Classification Model.

Dataset:https://catalog.data.gov/dataset/center-for-medicare-medicaid-services-cms-medicare-claims-data

The dataset selected for the Classification Model is the Medicare dataset. The dataset is about Cardiovascular Disease. The reason for choosing the dataset is because healthcare is one of the major sectors where prediction is required. From this dataset, we can predict the Break Out column where the disease is more likely to occur in the category of people. The dataset is also selected because it has categorical as well as numeric value, and for classification model we need the target variable as categorical variable. The Algorithm used to build the model is the Random Forest Algorithm as it had the highest training accuracy among the rest of the algorithms.

Random Forest Model

In [1]:
```
import pandas as pd #importing to read csv file
import seaborn as sns #Using for Graph
import matplotlib.pyplot as plt #Using for Graph
```

Reading the Data File

In [2]:
```
data=pd.read_csv("medicare.csv")
data.head()
```

Out[2]:

Area2	PriorityArea3	PriorityArea4	Category	Topic	...	Break_Out_Category	Break_Out	CategoryId	TopicId	IndicatorID	Data_Value_TypeID	Break(
None	None	None	Cardiovascular Diseases	Major Cardiovascular Disease	...	Age	75+	C1	T1	MD101	Crude	
None	None	None	Cardiovascular Diseases	Major Cardiovascular Disease	...	Overall	Overall	C1	T1	MD101	Crude	
None	None	None	Cardiovascular Diseases	Major Cardiovascular Disease	...	Race	Non-Hispanic White	C1	T1	MD101	Crude	
None	None	None	Cardiovascular Diseases	Major Cardiovascular Disease	...	Race	Non-Hispanic White	C1	T1	MD101	Crude	
None	None	None	Cardiovascular Diseases	Major Cardiovascular Disease	...	Gender	Male	C1	T1	MD101	Crude	

In [3]:
```
data.shape #Checking Number of rows and columns
```

Out[3]: (42640, 29)

The libraries and datasets were imported in the Jupyter Notebook. The dataset was not clean and pre-processed. So, I had to remove the multiple columns which were similar to columns already present and were holding the same value as the other columns (for example Category and Category ID, Topic and Topic ID). Also, some columns had None or the same values throughout the columns. Hence, they were also dropped.

Droping the columns which are not useful

In [4]:
```
data = data.drop(['LocationAbbr','DataSource','PriorityArea2','PriorityArea4',
                  'Category','Data_Value_Type','Data_Value_Alt','Data_Value_Footnote_Symbol',
                  'Data_Value_Footnote','CategoryId','TopicId','IndicatorID','Data_Value_TypeID',
                  'BreakOutCategoryId','BreakOutId','LocationID','GeoLocation'], axis=1)
data.head()
```

Out[4]:

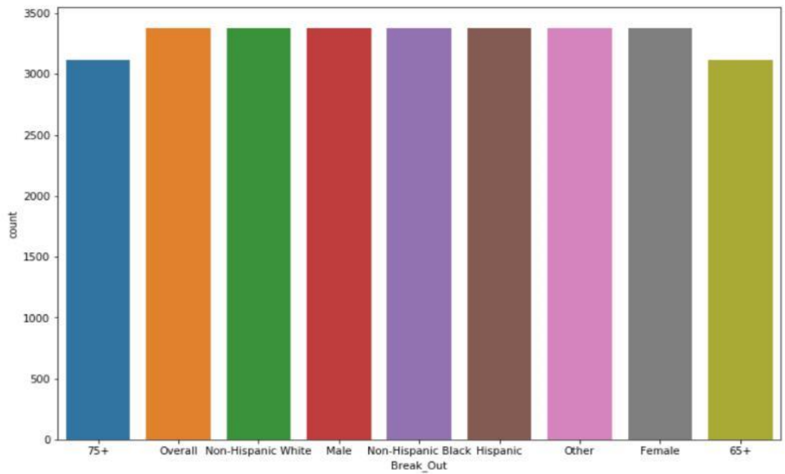
	Year	LocationDesc	PriorityArea1	PriorityArea3	Topic	Indicator	Data_Value_Unit	Data_Value	LowConfidenceLimit	HighConfidenceLimit	Breal
0	2008	New Mexico	None	None	Major Cardiovascular Disease	Prevalence of major cardiovascular disease hos...	Percent (%)	22.2	21.7	22.7	
1	2008	New York	None	None	Major Cardiovascular Disease	Prevalence of major cardiovascular disease hos...	Percent (%)	23.1	23.0	23.2	
2	2008	New York	None	None	Major Cardiovascular Disease	Prevalence of major cardiovascular disease hos...	Percent (%)	23.2	23.1	23.3	
3	2008	New Mexico	None	None	Major Cardiovascular Disease	Prevalence of major cardiovascular disease hos...	Percent (%)	22.4	21.9	22.8	
4	2008	New York	None	None	Major Cardiovascular Disease	Prevalence of major cardiovascular disease hos...	Percent (%)	25.9	25.7	26.0	

The Count plot of Target Variable was taken to see the difference between the different categories of the output variable. The ratio was 1:1.5, which is not suitable, so random sampling was done and the ratio was brought to 1:1 So that the predictions are not biased. After balancing the target variable, the shape of the data was 29900 rows and 12 columns. The NA values were also dropped.

Count of Output variable after Random Sampling

In [30]:
```
plt.figure(figsize=(12,8))
sns.countplot(x='Break_Out',data=balanced_data)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x195ee576048>



There were 2 columns (PriorityArea1 and PriorityArea3) which had multiple None values and when one of the columns had certain value the other was always None. So, I merged the two columns using the data frame. I could have also used PCA for dimension reduction, but I used a simple data frame code for it. After creating a new column (Priority Area), I dropped the old columns (PriorityArea1 and PriorityArea3).

As 7 columns had a categorical variable, I had to use label encoding on it, to change from categorical variable to Numeric one. After label encoding, I checked the correlation of the dataset as well as the input variable with the target variable. After the correlation, I didn't do feature scaling on the data, because for building a tree-based model and it does not always require feature scaling. Feature extraction was not done as it had only 10 columns.

Using Label Encoder to Change the Categorical variable to Numeric

```
In [38]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()

         data['Break Out']=le.fit_transform(data['Break_Out'])
         data['LocationDesc']=le.fit_transform(data['LocationDesc'])
         data['Topic']=le.fit_transform(data['Topic'])
         data['Indicator']=le.fit_transform(data['Indicator'])
         data['Data_Value_Unit']=le.fit_transform(data['Data_Value_Unit'])
         data['Break_Out_Category']=le.fit_transform(data['Break_Out_Category'])
         data['Priority Area']=le.fit_transform(data['Priority Area'])


         data.drop('Break_Out',axis=1,inplace=True)
         data.head()
```

Out[38]:

	Year	LocationDesc	Topic	Indicator	Data_Value_Unit	Data_Value	LowConfidenceLimit	HighConfidenceLimit	Break_Out_Category	Priority Area	Break Out
0	2008	30	4	5	0	22.2	21.7	22.7	0	1	1
1	2008	31	4	5	0	23.2	23.1	23.4	0	1	1
2	2012	22	2	0	0	18.4	18.2	18.7	0	1	1
3	2004	25	3	4	0	5.0	4.7	5.3	0	1	1
4	2004	34	3	4	0	7.6	7.5	7.7	0	1	1

After this process, the data was divided into features and targets with a break out column taken as a target variable. Another set of libraries was used like train test split, KFold, Cross Val score, Random Forest Classifier and accuracy score to fit and evaluate the model. The dataset was divided into training (70%) and testing (30%).

KFold cross-validation was used because it was easy to understand and easy to implement. It is a technique to evaluate a model on some random samples. With splits randomly kept at 10. The n estimator in the Random Forest model is the number of trees to be used. Since Random Forest is built on by multiple Decision Trees, hence parameter is set on how many trees are required. The Accuracy of the training score was around 75 percent.

Training Accuracy

```
In [45]: Result_model=cross_val_score(model,X_train,Y_train,cv=Kfoldobj)
         print(Result_model)

         [0.76353965 0.75145068 0.74613153 0.75338491 0.76547389 0.74854932
          0.73404255 0.75676983 0.75967118 0.76740812]

In [46]: print(Result_model.mean())

         0.754642166344294

In [47]: model.fit(X_train,Y_train)

Out[47]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=200,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

Storing the Prediction

```
In [48]: Prediction=model.predict(X_test)
```

Testing Accuracy

```
In [49]: testing_accuracy=accuracy_score(Y_test,Prediction)
         print(testing_accuracy)

         0.7589980819135733
```

The result of predictions was saved and testing accuracy was calculated to be almost 76 percent. The Actual and Predicted values were compared. For evaluating, further Confusion Matrix, Precision, Recall, and F1 score were also calculated.

Confusion Matrix

```
In [51]: from sklearn.metrics import confusion_matrix
         confusion_matrix(Y_test,Prediction)

Out[51]: array([[ 679,  239,    0,    0,    0,    0,    0,    0,    0],
                [ 247,  729,    0,    0,    0,    0,    0,    0,    0],
                [   0,    0,  825,    0,  167,    0,    0,    0,    0],
                [   0,    0,    0,  550,    0,  102,   41,  223,    0],
                [   0,    0,  222,    0,  817,    0,    0,    0,    0],
                [   0,    0,    0,  202,    0,  600,   67,  133,    0],
                [   0,    0,    0,    9,    0,   81,  872,   49,    0],
                [   0,    0,    0,  193,    0,   75,   86,  624,    0],
                [   0,    0,    0,    0,    0,    0,    0,    0, 1031]],
               dtype=int64)
```

Evaluating the model

```
In [52]: from sklearn.metrics import classification_report
         report=classification_report(Y_test,Prediction)
         print(report)

              precision    recall  f1-score   support

           0       0.73      0.74      0.74       918
           1       0.75      0.75      0.75       976
           2       0.79      0.83      0.81       992
           3       0.58      0.60      0.59       916
           4       0.83      0.79      0.81      1039
           5       0.70      0.60      0.65      1002
           6       0.82      0.86      0.84      1011
           7       0.61      0.64      0.62       978
           8       1.00      1.00      1.00      1031

    accuracy                           0.76      8863
   macro avg       0.76      0.76      0.76      8863
weighted avg       0.76      0.76      0.76      8863
```

The other classification models which were used are Decision tree, Naïve Bayes and KNN. The training accuracy of the Decision tree was good, but Random forest is an ensemble model, I wanted to give it a try. I learned how to build a Model from scratch. The dataset was not clean and required pre-processing. I completed all the steps required to build a classification model.

Decision Tree

```
In [29]: from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
         from sklearn.model_selection import train_test_split # Import train_test_split function
         from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

In [30]: Features_train, Features_test, Target_train, Target_test = train_test_split(Features, Target, test_size=0.3, random_state=1) # 7

In [31]:
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score

In [73]: Kfoldobj=KFold(n_splits=10)
         model = DecisionTreeClassifier()

In [74]: Result_model=cross_val_score(model,Features_train,Target_train,cv=Kfoldobj)
         Result_model

Out[74]: array([0.77747626, 0.78561737, 0.77815468, 0.77510176, 0.79647218,
                0.79613297, 0.77815468, 0.76756023, 0.78147268, 0.78724126])

In [75]: Result_model.mean()

Out[75]: 0.7823384082149637
```

Naive Bayes

```
In [87]: from sklearn.naive_bayes import GaussianNB
```

```
In [88]: Kfoldobj=KFold(n_splits=10)
         model = GaussianNB()
```

```
In [89]: Result_model=cross_val_score(model,Features_train,Target_train,cv=Kfoldobj)
         Result_model
```

```
Out[89]: array([0.44131615, 0.44369064, 0.44572592, 0.44504749, 0.44369064,
                0.43622795, 0.43249661, 0.45300305, 0.43841194, 0.43366135])
```

KNN

```
In [90]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [91]: Kfoldobj=KFold(n_splits=10)
         model = KNeighborsClassifier()
```

```
In [92]: Result_model=cross_val_score(model,Features_train,Target_train,cv=Kfoldobj)
         Result_model
```

```
Out[92]: array([0.66350068, 0.6587517 , 0.66214383, 0.65434193, 0.66994573,
                0.67503392, 0.66451832, 0.67017306, 0.66372582, 0.66270784])
```

Regression Model

Dataset:https://catalog.data.gov/dataset/regression-data-for-inclusionary-housing-simulation-model

The dataset selected for the Regression model is the Housing dataset. The dataset for the regression model requires continuous values. The Housing dataset has multiple columns and more than 1 million records. The dataset was selected because it had multiple columns that had continuous variables. The algorithm which was used is Multiple Linear Regression.

Reading the Data File

```
In [54]: data=pd.read_csv('Housing.csv')
         data.head()
```

Out[54]:

..	IntRate_10yr	IntRate_Mortg	LandUse_AsGiven	Zoning_AsGiven	ParkingMinReq	Bldg_SqFt_Use	Env_1000_Dens	SHistoric1	DevPotential_Dens	Area
..	5.09	7.07	PAPER LOT	P	NaN	21248.86502	42.49773	0	2.0	28331.82002
..	4.03	6.05	PDR	P	NaN	21248.86502	42.49773	0	2.0	28331.82002
..	4.27	5.88	PDR	P	NaN	21248.86502	42.49773	0	2.0	28331.82002
..	4.23	5.75	PDR	P	NaN	21248.86502	42.49773	0	2.0	28331.82002
..	4.47	6.27	PAPERLOT	P	NaN	21248.86502	42.49773	0	2.0	28331.82002

```
In [55]: data.shape
```

```
Out[55]: (1048575, 34)
```

As the Dataset is Huge, taking the 1st 50000 Entries

I also tried Random Sampling on the dataset, but the difference between training and testing score was too large, also after feature extraction, I couldn't get the score up. So I took the first 50000 Entries. I have also added the screenshot of random sample code score in report.

```
In [56]: new_data=data[:50000]
```

The important libraries and datasets were loaded in the Jupyter Notebook. As the dataset had 1 million records, I took the first 50K records. I also tried the random sample on the data, but the score difference between training and testing was too high. I also tried feature extraction on the random sample, but still, it didn't work well. So, I selected the first 50K records to go ahead. The dummy columns which were present in the excel sheet were directly removed from the excel sheet. After removing the dummy and unwanted columns, 34 columns were left.

Changing the Categorical Variable to Numeric

```
In [61]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
         new_data['LandUse_AsGiven']=le.fit_transform(new_data['LandUse_AsGiven'])
         new_data['Zoning_AsGiven']=le.fit_transform(new_data['Zoning_AsGiven'])
         new_data['MapBlkLot_Master']=le.fit_transform(new_data['MapBlkLot_Master'])
         new_data['MapBlkLot_Year']=le.fit_transform(new_data['MapBlkLot_Year'])
         new_data
```

Out[61]:

	year	Dev_Year	MapBlkLot_Master	MapBlkLot_Year	Res_Dummy	Price_OEA	Inc_CostSqFt	Inc_UnitFee1000	Price_Zillow	RentComm_YrEnd	...	IntRa
15	2001	0	1973	2068	0	50.561216	0.000000	0.00000	49.021000	93.55560	...	
16	2002	0	1973	2068	0	53.644685	28.046393	29.23575	52.208197	97.52959	...	
17	2003	0	1973	2068	0	54.746212	26.674754	29.23575	55.933722	85.59270	...	
18	2004	0	1973	2068	0	60.970784	25.900682	29.23575	64.821391	105.09890	...	
19	2005	0	1973	2068	0	70.412462	32.254786	29.23575	74.219141	119.20560	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
49995	2001	0	1966	2061	1	50.561216	0.000000	0.00000	49.021000	93.55560	...	
49996	2002	0	1966	2061	1	53.644685	28.046393	29.23575	52.208197	97.52959	...	
49997	2003	0	1966	2061	1	54.746212	26.674754	29.23575	55.933722	85.59270	...	
49998	2004	0	1966	2061	1	60.970784	25.900682	29.23575	64.821391	105.09890	...	
49999	2005	0	1966	2061	1	70.412462	32.254786	29.23575	74.219141	119.20560	...	

49536 rows × 34 columns

The NA values from the dataset were removed and the head and shape of the dataset were checked. There was 4 categoric variable in the dataset, they were changed to numeric with the help of label encoder. Correlation of the data was checked with each other as well as the target variable.

At first, I thought of giving a try without feature extraction as I had around 50K records and 34 columns. The data was divided into Features and Targets. And the feature scaling was done to get the mean as 0 and variance as 1.

Spliting the data into Feature and Target Variable

```
In [65]: Features=new_data.drop("Area",axis=1)
         Target=new_data["Area"]
```

Feature Scaling (To transform the data)

```
In [66]: from sklearn.preprocessing import StandardScaler

         feature_scaler = StandardScaler()
         X_scaled = feature_scaler.fit_transform(Features)
```

Spliting the data in training and testing.

```
In [67]: from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split( X_scaled, Target, test_size = 0.3, random_state = 100)
```

Fitting the Model

```
In [68]: from sklearn.linear_model import LinearRegression
         model = LinearRegression()
         model.fit(X_train, Y_train)

Out[68]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

After feature scaling, the data were divided into training (70%) and testing (30%) set. The Linear Regression model was used to fit the model and the predicted values where stored in a new variable. As the Multiple Linear Regression follows the formula (see image below), The coefficient and intercept were calculated.

The Formula for Multiple Linear Regression Is

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon$$

where, for $i = n$ observations:

$y_i =$ dependent variable

$x_i =$ expanatory variables

$\beta_0 =$ y-intercept (constant term)

$\beta_p =$ slope coefficients for each explanatory variable

$\epsilon =$ the model's error term (also known as the residuals)

The training and testing score were calculated and they came out as 82% and 80% respectively. Also, another evaluation score like Mean Absolute Error, Mean Squared Error, Root Mean Squared Error, and Variance score were calculated.

Training and Testing Score

```
In [72]: print('Train Score :', model.score(X_train,Y_train))
         print('Test Score:', model.score(X_test,Y_test))

Train Score : 0.8255039289336635
Test Score: 0.8026019980467649
```

```
In [73]: from sklearn import metrics
         from sklearn.metrics import r2_score

         print('MAE :',metrics.mean_absolute_error(Y_test, y_pred))

         print('MSE :', metrics.mean_squared_error(Y_test,y_pred))

         print('RMSE :', np.sqrt(metrics.mean_squared_error(Y_test,y_pred)))

         print('Variance score: %.2f' % r2_score(Y_test,y_pred))

MAE : 1223.8721616224163
MSE : 14765321.152345994
RMSE : 3842.567000371756
Variance score: 0.80
```

Just to compare, I also tried to use the feature extraction technique to check if the model underfits or not. For doing the same, I used the Recursive Feature Elimination technique. The 16 best features where selected. The rank was generated based on the RFE algorithm. The unwanted columns were dropped and the same process from above was followed. Like splitting the data into features and targets, doing the feature scaling, Splitting the data into train and test set. The model was fitted and the prediction was stored in the variable.

Recursive Features Elimination for Feature Extraction

```
In [76]: from sklearn.feature_selection import RFE
         from sklearn.linear_model import LinearRegression

         array = new_data.values
         X = array[:,0:33]
         Y = array[:,33]

         model=LinearRegression()
         rfe=RFE(model,16)
         k=rfe.fit(X,Y)
```

```
In [77]: print("Number of features"+" "+ str(k.n_features_))

Number of features 16
```

```
In [78]: print("Number of features"+" "+ str(k.support_))

Number of features [ True  True False False  True  True  True False  True  True False False
 False False False  True False  True False False False False False False
 False  True  True  True  True False  True  True  True]
```

```
In [79]: print("Number of features"+" "+ str(k.ranking_))

Number of features [ 1  1 12 18  1  1  1 10  1  1  5  7  6 17 16  1 14  1 15  3  9  8  2 11
  4  1  1  1  1 13  1  1  1]
```

The Actual and predicted value table was created, scores of testing and training were 75% and 79% respectively. After doing the feature extraction, the evaluation scores of Mean Absolute Error, Mean Squared Error, Root Mean Squared Error were increased, but the variance decreased from 80 to 75 percent.

Training and Testing Score

```
In [89]: print('Train Score :', model.score(X_train,Y_train))
         print('Test Score:', model.score(X_test,Y_test))

Train Score : 0.7901975358319431
Test Score: 0.7506136034985831
```

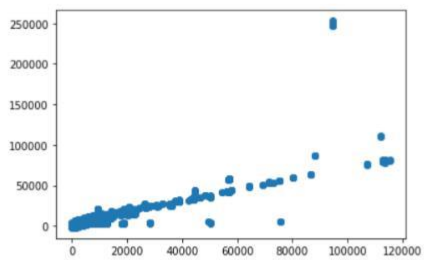
Evaluating the model

```
In [90]: from sklearn import metrics
         from sklearn.metrics import r2_score

         print('MAE :',metrics.mean_absolute_error(Y_test, y_pred))

         print('MSE :', metrics.mean_squared_error(Y_test,y_pred))

         print('RMSE :', np.sqrt(metrics.mean_squared_error(Y_test,y_pred)))

         print('Variance score: %.2f' % r2_score(Y_test,y_pred))

MAE : 1431.133044229348
MSE : 18654040.05579586
RMSE : 4319.032305481849
Variance score: 0.75
```

The scatter plot of Actual and Predicted value was created and it showed a somewhat linear line in the graph.

Plotting the graph

```
In [91]: plt.scatter(Y_test, y_pred)

Out[91]: <matplotlib.collections.PathCollection at 0x1959492e088>
```



Also, I tried to use Random Sample for 50K values, and I have added the snapshots of the code below. But the score was too less. I also tried it with feature extraction, but was not able to decrease the score between them.

```
In [4]: new_data = data.sample(frac=0.05,random_state=2)
        new_data.shape

Out[4]: (52429, 34)
```

Without Feature Extraction

```
In [59]: print('Train Score :', regressor.score(X_train,Y_train))
         print('Test Score:', regressor.score(X_test,Y_test))

Train Score : 0.3080660117951125
Test Score: 0.18877771004185395
```

```
In [60]:
         from sklearn import metrics
         from sklearn.metrics import r2_score

         print('MSE :', metrics.mean_squared_error(Y_test,y_pred))

         print('RMSE :', np.sqrt(metrics.mean_squared_error(Y_test,y_pred)))

         print('Variance score: %.2f' % r2_score(Y_test,y_pred))

MSE : 108848010.7587905
RMSE : 10433.025005183803
Variance score: 0.19
```

With Feature Extraction

```
In [75]: print('Train Score :', regressor.score(X_train,Y_train))
         print('Test Score:', regressor.score(X_test,Y_test))

Train Score : 0.30481761281672803
Test Score: 0.17029229785728217
```

```
In [76]:
         from sklearn import metrics
         from sklearn.metrics import r2_score

         print('MSE :', metrics.mean_squared_error(Y_test,y_pred))

         print('RMSE :', np.sqrt(metrics.mean_squared_error(Y_test,y_pred)))

         print('Variance score: %.2f' % r2_score(Y_test,y_pred))

MSE : 111328342.43760908
RMSE : 10551.2246889927
Variance score: 0.17
```

I learned how to build a model from scratch. The dataset was not clean and required pre-processing. I completed all the steps required to build a regression model. The task of selecting the dataset from the site was difficult, but somehow, I managed to find the dataset which can be used to work on and complete the model.