REPORT OF LUNG CANCER DETECTION MODEL EVALUATION

Project Topic

This project is about creating lung cancer detection using Supervised Machine learning Algorithms. This project outcome is to help the physicians to help and save time in identifying the nodules present in CT lung images in the early stage of lung cancer. Since that process is the time consuming and very critical

<u>Motive</u>

The effectiveness of the cancer prediction system helps people to know their cancer risk at a low cost and it also helps the people to take the appropriate decision based on their cancer risk status. The data is collected from the website online lung cancer prediction system.

Data

Data is a Public kaggle dataset which is called Lung Cancer.
source link:
https://www.kaggle.com/datasets/nancyalaswad90/lung-cancer/data

In-Text Citation: (Hong & Yang, 1991)

Reference List Entry:

Hong, Z. Q., & Yang, J. Y. (1991). Optimal Discriminant Plane for a Small Number of Samples and Design Method of Classifier on the Plane Lung Cancer Dataset. Kaggle.
https://www.kaggle.com/datasets/nancyalaswad90/lung-cancer/data

License: CC BY-NC-SA 4.0

Format: CSV

Size:

RangeIndex: 309 entries,
Data Types: int64(14), object(2)

```
data.info()
```

Data Cleaning

```
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 309 non-null    object
 1   AGE                    309 non-null    int64
 2   SMOKING                309 non-null    int64
 3   YELLOW_FINGERS         309 non-null    int64
 4   ANXIETY                309 non-null    int64
 5   PEER_PRESSURE          309 non-null    int64
 6   CHRONIC DISEASE        309 non-null    int64
 7   FATIGUE                309 non-null    int64
 8   ALLERGY                309 non-null    int64
 9   WHEEZING               309 non-null    int64
 10  ALCOHOL CONSUMING      309 non-null    int64
 11  COUGHING               309 non-null    int64
 12  SHORTNESS OF BREATH    309 non-null    int64
 13  SWALLOWING DIFFICULTY  309 non-null    int64
 14  CHEST PAIN             309 non-null    int64
 15  LUNG_CANCER            309 non-null    object
```

Data doesn’t contain any null values in all of the columns and only two columns are object types while all of them are 64 bits integer type.We compute this by using the info method on dataframe object.

Summary

Age Distribution: The dataset covers a range of ages from 21 to 87, with the majority (50%) failing between 57 and 69 years.

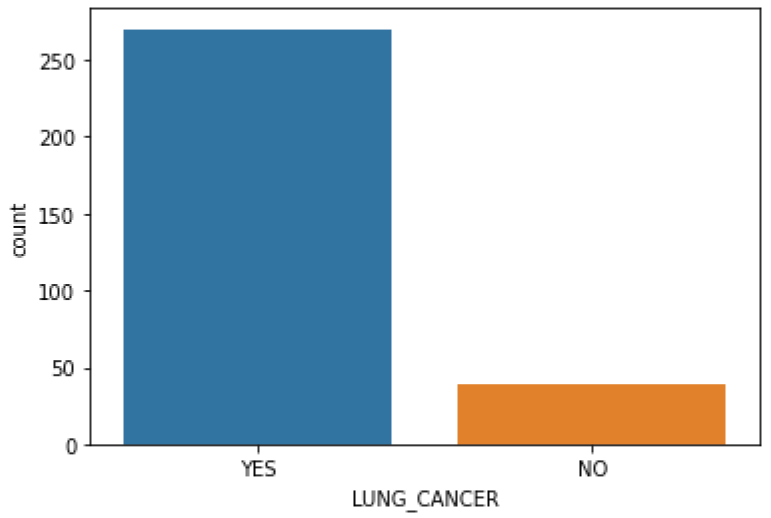
Categorical Variables: The statistics for other variables suggest that they might represent binary attributes.

For the detail of data distribution please check the output of

dataset.describe() row.

Analyzing Lung Cancer Column

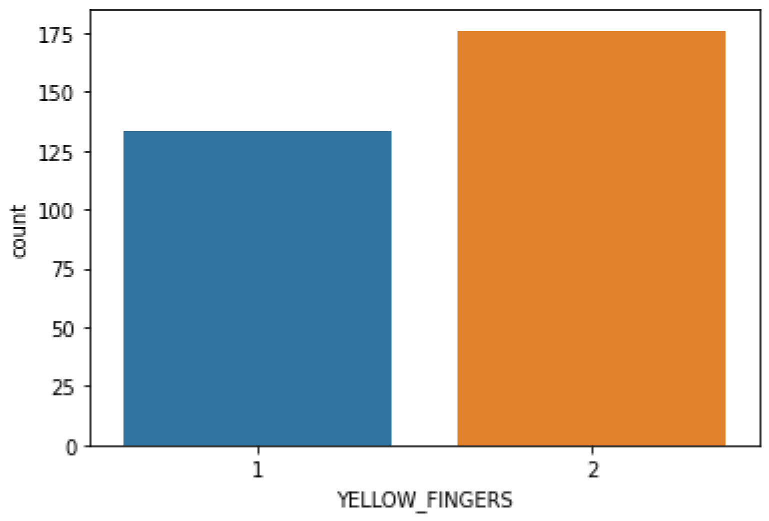
```
sns.countplot(x = 'LUNG_CANCER',data = dataset)
```



The data has more present lung cancer data. So the predicted value can be biased to the more data we need to resampled for this. I will show the resampled in the following of the report after splitting the data into train and test sets.

Analyzing YELLOW FINGERS Column

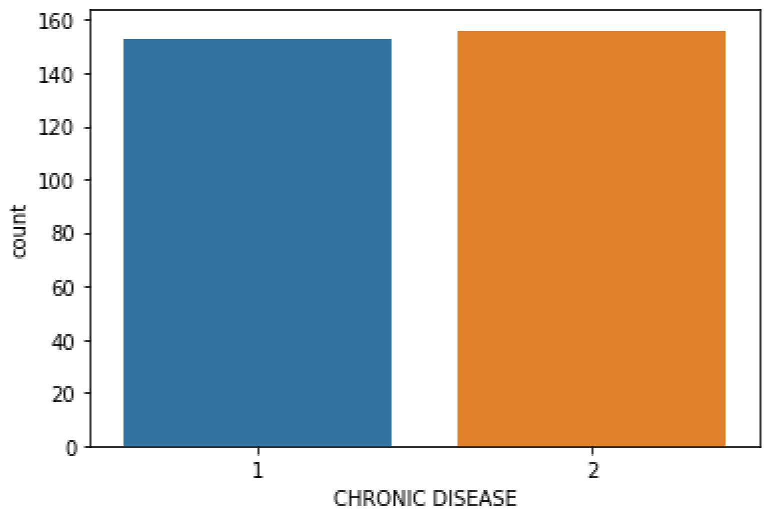
```
sns.countplot(x = 'YELLOW_FINGERS',data = dataset)
```



In this column the data is a little bit unbalanced.

Analyzing CHRONIC DISEASE Column

```
sns.countplot(x = 'CHRONIC DISEASE',data = dataset)
```



The data are nearly equally distributed.

Label Encoding

```
le = preprocessing.LabelEncoder()
dataset['GENDER'] = le.fit_transform(dataset['GENDER'])
dataset['LUNG_CANCER'] = le.fit_transform(dataset['LUNG_CANCER'])
```

Label Encoding is needed for two objects such as LUNG_CANCER whose values are (Yes, NO) and GENDER (M,F). It will change the values from categories to numerical here (0,1). Label encoding is required for training the data. Now all the values are integers type.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a critical initial step in the data analysis process that involves exploring and summarizing the main characteristics, patterns, and relationships present in a dataset. It's performed to gain insights, discover patterns, and identify potential issues or trends within the data.

Why Perform EDA?

EDA helps understand the structure, nature, and content of the dataset. It gives an overview of what the data contains and how it's organized.

EDA uncovers patterns, trends, correlations, and relationships among variables, helping to generate hypotheses and insights.

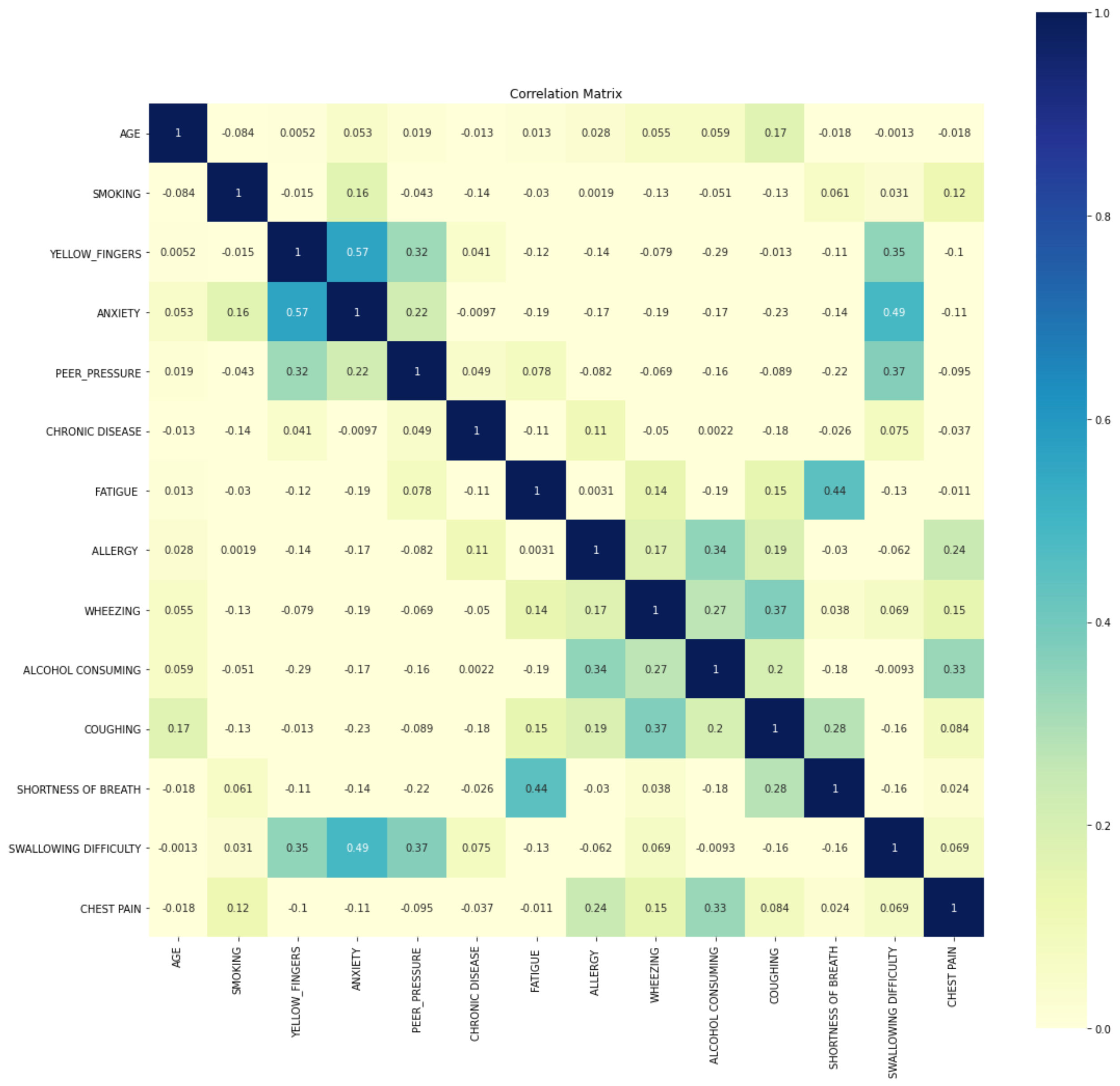
It helps identify potential errors, outliers, missing values, or inconsistencies in the data that may impact subsequent analyses or modeling.

EDA aids in selecting relevant features (variables) for modeling and in creating new informative features through feature engineering.

Correlation Matrix

Here Anxiety and Yellow fingers (0.57) columns are more correlated than other columns and rest are (Yellow Fingers & Peer Pressure, 0.32), (Anxiety & Swallowing Difficulties, 0.49), (Wheezing & coughing , 0.37), (Fatigue & Shortness of Breath, 0.44)

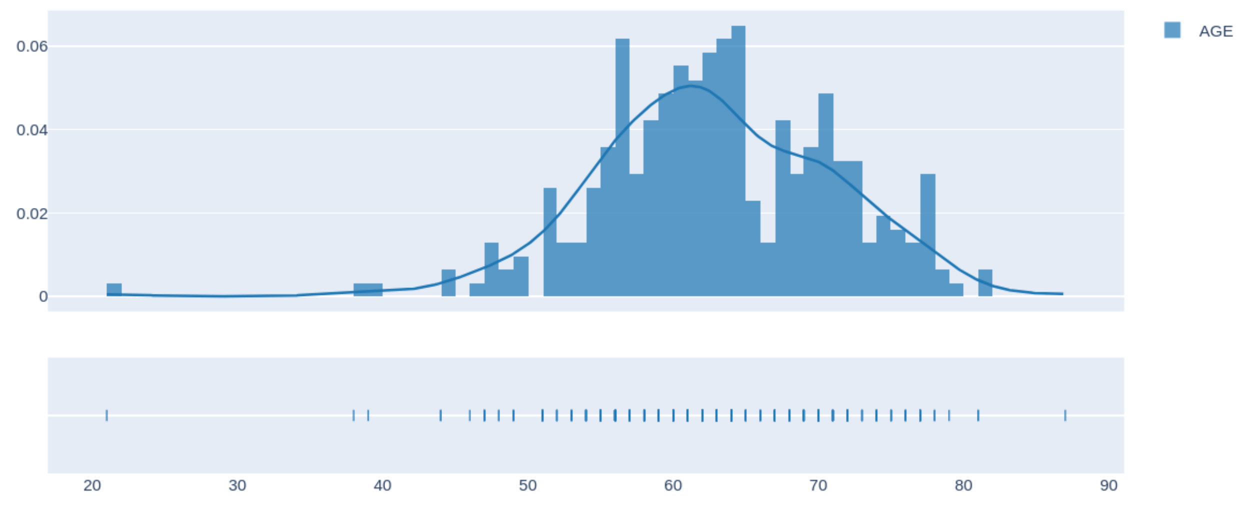
```python
plt.figure(figsize=(18, 18))  # Set the size of the plot

# Customize the heatmap using Seaborn
sns.heatmap(correlation_matrix,annot=True, square=True, vmin=0, vmax=1,cmap="YlGnBu");
plt.title('Correlation Matrix')  # Set the title of the plot
plt.show()
```



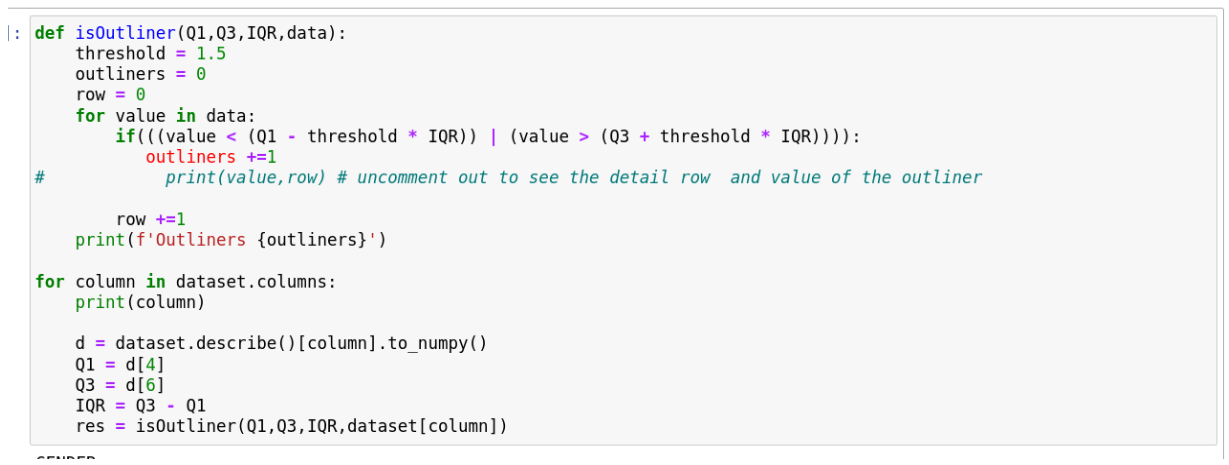
Detecting Outliers

I used Interquartile range, Q1 AND Q3 to detect outliers in the dataset and found 2 in AGE column the value and row are like this

Value	Row
21	22
38	238



Histogram of AGE Column



```
def isOutliner(Q1,Q3,IQR,data):
    threshold = 1.5
    outliners = 0
    row = 0
    for value in data:
        if(((value < (Q1 - threshold * IQR)) | (value > (Q3 + threshold * IQR)))):
            outliners +=1
#             print(value,row) # uncomment out to see the detail row  and value of the outliner

        row +=1
    print(f'Outliners {outliners}')

for column in dataset.columns:
    print(column)

    d = dataset.describe()[column].to_numpy()
    Q1 = d[4]
    Q3 = d[6]
    IQR = Q3 - Q1
    res = isOutliner(Q1,Q3,IQR,dataset[column])
```

MODEL & Result & Analysis

The model we need here is to predict binary classification (yes, no) for lung cancer. So we can’t use the continuous outcome models and multiple regression models.

The models that I used in this project are

1. Random Forest Classifier

2. Decision Tree
3. Voting Classifier
4. Logistic Regression
5. Artificial Neural Network (ANN)
6. K Nearest Neighbour (KNN)

Here there is a new model which didn’t cover in the class, the Voting Classifier (VC).

A Voting Classifier is an ensemble machine learning technique that combines the predictions from multiple individual machine learning models to make a final prediction. It aggregates the predictions of each classifier and outputs the class label that received the most votes (or highest probability in the case of soft voting).
There are two main types of Voting Classifiers:

Hard Voting: In hard voting, the final prediction is determined by a simple majority vote among the individual classifiers. The class that receives the most votes from the individual classifiers is selected as the final prediction.

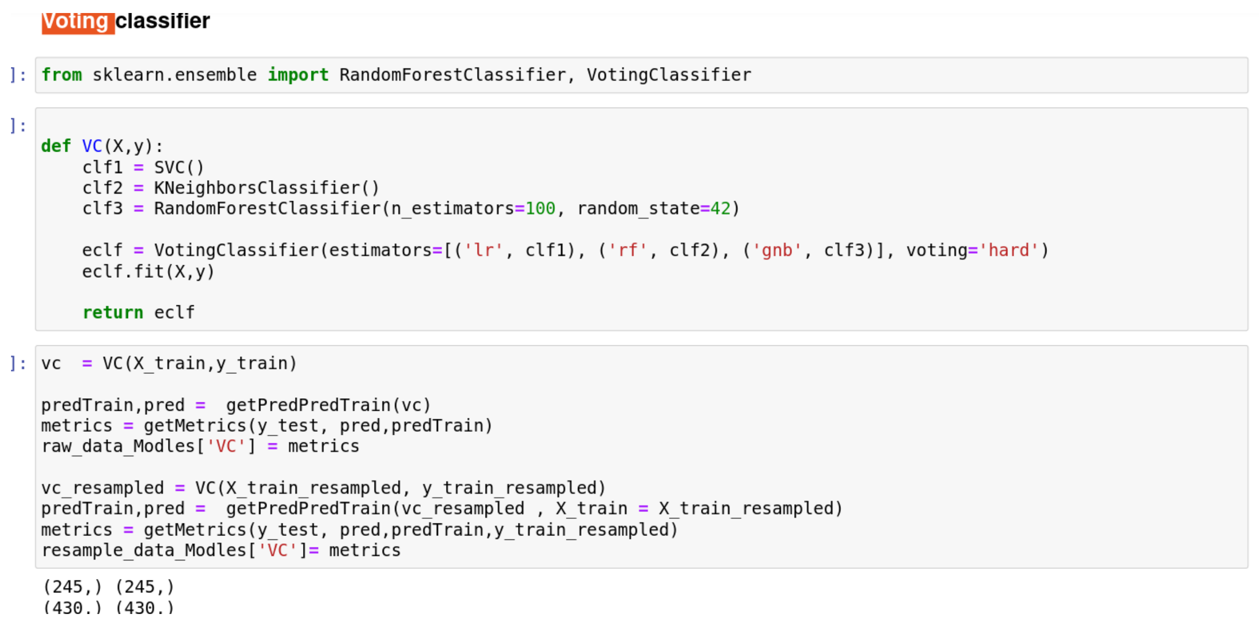
Soft Voting: In soft voting, each classifier's prediction is associated with a probability score for each class. The final prediction is determined by averaging the predicted probabilities for each class across all classifiers and selecting the class with the highest average probability.

Classifier Diversity: Voting classifiers work well when the individual classifiers are diverse in terms of their learning algorithms, data representation, or training process. When diverse classifiers are combined, they can compensate for each other's weaknesses and improve overall performance.

Performance Improvement: Voting classifiers can sometimes outperform individual classifiers by reducing variance and improving generalization. They tend to perform better when the individual classifiers have similar accuracy but make different types of errors.

Usage: They are commonly used with a variety of classifiers, including decision trees, logistic regression, support vector machines (SVM), k-nearest neighbors (KNN), etc.
Implementation: In Python, sci

Here is the detailed implementation of the VC model.



Voting classifier

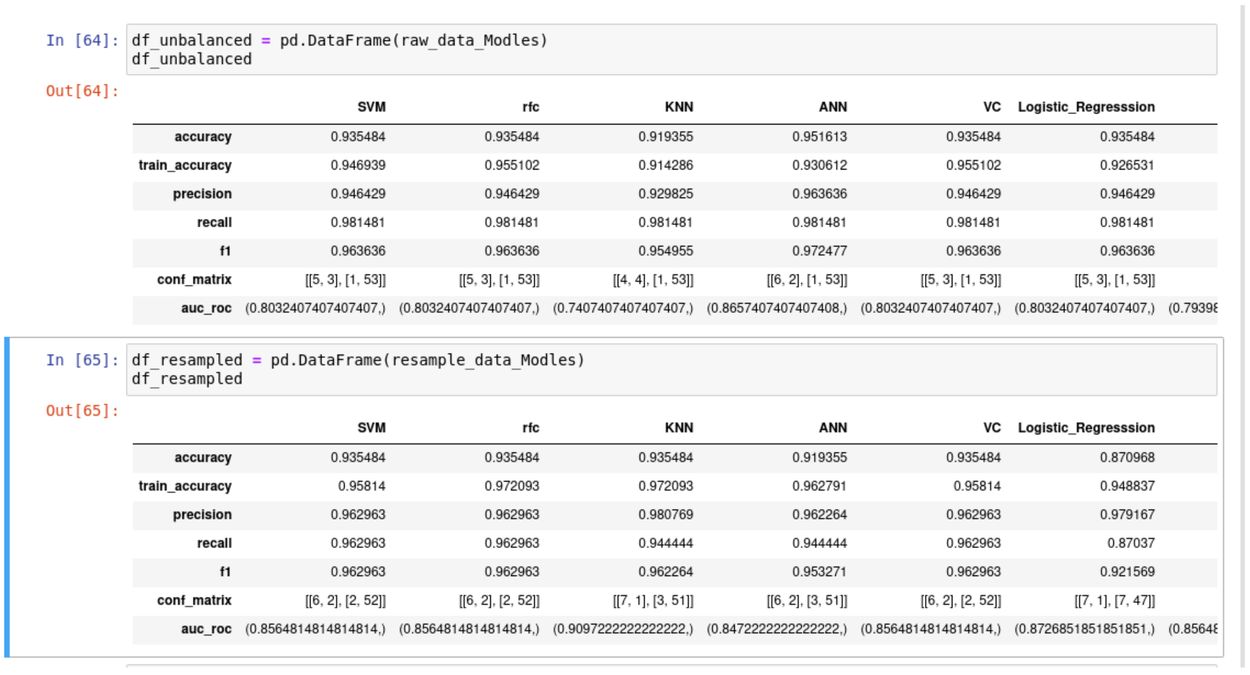
```
]: from sklearn.ensemble import RandomForestClassifier, VotingClassifier
```

```
]:
def VC(X,y):
    clf1 = SVC()
    clf2 = KNeighborsClassifier()
    clf3 = RandomForestClassifier(n_estimators=100, random_state=42)

    eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting='hard')
    eclf.fit(X,y)

    return eclf
```

```
]: vc  = VC(X_train,y_train)

predTrain,pred =  getPredPredTrain(vc)
metrics = getMetrics(y_test, pred,predTrain)
raw_data_Modles['VC'] = metrics

vc_resampled = VC(X_train_resampled, y_train_resampled)
predTrain,pred =  getPredPredTrain(vc_resampled , X_train = X_train_resampled)
metrics = getMetrics(y_test, pred,predTrain,y_train_resampled)
resample_data_Modles['VC']= metrics
```

```
(245,) (245,)
(430,) (430,)
```

The accuracy, train_accuracy, precision, recall, f1 score, conf_matrix and auc_roc values for models trained on unbalanced data are as follow.Here are the results of models trained on imbalance data and resampled data.



```
In [64]: df_unbalanced = pd.DataFrame(raw_data_Modles)
         df_unbalanced
```

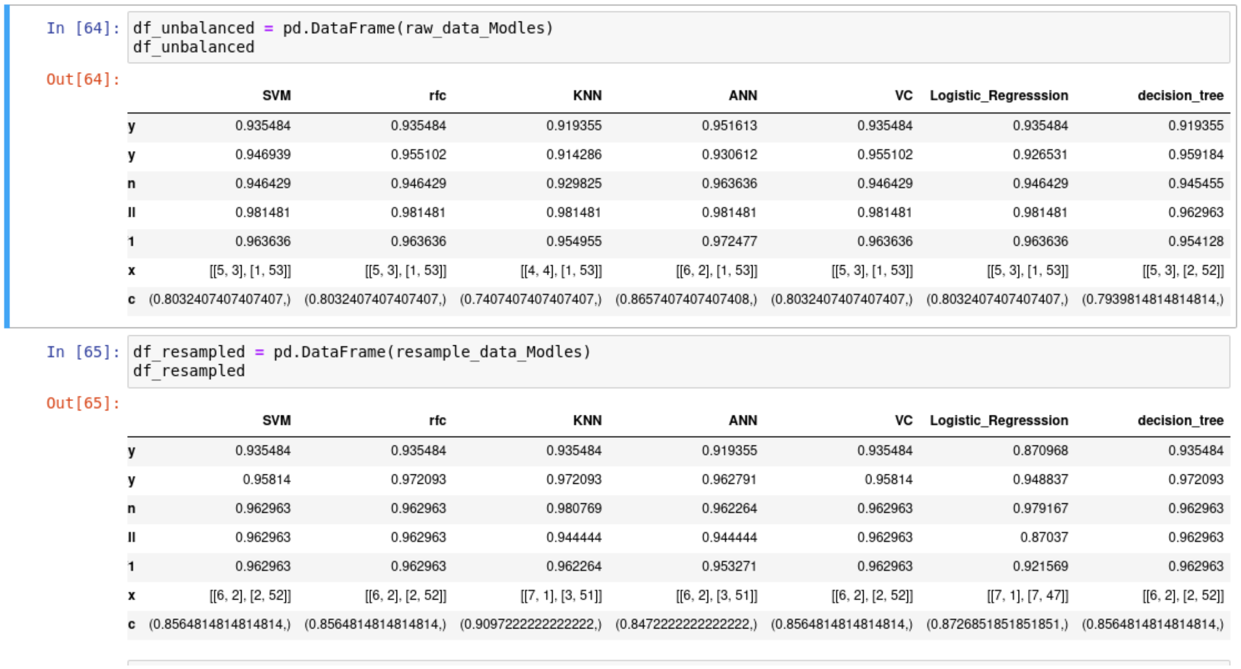
Out[64]:

	SVM	rfc	KNN	ANN	VC	Logistic_Regresssion	
accuracy	0.935484	0.935484	0.919355	0.951613	0.935484	0.935484	
train_accuracy	0.946939	0.955102	0.914286	0.930612	0.955102	0.926531	
precision	0.946429	0.946429	0.929825	0.963636	0.946429	0.946429	
recall	0.981481	0.981481	0.981481	0.981481	0.981481	0.981481	
f1	0.963636	0.963636	0.954955	0.972477	0.963636	0.963636	
conf_matrix	[[5, 3], [1, 53]]	[[5, 3], [1, 53]]	[[4, 4], [1, 53]]	[[6, 2], [1, 53]]	[[5, 3], [1, 53]]	[[5, 3], [1, 53]]	
auc_roc	(0.8032407407407407,)	(0.8032407407407407,)	(0.7407407407407407,)	(0.8657407407407408,)	(0.8032407407407407,)	(0.8032407407407407,)	(0.79398

```
In [65]: df_resampled = pd.DataFrame(resample_data_Modles)
         df_resampled
```

Out[65]:

	SVM	rfc	KNN	ANN	VC	Logistic_Regresssion	
accuracy	0.935484	0.935484	0.935484	0.919355	0.935484	0.870968	
train_accuracy	0.95814	0.972093	0.972093	0.962791	0.95814	0.948837	
precision	0.962963	0.962963	0.980769	0.962264	0.962963	0.979167	
recall	0.962963	0.962963	0.944444	0.944444	0.962963	0.87037	
f1	0.962963	0.962963	0.962264	0.953271	0.962963	0.921569	
conf_matrix	[[6, 2], [2, 52]]	[[6, 2], [2, 52]]	[[7, 1], [3, 51]]	[[6, 2], [3, 51]]	[[6, 2], [2, 52]]	[[7, 1], [7, 47]]	
auc_roc	(0.8564814814814814,)	(0.8564814814814814,)	(0.9097222222222222,)	(0.8472222222222222,)	(0.8564814814814814,)	(0.8726851851851851,)	(0.85648



```
In [64]: df_unbalanced = pd.DataFrame(raw_data_Modles)
         df_unbalanced
Out[64]:
```

	SVM	rfc	KNN	ANN	VC	Logistic_Regresssion	decision_tree
y	0.935484	0.935484	0.919355	0.951613	0.935484	0.935484	0.919355
y	0.946939	0.955102	0.914286	0.930612	0.955102	0.926531	0.959184
n	0.946429	0.946429	0.929825	0.963636	0.946429	0.946429	0.945455
ll	0.981481	0.981481	0.981481	0.981481	0.981481	0.981481	0.962963
1	0.963636	0.963636	0.954955	0.972477	0.963636	0.963636	0.954128
x	[[5, 3], [1, 53]]	[[5, 3], [1, 53]]	[[4, 4], [1, 53]]	[[6, 2], [1, 53]]	[[5, 3], [1, 53]]	[[5, 3], [1, 53]]	[[5, 3], [2, 52]]
c	(0.8032407407407407,)	(0.8032407407407407,)	(0.7407407407407407,)	(0.8657407407407408,)	(0.8032407407407407,)	(0.8032407407407407,)	(0.7939814814814814,)

```
In [65]: df_resampled = pd.DataFrame(resample_data_Modles)
         df_resampled
Out[65]:
```

	SVM	rfc	KNN	ANN	VC	Logistic_Regresssion	decision_tree
y	0.935484	0.935484	0.935484	0.919355	0.935484	0.870968	0.935484
y	0.95814	0.972093	0.972093	0.962791	0.95814	0.948837	0.972093
n	0.962963	0.962963	0.980769	0.962264	0.962963	0.979167	0.962963
ll	0.962963	0.962963	0.944444	0.944444	0.962963	0.87037	0.962963
1	0.962963	0.962963	0.962264	0.953271	0.962963	0.921569	0.962963
x	[[6, 2], [2, 52]]	[[6, 2], [2, 52]]	[[7, 1], [3, 51]]	[[6, 2], [3, 51]]	[[6, 2], [2, 52]]	[[7, 1], [7, 47]]	[[6, 2], [2, 52]]
c	(0.8564814814814814,)	(0.8564814814814814,)	(0.9097222222222222,)	(0.8472222222222222,)	(0.8564814814814814,)	(0.8726851851851851,)	(0.8564814814814814,)

As You can see that the SVM,RFC and Decision tree models are the best on resampled data. You can also recognize that RFC is prone to imbalance data and much better on accuracy on imbalance data. The resample data has lesser data and causes low accuracy but higher precision on resample data.

Most of the models don't have overfitting because the difference between accuracy on the test set and train set (train_accuracy) is not that much different.

The choice of the "best" model for lung cancer prediction depends on the specific needs of the application:

- If Balancing Precision and Recall: SVM, RFC, KNN, ANN, VC, and Decision Tree perform well in both precision and recall. Consider these for a balance between correctly identifying positive cases and minimizing false positives.
- If Prioritizing Precision: KNN trained on resampled data demonstrates the highest precision (0.980769).
- If Simplicity or Interpretability is Essential: Decision Tree or Logistic Regression might be preferable due to their simplicity and interpretability.

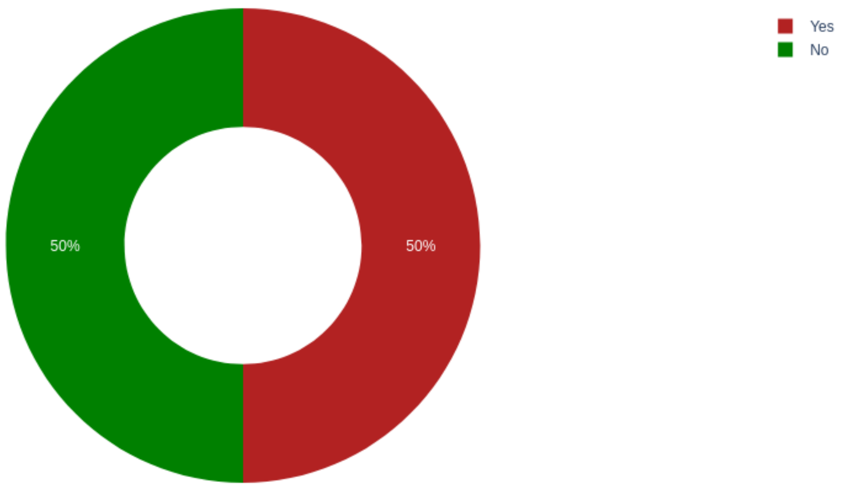
Here for Lung Cancer detection, I think even though the difference between false positive and false negative can’t be huge. But in the health related model the precision is more important otherwise we will classify the people who don't have disease as having one.

For these things KNN model on resampled data is best in overall even the recall is slightly lower than other tree models but not that different and best in overall and not complicated and more easier to interpret than other models such SVM and ANN.

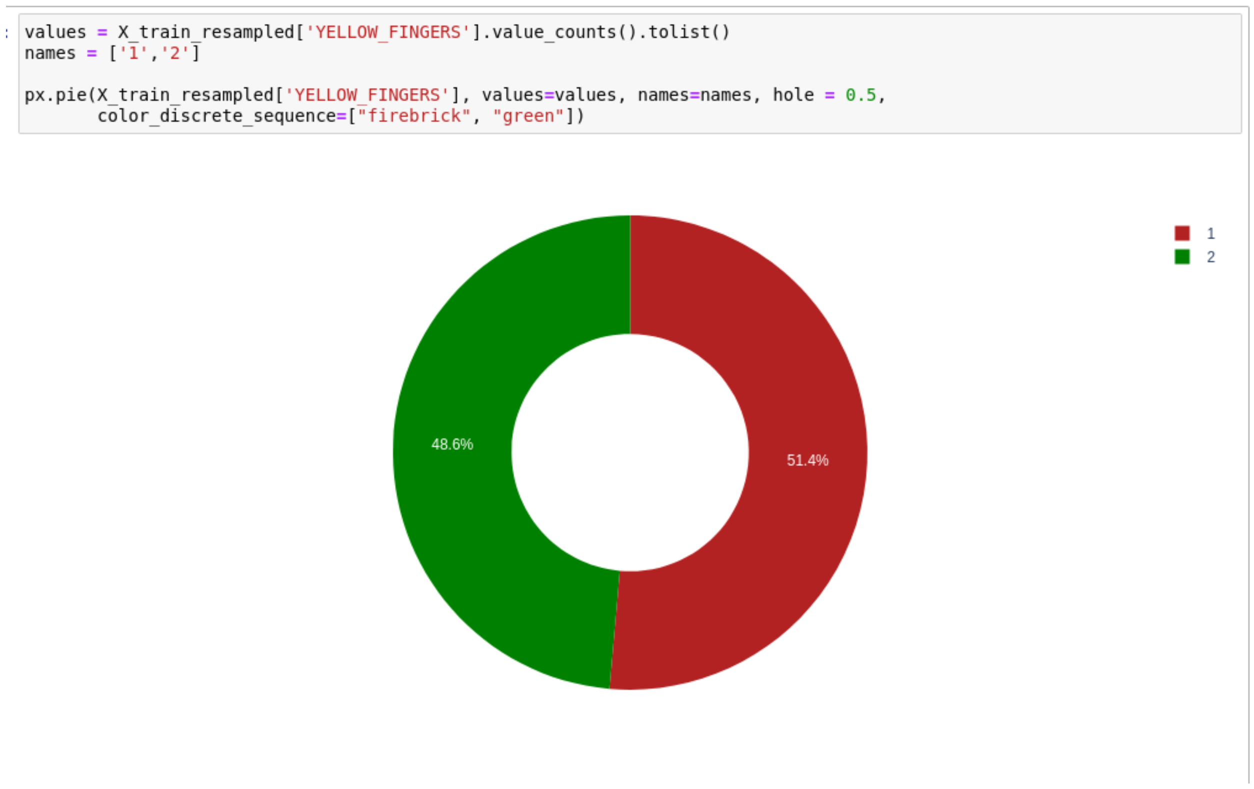
For **Resampling SMOTE technique** is used here.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
values = y_train_resampled.value_counts().tolist()
names = ['Yes','No']

px.pie(y_train_resampled, values=values, names=names, hole = 0.5,
       color_discrete_sequence=["firebrick", "green"])
```



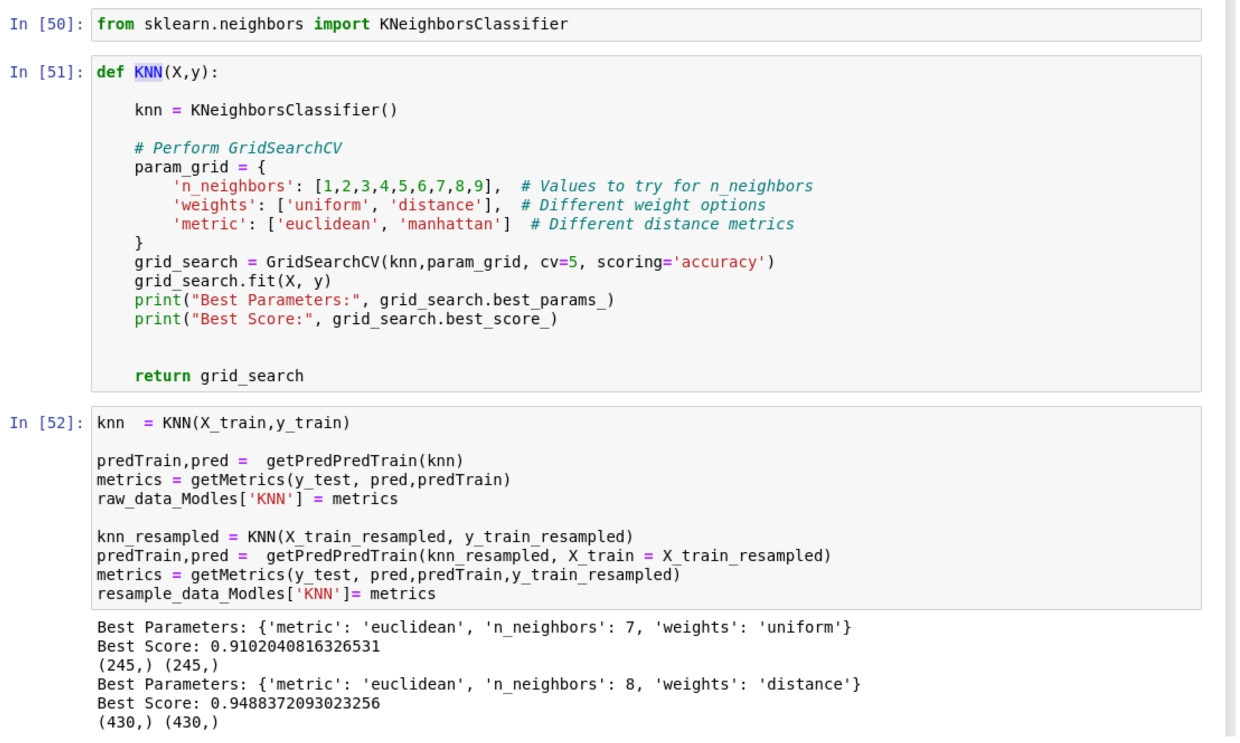
Here the y_train column is LUNG_CANCER which is strongly imbalance in the original data which could hugely impact the model's accuracy.


```
values = X_train_resampled['YELLOW_FINGERS'].value_counts().tolist()
names = ['1','2']

px.pie(X_train_resampled['YELLOW_FINGERS'], values=values, names=names, hole = 0.5,
       color_discrete_sequence=["firebrick", "green"])
```

Above diagram is the data distribution of the Yellow Finger column after resampling on the Train Set which was significantly imbalanced on the original data.

For the detailed implementation of other models please check the jupyter notebook.

This is the model implementation



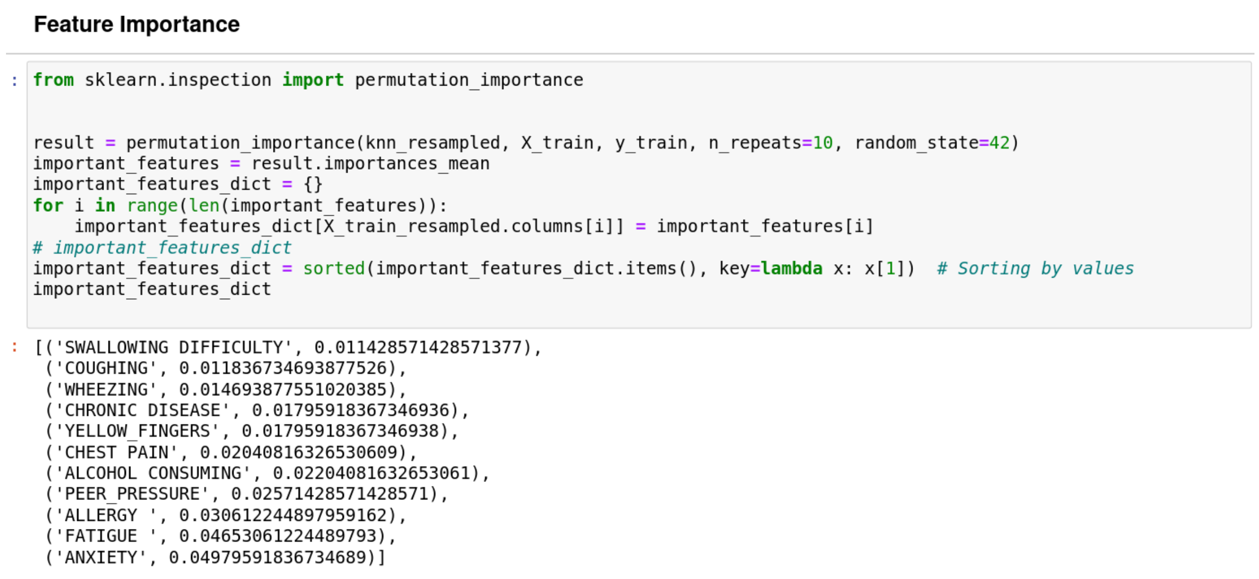
```
In [50]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [51]: def KNN(X,y):

             knn = KNeighborsClassifier()

             # Perform GridSearchCV
             param_grid = {
                 'n_neighbors': [1,2,3,4,5,6,7,8,9],  # Values to try for n_neighbors
                 'weights': ['uniform', 'distance'],  # Different weight options
                 'metric': ['euclidean', 'manhattan']  # Different distance metrics
             }
             grid_search = GridSearchCV(knn,param_grid, cv=5, scoring='accuracy')
             grid_search.fit(X, y)
             print("Best Parameters:", grid_search.best_params_)
             print("Best Score:", grid_search.best_score_)


             return grid_search
```

```
In [52]: knn  = KNN(X_train,y_train)

         predTrain,pred =  getPredPredTrain(knn)
         metrics = getMetrics(y_test, pred,predTrain)
         raw_data_Modles['KNN'] = metrics

         knn_resampled = KNN(X_train_resampled, y_train_resampled)
         predTrain,pred =  getPredPredTrain(knn_resampled, X_train = X_train_resampled)
         metrics = getMetrics(y_test, pred,predTrain,y_train_resampled)
         resample_data_Modles['KNN']= metrics

         Best Parameters: {'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'uniform'}
         Best Score: 0.9102040816326531
         (245,) (245,)
         Best Parameters: {'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'distance'}
         Best Score: 0.9488372093023256
         (430,) (430,)
```

Here created the KNN model using GridSearchCV and trained on both imbalance data and resampled data.
The best n_neighbour for imbalance data and resampled data are 7 , 8 and for weights (uniform , distance) respectively.

Let's calculate the important features of this model.



Feature Importance

```
: from sklearn.inspection import permutation_importance


  result = permutation_importance(knn_resampled, X_train, y_train, n_repeats=10, random_state=42)
  important_features = result.importances_mean
  important_features_dict = {}
  for i in range(len(important_features)):
      important_features_dict[X_train_resampled.columns[i]] = important_features[i]
  # important_features_dict
  important_features_dict = sorted(important_features_dict.items(), key=lambda x: x[1])  # Sorting by values
  important_features_dict
```

```
: [('SWALLOWING DIFFICULTY', 0.011428571428571377),
   ('COUGHING', 0.011836734693877526),
   ('WHEEZING', 0.014693877551020385),
   ('CHRONIC DISEASE', 0.01795918367346936),
   ('YELLOW_FINGERS', 0.01795918367346938),
   ('CHEST PAIN', 0.02040816326530609),
   ('ALCOHOL CONSUMING', 0.02204081632653061),
   ('PEER_PRESSURE', 0.02571428571428571),
   ('ALLERGY ', 0.030612244897959162),
   ('FATIGUE ', 0.04653061224489793),
   ('ANXIETY', 0.04979591836734689)]
```

Retraining KNN model using the Important Features

In the following techniques, the KNN model is retrained by dropping some less important features. And here you will see the accuracy of the model is not getting better with these techniques.

Retraining KNN using Feature Importance Selection Technique

```
: from sklearn.feature_selection import SelectKBest, f_classif
  model = knn_resampled

  X_train2, y_train2 = X_train_resampled,y_train_resampled
  X_test2 = X_test
  X_train2 = X_train2.drop(['SWALLOWING DIFFICULTY'],axis=1)
  X_test2  = X_test2.drop(['SWALLOWING DIFFICULTY'],axis=1)
```

```
: knn_resampled2 = KNN(X_train2,y_train2)
  predTrain,pred =  getPredPredTrain(knn_resampled2,X_test2, X_train = X_train2)
  metrics = getMetrics(y_test, pred,predTrain,y_train2)
  metrics
```

```
  Best Parameters: {'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'uniform'}
  Best Score: 0.9534883720930232
  (430,) (430,)
```

```
: {'accuracy': 0.9193548387096774,
   'train_accuracy': 0.9558139534883721,
   'precision': 0.9803921568627451,
   'recall': 0.9259259259259259,
   'f1': 0.9523809523809523,
   'conf_matrix': array([[ 7,  1],
          [ 4, 50]]),
   'auc_roc': (0.900462962962963,)}
```

```
1]: X_train2, y_train2 = X_train_resampled,y_train_resampled
    X_test2 = X_test
    X_train2 = X_train2.drop(['SWALLOWING DIFFICULTY','COUGHING'],axis=1)
    X_test2  = X_test.drop(['SWALLOWING DIFFICULTY','COUGHING'],axis=1)
```

```
2]: knn_resampled2 = KNN(X_train2,y_train2)
    predTrain,pred =  getPredPredTrain(knn_resampled2,X_test2, X_train = X_train2)
    metrics = getMetrics(y_test, pred,predTrain,y_train2)
    metrics
```

```
    Best Parameters: {'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'distance'}
    Best Score: 0.9465116279069766
    (430,) (430,)
```

```
2]: {'accuracy': 0.9354838709677419,
     'train_accuracy': 0.9697674418604652,
     'precision': 0.9807692307692307,
     'recall': 0.9444444444444444,
     'f1': 0.9622641509433962,
     'conf_matrix': array([[ 7,  1],
            [ 3, 51]]),
     'auc_roc': (0.9097222222222222,)}
```

```
X_train2, y_train2 = X_train_resampled,y_train_resampled
X_test2 = X_test
X_train2 = X_train2.drop(['SWALLOWING DIFFICULTY','COUGHING','WHEEZING'],axis=1)
X_test2  = X_test.drop(['SWALLOWING DIFFICULTY','COUGHING','WHEEZING'],axis=1)
```

```
knn_resampled2 = KNN(X_train2,y_train2)
predTrain,pred =  getPredPredTrain(knn_resampled2,X_test2, X_train = X_train2)
metrics = getMetrics(y_test, pred,predTrain,y_train2)
metrics
```

```
Best Parameters: {'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'distance'}
Best Score: 0.9372093023255814
(430,) (430,)

{'accuracy': 0.91935483870967774,
 'train_accuracy': 0.9651162790697675,
 'precision': 0.9803921568627451,
 'recall': 0.9259259259259259,
 'f1': 0.9523809523809523,
 'conf_matrix': array([[ 7,  1],
        [ 4, 50]]),
 'auc_roc': (0.900462962962963,)}
```

Discussion & Conclusion

The KNN with accuracy of 0.935 is a pretty high model but we could make it better by adding more data to the dataset. I tried to retrain the model by removing the less important feature but the accuracy is not better and even decreases when the removed more less important features.Since the Health Related model like this where the precision and recall is very critical and the resulting KNN have achieved acceptable performance already if we have more data we can make more prediction and more conclusion on the model.