state
7. Instability

Model 2:

1. Prior war
2. Per capita income

3. log(population)
4. Oil exporter
5. New state
6. Religious fractionalization

Model 3:

1. Prior war
2. Per capita income
3. log(population)
4. log(% mountainous)
5. Oil exporter
6. New state
7. Instability
8. Anocracy

Model 4:

1. Prior war
2. Per capita income
3. log(population)
4. log(% mountainous)
5. Noncontiguous state
6. Oil exporter
7. New state
8. Instability

Model 5:

1. Per capita income
2. log(population)
3. log(% mountainous)
4. Oil exporter
5. New state
6. Instability

7. Religious fractionalization
8. Anocracy

Part f

The independent variables that have the greatest impact on the dependent variable are:

1. Per capita income
2. log(population)
3. New state

Question 2:

Build a two-class logistic regression model from scratch. You will need to work on the following:

```
a. Implement the sigmoid function from scratch and call it sigmoid_f

b. Implement the hypothesis function from scratch and call it classifier_f

c. Implement the entropy function as your cost function and call it binary_loss_f

d. Implement gradient descent for logistic regression and call it gradient_f

e. Combining the functionalities of what you have coded above, create an optimizer
function and call it optimizer_f.
```

Note: You should find out the input and output to the functions above by reviewing the class notes and the textbook; in other words, this will be part of the challenge! If needed, use 265 as your random seed.

Let's test your code on a dataset. Load the Breast Cancer Wisconsin Dataset provided by sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_cancer

Now, do the following:

```
a. Set the target column as your Y variable.
```

b. Set all other numeric variables (excluding index) as your X matrix.

c. Apply 0-1 normalization on both the X matrix and Y vector.

d. Run logistic regression by using the code you have written (no need to do train/test split). Set the maximum number of iterations to 10,000.

e. Report the final equation you have obtained for logistic regression.

f. Also indicate which coefficients are positively associated and which coefficients are negatively associated with the target variable. Rank them from positive to negative. Interpret the results.

In [65]:

```
import math
import numpy as np
import matplotlib.pyplot as plt

def sigmoid_f(z):
    s_val = 1/(1 + math.e**(-z)) # returns the result of the sigmoid funtion
    return (s_val)

def classifier_f(X, weights):
    z = np.dot(X, weights) # calculates predicted values for dependent variable
    return sigmoid_f(z)

def binary_loss_f(y_pred, y):
    return sum(-y*np.log(y_pred) - (1-y)*np.log(1-y_pred))/len(y) # returns binary loss value

def gradient_f(X, weights, learning_rate, num_iters):
    weights = np.random.random(X.shape[1])
    costs = []
    for epoch in range(num_iters):
        y_pred = classifier_f(X, weights) # predicted values
        loss = y_pred - y
        weights_grad = X.T.dot(loss)
        weights -= learning_rate * weights_grad # updates the weight vector
        cost = binary_loss_f(y_pred, y) # calculates cost
        costs.append(cost) # saves the cost history
        if (epoch%100==0):
            print(f"Epoch :{epoch} || Cost: {float(cost)}")
    return weights, costs

def optimizer_f(X, y, learning_rate, num_iters):
```

```
    weights, cost = gradient_f(X, y, learning_rate, num_iters) # returns weight and cost vectors
    print("weights = ", weights)
    plt.plot(cost)
    return weights
```

In [66]:

```
import sklearn.datasets
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_breast_cancer

# load dataset

breast_cancer = load_breast_cancer()
X = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
X.head()
```

Out[66]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	perir
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	18
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	15
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	15
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	15

5 rows × 30 columns

In [67]:

```
y = breast_cancer.target
```

In [68]:

```
# scale the data

Scaler = MinMaxScaler()
X = Scaler.fit_transform(X)
```

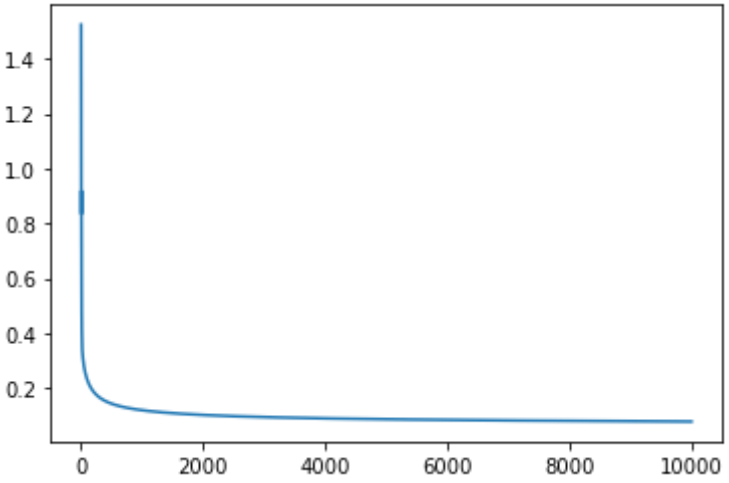
In [71]:
```
# set learning rate and epochs

number_of_iterations = 10000
learning_rate = 0.01
weights_opt = optimizer_f(X, y, learning_rate = learning_rate,num_iters = number_of_iterations)
```

```
Epoch :0 || Cost: 1.5250591192914391
Epoch :100 || Cost: 0.230228010005412
Epoch :200 || Cost: 0.18737108257039845
Epoch :300 || Cost: 0.16545774264825774
Epoch :400 || Cost: 0.1518439668600528
Epoch :500 || Cost: 0.1424101186720937
Epoch :600 || Cost: 0.1353939701150019
Epoch :700 || Cost: 0.1299137397319381
Epoch :800 || Cost: 0.1254774935286902
Epoch :900 || Cost: 0.12178785796530608
Epoch :1000 || Cost: 0.11865369261299923
Epoch :1100 || Cost: 0.1159460656655422
Epoch :1200 || Cost: 0.11357456363292885
Epoch :1300 || Cost: 0.11147372738668787
Epoch :1400 || Cost: 0.1095948781441754
Epoch :1500 || Cost: 0.10790097606723655
Epoch :1600 || Cost: 0.10636326759170042
Epoch :1700 || Cost: 0.10495903170656992
Epoch :1800 || Cost: 0.10367002579590036
Epoch :1900 || Cost: 0.10248139070443248
Epoch :2000 || Cost: 0.10138086521559592
Epoch :2100 || Cost: 0.10035821350971302
Epoch :2200 || Cost: 0.09940480174774173
Epoch :2300 || Cost: 0.09851328050417289
Epoch :2400 || Cost: 0.09767734320568809
Epoch :2500 || Cost: 0.0968915397496337
Epoch :2600 || Cost: 0.09615113065227944
Epoch :2700 || Cost: 0.09545197135486314
Epoch :2800 || Cost: 0.09479041929202543
Epoch :2900 || Cost: 0.09416325839998958
Epoch :3000 || Cost: 0.0935676371862991
Epoch :3100 || Cost: 0.09300101749298484
Epoch :3200 || Cost: 0.09246113179688345
Epoch :3300 || Cost: 0.0919459473988434
Epoch :3400 || Cost: 0.0914536362220907
Epoch :3500 || Cost: 0.09098254921241285
Epoch :3600 || Cost: 0.09053119453797821
Epoch :3700 || Cost: 0.09009821894387476
Epoch :3800 || Cost: 0.08968239173885709
Epoch :3900 || Cost: 0.08928259098826333
Epoch :4000 || Cost: 0.08889779156385214
Epoch :4100 || Cost: 0.08852705476292833
```

```
Epoch :4200 || Cost: 0.08816951925886296
Epoch :4300 || Cost: 0.08782439318546412
Epoch :4400 || Cost: 0.08749094719055595
Epoch :4500 || Cost: 0.08716850832101268
Epoch :4600 || Cost: 0.08685645462358967
Epoch :4700 || Cost: 0.08655421036408326
Epoch :4800 || Cost: 0.08626124178239497
Epoch :4900 || Cost: 0.08597705331354501
Epoch :5000 || Cost: 0.08570118421506778
Epoch :5100 || Cost: 0.08543320554988547
Epoch :5200 || Cost: 0.08517271748102537
Epoch :5300 || Cost: 0.08491934684064997
Epoch :5400 || Cost: 0.08467274494102328
Epoch :5500 || Cost: 0.08443258559939826
Epoch :5600 || Cost: 0.08419856335250848
Epoch :5700 || Cost: 0.08397039183950787
Epoch :5800 || Cost: 0.08374780233489058
Epoch :5900 || Cost: 0.0835305424152382
Epoch :6000 || Cost: 0.0833183747456214
Epoch :6100 || Cost: 0.08311107597320025
Epoch :6200 || Cost: 0.08290843571704098
Epoch :6300 || Cost: 0.08271025564445245
Epoch :6400 || Cost: 0.08251634862525672
Epoch :6500 || Cost: 0.08232653795638176
Epoch :6600 || Cost: 0.08214065665000639
Epoch :6700 || Cost: 0.08195854677923345
Epoch :6800 || Cost: 0.08178005887591262
Epoch :6900 || Cost: 0.08160505137581052
Epoch :7000 || Cost: 0.08143339010682646
Epoch :7100 || Cost: 0.08126494781639315
Epoch :7200 || Cost: 0.08109960373460137
Epoch :7300 || Cost: 0.08093724316993021
Epoch :7400 || Cost: 0.08077775713477284
Epoch :7500 || Cost: 0.08062104199822716
Epoch :7600 || Cost: 0.08046699916386073
Epoch :7700 || Cost: 0.08031553477037595
Epoch :7800 || Cost: 0.0801665594133038
Epoch :7900 || Cost: 0.08001998788601984
Epoch :8000 || Cost: 0.07987573893853792
Epoch :8100 || Cost: 0.07973373505267486
Epoch :8200 || Cost: 0.07959390223230796
Epoch :8300 || Cost: 0.07945616980755486
Epoch :8400 || Cost: 0.07932047025181511
Epoch :8500 || Cost: 0.07918673901069848
Epoch :8600 || Cost: 0.07905491434195017
Epoch :8700 || Cost: 0.0789249371655625
Epoch :8800 || Cost: 0.07879675092332199
Epoch :8900 || Cost: 0.07867030144711228
Epoch :9000 || Cost: 0.07854553683534181
```

```
Epoch :9100 || Cost: 0.07842240733692085
Epoch :9200 || Cost: 0.07830086524225674
Epoch :9300 || Cost: 0.07818086478077768
Epoch :9400 || Cost: 0.07806236202453698
Epoch :9500 || Cost: 0.07794531479747985
Epoch :9600 || Cost: 0.07782968258999315
Epoch :9700 || Cost: 0.0777154264783811
Epoch :9800 || Cost: 0.07760250904894211
Epoch :9900 || Cost: 0.07749089432634176
weights =  [ 21.83938371   1.59679199  17.79131246  -9.84279612   6.81421289
  -3.51228902 -16.36068281 -28.8908349    8.90161143  21.48705721
 -16.73535483   7.4374776   -9.6636914  -20.12650306  -3.2161601
  -0.16602327   7.4048269    3.71846032   9.01465592   3.58531003
  -2.63448381  -9.6823236    1.2701837  -23.4098471   -0.11542564
   1.19253629   2.49233147  -0.43311392 -14.54950763  -9.02942077]
```



In [73]:

```
# print log reg equation

eq = ""
for i, col in enumerate(breast_cancer.feature_names):
    if i == 0:
        eq += f"{weights_opt[i]}*{col}"
        eq += " "
    else:
        eq += f"+({weights_opt[i]})*{col}"
        eq += " "
print(eq)
```

```
21.839383705191462*mean radius +(1.596791990260688)*mean texture +(17.791312463824244)*mean perimeter +(-9.8427
96122741028)*mean area +(6.814212888124369)*mean smoothness +(-3.5122890164092)*mean compactness +(-16.36068280
```

```
638367)*mean concavity +(-28.89083490172692)*mean concave points +(8.901611425176887)*mean symmetry +(21.487057
20808176)*mean fractal dimension +(-16.73535483059294)*radius error +(7.437477599025896)*texture error +(-9.663
691395411341)*perimeter error +(-20.126503059831077)*area error +(-3.216160100235993)*smoothness error +(-0.166
02327329112487)*compactness error +(7.404826899700334)*concavity error +(3.718460317010289)*concave points erro
r +(9.014655920673821)*symmetry error +(3.585310030168413)*fractal dimension error +(-2.634483814563519)*worst
radius +(-9.682323599867244)*worst texture +(1.2701837043219981)*worst perimeter +(-23.409847103325067)*worst a
rea +(-0.11542563576074742)*worst smoothness +(1.19253628743609)*worst compactness +(2.4923314655389976)*worst
concavity +(-0.4331139163786498)*worst concave points +(-14.549507629636972)*worst symmetry +(-9.02942076918891
5)*worst fractal dimension
```

In [74]:

```
# separate negative and positive coefficients and respective attributes into separate dictionaries

coeffs_pos = {}
coeffs_neg = {}
for i, col in enumerate(breast_cancer.feature_names):
    if weights_opt[i]>0:
        coeffs_pos[col] = weights_opt[i]
    else:
        coeffs_neg[col] = weights_opt[i]
print("Attributes with positive cofficients:", sorted(coeffs_pos.items(), key=lambda x: x[1],reverse=True))
print("Attributes with negative cofficients:", sorted(coeffs_neg.items(), key=lambda x: x[1],reverse=True))
```

```
Positive cofficient features : [('mean radius', 21.839383705191462), ('mean fractal dimension', 21.487057208081
76), ('mean perimeter', 17.791312463824244), ('symmetry error', 9.014655920673821), ('mean symmetry', 8.9016114
25176887), ('texture error', 7.437477599025896), ('concavity error', 7.404826899700334), ('mean smoothness', 6.
814212888124369), ('concave points error', 3.718460317010289), ('fractal dimension error', 3.585310030168413),
('worst concavity', 2.4923314655389976), ('mean texture', 1.596791990260688), ('worst perimeter', 1.27018370432
19981), ('worst compactness', 1.19253628743609)]
Negative cofficient features : [('worst smoothness', -0.11542563576074742), ('compactness error', -0.1660232732
9112487), ('worst concave points', -0.4331139163786498), ('worst radius', -2.634483814563519), ('smoothness err
or', -3.216160100235993), ('mean compactness', -3.5122890164092), ('worst fractal dimension', -9.02942076918891
5), ('perimeter error', -9.663691395411341), ('worst texture', -9.682323599867244), ('mean area', -9.8427961227
41028), ('worst symmetry', -14.549507629636972), ('mean concavity', -16.36068280638367), ('radius error', -16.7
3535483059294), ('area error', -20.126503059831077), ('worst area', -23.409847103325067), ('mean concave point
s', -28.89083490172692)]
```

References: https://towardsdatascience.com/logistic-regression-from-scratch-69db4f587e17

Question 3:

Implement the three following cross-validation algorithms from scratch:

a. Leave-one-out cross-validation

b. K-fold cross-validation

c. Train-test split cross-validation

Test your results on the California Housing Dataset:

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html#sklearn.datasets.fetch_california_housing

Now, do the following:

```
a. Implement the cross-validation algorithms from scratch.

b. Choose the following features from the dataset as your X matrix: MedInc,
HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude

c. Choose the following feature from the dataset as your Y matrix: MedHouseVal

d. Apply 0 – 1 normalization on X and Y.

e. Apply the cross-validation algorithms that you implemented to train your model.

(For splitting your data always use 265 as your random number or seed value).
```

Note: You cannot use the pre-packaged algorithms for splitting the data.

To split the data please do the following:

```
    a. Install the random package written for Python

    b. Set (the initial) random.seed() to 265

    c. Create a list of integers that will function as your index numbers:
    list(range(0,len(name_of_dataset))

    d. Pick one integer for Train-Test Split CV from the list your created in c) to
    split 70% of your data as training set and the remaining 30% as the test set.
```

e. For the K-fold CV, set k = 5. Please divide the dataset into 5 quasi-equal portions starting from index 0.

f. For LOOCV, start the training by randomly picking a feature vector associated with an index in your dataset (Reminder: random seed is 265) – you will need to run the model on every point.

f. Using scikit's sklearn.linear_model.LinearRegression, predict the house prices by using all of the data in your X matrix. Compare different techniques of CV. Which CV provides the lowest MSE? Why? Interpret the results.

In [93]:

```
# mean squared error calculation
def MSE(y, y_pred):
    return np.square(np.subtract(y, y_pred)).mean()

# leave one out cross validation
def leave_one_out_CV(X, y, model_name):
    n = len(X)
    errors = []
    models_dict = {}

    for i in range(n):
        X_train, X_test = X.drop(i,axis=0), X.iloc[i,:].values
        y_train, y_test = np.delete(y,i), y[i]
        model = model_name
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test.reshape(1,-1))
        MSEscore = MSE(y_test, y_pred)
        if idx%1000==0:
            print(f"MSE score for model- {i} is ", MSEscore)
            errors.append(MSEscore)
            models_dict[i] = model

    print("Average mean of MSE is", np.mean(errors))
    return models_dict, errors

# k fold cross validation basic code
def k_fold_CV(dataset, k):
    datasplit = {}
    num_ele_per_fold = int(dataset.shape[0]/k)
    for i in range(k):
        fold=[]
        while (len(fold) < num_ele_per_fold):
```

```
            r = randrange(dataset.shape[0])
            ind = dataset.index[r]
            fold.append(ind)
            dataset = dataset.drop(ind)
        datasplit[i]=np.asarray(fold)
    return datasplit

# implementation of k fold cross validation
def k_fold_CV_imp(X, y, k, model_name):
        errors = []
        models_dict = {}
        fold_ids = k_fold_CV(X, k)
        for ind in range(k):
            print(f"{ind} fold")
            X_train, X_test = X.drop(fold_ids[ind],axis=0), X.iloc[fold_ids[ind],:].values
            y_train, y_test = np.delete(y, fold_ids[ind]),y[fold_ids[ind]]
            model = model_name
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            MSEscore = MSE(y_test, y_pred)
            print(f"MSE score for model {ind} is ", MSEscore)
            print("----------------------------------------------------------------------")
            errors.append(MSEscore)
            models_dict[ind] = model
        print("----------------------------------------------------------------------")
        print("Average mean of the MSE error is", np.mean(errors))
        return models_dict, errors

# train test split
def train_test_split(X, y, model_name):
            df = pd.concat([X, pd.DataFrame(y)], axis=1)
            df = df.sample(frac=1, random_state=265)
            indices = range(X.shape[0])
            train_size = int(0.7 * X.shape[0])
            train_indices = indices[:train_size]
            test_indices = indices[train_size:]
            X_train, X_test = df.iloc[train_indices],df.iloc[test_indices]
            y_train, y_test = df.iloc[train_indices,-1],df.iloc[test_indices,-1]
            model = model_name
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            MSEscore = MSE(y_test, y_pred)
            print(f"MSE score for model is ", MSEscore)
            return model
```

In [94]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
import random
from random import randrange

np.random.seed(265)
random.seed(265)

# importing the data
df_cal_housing = fetch_california_housing()

X = df_cal_housing.data
y = df_cal_housing.target

# scaling the data
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
X = pd.DataFrame(X, columns = df_cal_housing.feature_names)

# performing k fold cross validation on the data
models, errors = k_fold_CV_imp(X, y, 5, LinearRegression())
```

```
0 fold
MSE score for model 0 is  0.529974940061078
----------------------------------------------------------------------
1 fold
MSE score for model 1 is  0.5477645459252342
----------------------------------------------------------------------
2 fold
MSE score for model 2 is  0.5040908861674307
----------------------------------------------------------------------
3 fold
MSE score for model 3 is  0.5278838122736698
----------------------------------------------------------------------
4 fold
MSE score for model 4 is  0.5391523948478724
----------------------------------------------------------------------
----------------------------------------------------------------------
Average mean of the MSE error is 0.529773315855057
```

In [95]:
```
# performing train test split on the data
model = train_test_split(X, y, LinearRegression())
```

```
MSE score for model is  9.032338394368907e-31
```

In [ ]: