TASK 1. REGRESSION

HOUSEPRICE_DATA_LINEAR.PY

Model Implementation:

We trained our data with 15% test size of our data. We found the highest correlating features with price and used these features from top and gradually worked our way down to less correlating features followed by a mix of two, three and four features to attain optimal accuracy score whist testing against our target feature(price).

Model visualization:

Model was not visualized; our best model accuracy was using the top four (4) correlating features which would not fit into our Linear or Multiple regression.

```
[ ] X= df[['sqft_living','grade','sqft_above','sqft_living15']] # Using the 4 highest correlating features gives the highest accuracy so far
    Y= df['price']
    regr.fit(X,Y)
    regr.score(X,Y)

    0.5419882715173809
```

Model Improvement:

- Using two (2) input features increased our model accuracy as opposed to one (1)
- Using three (3) input features also increased our model accuracy as opposed to two (2)
- For every additional input feature, accuracy increased significantly.
- We saw a stagnancy in accuracy score when input feature was beyond four (4)

```
[ ] X= df[['bathrooms']]
    Y= df['price']
    regr.fit(X,Y)
    regr.score(X,Y)

    0.2757657943583819

[ ] X= df[['bedrooms']]
    Y= df['price']
    regr.fit(X,Y)
    regr.score(X,Y)

    0.09507254960523659

[ ] X= df[['view']]
    Y= df['price']
    regr.fit(X,Y)
    regr.score(X,Y)

    0.15788422070137265

[ ] X= df[['bathrooms', 'bedrooms', 'view']] # Using more than one feature improves the the model
    Y= df['price'] # There is an improvement in score using feature from the 3 previous cells
    regr.fit(X,Y)
    regr.score(X,Y)

    0.3707802133177748
```

<u>Model effectiveness:</u>

Changing from linear to multiple regression and using all our input features, we attained an accuracy score of **69 percent**

```
[ ] regr.score(x_test[['bedrooms', 'bathrooms', 'sqft_living','sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement','yr_built', 'yr_renovated',
            'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15']], y_test) # Our model has a 69 percent accuracy, Can also be interpreted as coefficient of determination

0.690599829594227
```

Afrer the R2_scores, we should note the following:

- Combining two or more features increases our R2 score
- The sqft_living is the single highest determinant of the price of house with a higher R2 score
- Using the 4 highest detrerminants(grade,sqft(living,above,living15) our R2 score reached a significantly new height of 0.54
- From the previous cell, our model is 69 percent accurate

<u>Model prediction, recommendation and deduction:</u>

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
17384	2	1.50	1430	1650	3.0	0	0	3	7	1430	0	1999	0	98125	47.7222	-122.290	1430	1650
722	4	3.25	4670	51836	2.0	0	0	4	12	4670	0	1988	0	98005	47.6350	-122.164	4230	41075
2680	2	0.75	1440	3700	1.0	0	0	3	7	1200	240	1914	0	98107	47.6707	-122.364	1440	4300
18754	2	1.00	1130	2640	1.0	0	0	4	8	1130	0	1927	0	98109	47.6438	-122.357	1680	3200
14554	4	2.50	3180	9603	2.0	0	2	3	9	3180	0	2002	0	98155	47.7717	-122.277	2440	15261

```
y_test[:5]

17384     297000.0
722      1580000.0
2680      562100.0
18754     631500.0
14554     780000.0
Name: price, dtype: float64

[ ] regr.predict(x_test[:5]) # Linear model prediction of houseprices of the above cell.(y_test)

array([ 377606.29577026, 1539911.31784929,  545707.24632067,
        579000.96724091,  977780.15236804])
```

When we inputted *regr.predict(x_test[:5]), t*he above image tells us the linear regression prediction of houses no **17384, 722, 2680, 18754 and 14554. T**he feature values of the houses can be found in the topmost cell output in our image.

House no.	**Actual price. (ap)**	**Model price prediction. (mp)**	**Difference. (mp –** ap**)**
17384	297000	377606	80606
722	1580000	1539911	-40089
2680	562100	545707	-16393
18754	631500	579000	-52500
14554	780000	977780	197780

From the table above, considering all input features in our model. Multiple regression gives a future prediction of our house prices.

TASK 2. CLUSTERING

COUNTRY_DATA_CLUSTERING.PY

Model implementation:

The general idea was to find clusters between opposing features or features whose values directly affect the other, positive or negative. We paired these clusters and found a center using the Kmeans method.

```
scaler = MinMaxScaler() # Transforms numeric feature data to range 0 to 1.

cds = scaler.fit_transform(cdd.to_numpy())
cds = pd.DataFrame(cds, columns=[
    'child_mort', 'exports',  'health', 'imports',  'income', 'inflation',  'life_expec', 'total_fer',  'gdpp'])

print("Scaled Dataset Using MinMaxScaler")
cds.head()
```

Scaled Dataset Using MinMaxScaler

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	0.426485	0.049482	0.358608	0.257765	0.008047	0.126144	0.475345	0.736593	0.003073
1	0.068160	0.139531	0.294593	0.279037	0.074933	0.080399	0.871795	0.078864	0.036833
2	0.120253	0.191559	0.146675	0.180149	0.098809	0.187691	0.875740	0.274448	0.040365
3	0.566699	0.311125	0.064636	0.246266	0.042535	0.245911	0.552268	0.790221	0.031488
4	0.037488	0.227079	0.262275	0.338255	0.148652	0.052213	0.881657	0.154574	0.114242

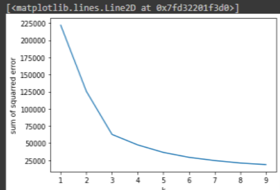
From the above image, using Minmax scaler, our values were scaled into range (0-1) to help was align our clusters.

Model improvement:

Before generating clusters, we found the optimal number of clusters needed for our features called the elbow method. This gives us insights on the number of clusters needed.

```
[ ] plt.xlabel('k') # Elbow method to find the perfect number of clusterm = 3
    plt.ylabel('sum of squarred error')
    plt.plot(k_rng,sse)

    [<matplotlib.lines.Line2D at 0x7fd32201f3d0>]
```




```
[ ] km = KMeans(n_clusters=3)
    km
```

Model visualization:

Using two features, import and export. We visualized our model with clusters and finding their centers which basically means our reference point.

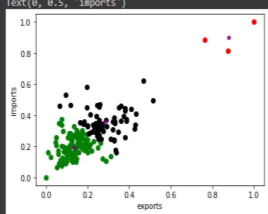
We repeated this process for our features in pairs (I.e. life_expect and health.)

```
df1 = cds[cds.cluster1==0] # Visualization of clusters and centroids
df2 = cds[cds.cluster1==1]
df3 = cds[cds.cluster1==2]

plt.scatter(df1.exports, df1['imports'], color= 'green')
plt.scatter(df2.exports, df2['imports'], color= 'red')
plt.scatter(df3.exports, df3['imports'], color= 'black')
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color ='purple', marker= '*', label= 'centroid')

plt.xlabel ('exports')
plt.ylabel('imports')
```

```
Text(0, 0.5, 'imports')
```

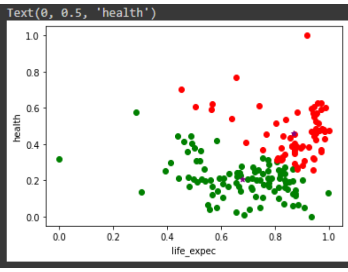


From the above image we can see the clusters of our imports and exports. There is a high import, and we can deduct the4 following

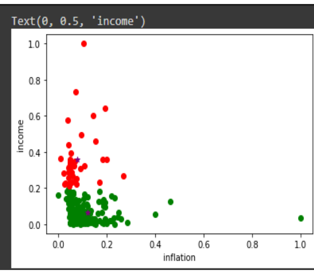
- Import values in the range of 0.8 -1.0 and export clusters in the range of 0.8-1.0 have lesser attributes hence there are not many imports and exports in that range.
- Most imports and exports happened at values ranging from 0.0-0.6

<u>Model prediction, recommendation and deduction:</u>

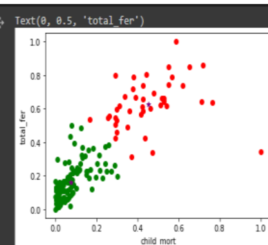
```
Text(0, 0.5, 'health')
```




```
Text(0, 0.5, 'income')
```




```
Text(0, 0.5, 'total_fer')
```



Health and Life_expec	**Income and inflation**	**Total_fer and child_mort**
When the health is in range 0.3-1.0, life expectancy falls between 0.5 to 1.0	When income was 1.0, inflation was 0.1 and vice versa	0.0 to 1.0 on the total_fert and child_mort axis saw a forward progressive movement in cluster
INTERPRETATION	**INTERPRETATION**	**INTERPRETATION**
This means that a higher health results in long life	High income earners have more purchasing power while low-income earners have lesser purchasing power. There is a high disparity between the rich and the poor.	With an increase in fertility comes a corresponding child mortality rate.

TASK 3. CLASSIFICATION

NBA_ROOKIE_DATA_LOGISTICS.PY

Model implementation:

Logistics regression was applied to our dataset, feeding our testing data with 1/3 of our training data. Predictions were then made based off this to determine the length of the career of an NBA player. To get an in-depth classification, we used all our features, then scaled using MinMax scaler (0-1) for better accuracy.

```
[ ] scaler = MinMaxScaler() # Transforms numeric feature data to range 0 to 1.

    nbs = scaler.fit_transform(nb.to_numpy())
    nbs = pd.DataFrame(nbs, columns=[
        'Games Played', 'Minutes Played', 'Points Per Game',  'Field Goals Made', 'Field Goal Attempts',  'Field Goal Percent', '3 Point Made', '3 Point Attempt' ,'3 Point Percent',
        'Free Throw Made',  'Free Throw Attempts',  'Free Throw Percent', 'Offensive Rebounds', 'Defensive Rebounds', 'Rebounds', 'Assists',  'Steals', 'Blocks', 'Turnovers',  'TARGET_5Yrs'])

    print("Scaled Dataset Using MinMaxScaler")
    nbs.head()
```

Scaled Dataset Using MinMaxScaler

	Games Played	Minutes Played	Points Per Game	Field Goals Made	Field Goal Attempts	Field Goal Percent	3 Point Made	3 Point Attempt	3 Point Percent	Free Throw Made	Free Throw Attempts	Free Throw Percent	Offensive Rebounds	Defensive Rebounds	Rebounds	Assists	Steals	Blocks	Turnovers	TARGET_5Yrs
0	0.352113	0.642857	0.243636	0.232323	0.357895	0.218437	0.217391	0.323077	0.250	0.207792	0.225490	0.699	0.132075	0.340426	0.279412	0.179245	0.16	0.102564	0.279070	0.0
1	0.338028	0.629630	0.236364	0.171717	0.310526	0.116232	0.304348	0.430769	0.235	0.337662	0.333333	0.765	0.094340	0.191489	0.154412	0.349057	0.44	0.128205	0.348837	0.0
2	0.887324	0.322751	0.163636	0.171717	0.205263	0.368737	0.173913	0.261538	0.244	0.116883	0.127451	0.670	0.094340	0.159574	0.139706	0.094340	0.20	0.076923	0.209302	0.0
3	0.661972	0.224868	0.181818	0.202020	0.247368	0.376754	0.043478	0.076923	0.226	0.116883	0.127451	0.689	0.188679	0.074468	0.117647	0.075472	0.24	0.025641	0.209302	1.0
4	0.521127	0.222222	0.138182	0.131313	0.115789	0.573146	0.000000	0.015385	0.000	0.168831	0.186275	0.674	0.188679	0.138298	0.161765	0.028302	0.12	0.102564	0.162791	1.0

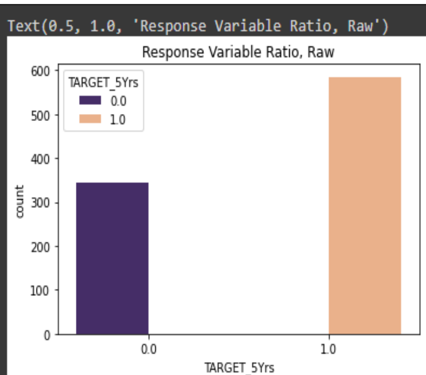
```
x= nbs.drop('TARGET_5Yrs', axis=1) # Dependent variables.
y= nbs['TARGET_5Yrs']  # Indepenent variable.
```

```
[ ] x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_state=0) # Train, test and split data.
```

Model visualization:

After scaling we called our target variable and its value to identify the distribution in terms of numbers

```
Text(0.5, 1.0, 'Response Variable Ratio, Raw')
```



From the above image, we can deduct the number of NBA players with career span of less than 5, denoted by 0.0 and 5 or more years denoted by 1.0

Model prediction, recommendation and deduction:

```
y_test[:5]

1287    1.0
445     1.0
458     1.0
251     1.0
1251    0.0
Name: TARGET_5Yrs, dtype: float64

[ ] logre.predict(x_test[:5]) # Our model indicates player (1287,445,458,251,1251) will all last at least 5 years in the NBA which are all correct predictions correlating with
    # y_test in the above cell

array([1., 1., 1., 1., 0.])
```

From the above image. We see how well our model's prediction performs.

Player index number	Y_test(target feature)	X_test(model predictor)	Model Conclusion
1287	1.0	1	5 or more years
445	1.0	1	5 or more years
458	1.0	1	5 or more years
251	1.0	1	5 or more years
1251	0.0	0	Less than 5 years

Our model correctly predicted or classified all our target variables and at **71 percent accuracy.**

```
[ ] print(classification_report(y_test,predictions))

              precision    recall  f1-score   support

         0.0       0.69      0.48      0.57       158
         1.0       0.72      0.86      0.78       241

    accuracy                           0.71       399
   macro avg       0.70      0.67      0.67       399
weighted avg       0.71      0.71      0.70       399

[ ] accuracy_score(y_test,predictions) # Our logistic regression model is 71 pecent accurate

0.7092731829573935
```

The above images show how strong our model(predictor) is with our target variable.

- Less than 5 years has a 69 percent accuracy when predicting
- 5 or more years has a 72 percent accuracy when predicting
- An overall accuracy of 71 percent was achieved.

TASK 3. CLASSIFICATION

NBA_ROOKIE_DATA_GNB.PY

Model implementation:

Gaussian Naïve Bayes was applied tom our data, feeding our testing data with 1/3 of our training data. Predictions were then made based off this to determine the length of the career of an NBA player. To get an in-depth classification, we used all our features, then scaled using MinMax scaler (0-1) for better accuracy.

```
gnb=GaussianNB() # 66 percent accurate on a player lasting 5 years in the NBA.
gnb.fit(X_train,y_train)
print(gnb)

gnb.score(X_test,y_test)

GaussianNB()
0.6613995485327314

y_test[:5]

1287    1.0
445     1.0
458     1.0
251     1.0
1251    0.0
Name: TARGET_5Yrs, dtype: float64

gnb.predict(X_test[:5]) # Our GaussianNB got 2 predictions wrong(1287,458) from our y_test above. consider we have 66 percent accuracy.

array([0., 1., 0., 1., 0.])
```

From the above image, we saw **66 percent model accurac**y:

Player index number	Y_test(target feature)	X_test(model predictor)	Model Conclusion
1287	1.0	0	Less than 5 years
445	1.0	1	5 years or more
458	1.0	0	Less than 5 years
251	1.0	1	5 years or more
1251	0.0	0	Less than 5 years

- Our GNB model got two (2) NBA player career span wrong

Model Improvement (Bernouli Naïve bayes):

After attaining accuracy with our GNB model. We implemented the BNB model to increase accuracy and ultimately an improvement.

```
bernnb = BernoulliNB(binarize = 0.1) # Best  Naive Bayes probalistic model on a player lasting 5 years in the NBA.
bernnb.fit(X_train, y_train)
print(bernnb)

bernnb.score(X_test, y_test)

BernoulliNB(binarize=0.1)
0.6817155756207675

y_test[:5]

1287    1.0
445     1.0
458     1.0
251     1.0
1251    0.0
Name: TARGET_5Yrs, dtype: float64

bernnb.predict(X_test[:5]) # The BernoulliNB got 1 prediction wrong(1287) from our y_test. Consider we have 68 percent accuracy

array([0., 1., 1., 1., 0.])
```

From the image above we can find the following information at **68 percent accuracy;**

Player index number	Y_test(target feature)	X_test(model predictor)	Model Conclusion
1287	1.0	0	Less than 5 years
445	1.0	1	5 years or more
458	1.0	1	5 years or more
251	1.0	1	5 years or more
1251	0.0	0	Less than 5 years

- Our BNB model got two (1) NBA player career span wrong.

TASK 3. CLASSIFICATION

NBA_ROOKIE_DATA_NeuralN.PY

Model implementation:

MLP classifier of the Neural network was applied to our dataset, feeding our testing data with 1/3 of our training data. Predictions were then made based off this to determine the length of the career of an NBA player. To get an in-depth classification, we used all our features, then scaled using MinMax scaler (0-1) for better accuracy.

Scaled Dataset Using MinMaxScaler

	Games Played	Minutes Played	Points Per Game	Field Goals Made	Field Goal Attempts	Field Goal Percent	3 Point Made	3 Point Attempt	3 Point Percent	Free Throw Made	Free Throw Attempts	Free Throw Percent	Offensive Rebounds	Defensive Rebounds	Rebounds	Assists	Steals	Blocks	Turnovers	TARGET_5Yrs
0	0.352113	0.642857	0.243636	0.232323	0.357895	0.218437	0.217391	0.323077	0.250	0.207792	0.225490	0.699	0.132075	0.340426	0.279412	0.179245	0.16	0.102564	0.279070	0.0
1	0.338028	0.629630	0.236364	0.171717	0.310526	0.116232	0.304348	0.430769	0.235	0.337662	0.333333	0.765	0.094340	0.191489	0.154412	0.349057	0.44	0.128205	0.348837	0.0
2	0.887324	0.322751	0.163636	0.171717	0.205263	0.368737	0.173913	0.261538	0.244	0.116883	0.127451	0.670	0.094340	0.159574	0.139706	0.094340	0.20	0.076923	0.209302	0.0
3	0.661972	0.224868	0.181818	0.202020	0.247368	0.376754	0.043478	0.076923	0.226	0.116883	0.127451	0.689	0.188679	0.074468	0.117647	0.075472	0.24	0.025641	0.209302	1.0
4	0.521127	0.222222	0.138182	0.131313	0.115789	0.573146	0.000000	0.015385	0.000	0.168831	0.186275	0.674	0.188679	0.138298	0.161765	0.028302	0.12	0.102564	0.162791	1.0

```
[ ] x= nbss.iloc[:,0:19].values # Dependent variable
    y= nbss.iloc[:,19].values # Independent variable

[ ] x_train, x_test,y_train,y_test= train_test_split(x,y, test_size=1/3, random_state=0) # Train, test and split data.

mlp = MLPClassifier(hidden_layer_sizes=(40,10), activation= 'tanh', #Neural network
    random_state=0, max_iter=2000)

+ Code  + Text

[ ] mlp.fit(x_train,y_train)

MLPClassifier(activation='tanh', hidden_layer_sizes=(40, 10), max_iter=2000,
              random_state=0)

[ ] mlp.score(x_test,y_test) #The neural network model using Mlpclassifier is 72 percent accurate

0.7178329571106095
```

- From the above image we can see our Neural network accuracy score of **72 percent**, after we scaled our dataset using MinMax scaler, training, testing and splitting.

```
[ ] y_test[:5]

    array([1., 1., 1., 1., 0.])

[ ] mlp.predict(x_test[:5])  # Our model is correct, all 5 players will last at least 5 years in the NBA which are all correct predictions correlating with
    # y_test in the above cell

    array([1., 1., 1., 1., 0.])
```

Y_test(target feature)	X_test(model predictor)	Model Conclusion
1.0	1	5 years or more
1.0	1	5 years or more
1.0	1	5 years or more
1.0	1	5 years or more
0.0	0	Less than 5 years

Model prediction, recommendation and deduction:**(LOGISTICS, GNB, NEURAL NETWORKS).**

Logistics(accuracy)%	Gaussian Naïve Bayes (accuracy)%	Neural Network (accuracy)%	Bernoulli Naïve bayes 9(accuracy)%
71%	**66%**	**72%**	**68%**
• 5/5 correct predictions • In every 100 predictions, 29 predictions will be wrong	• 3/5 correct predictions • In every 100 predictions, 34 predictions will be wrong	• 5/5 correct predictions • In every 100 predictions, 28 predictions will be wrong • Neural network remains our best predictor	• 4/5 correct predictions • In every 100 predictions, 32 predictions will be wrong