LNMIIT
The LNM Institute of Information Technology

Introduction to Data Science

Project Report :

Lung Cancer Detection Model

Submitted by:
Group - CLUSTERS

Charu Sharma (20ucs055)
Himanshu Sharma (20ucs080)
Hritul Srivastava (20ucs082)
Subodh (20ucc106)

Submitted to:
Course Instructor
Dr. Sakthi Balan Muthiah
Associate Professor
Dept. of CSE, LNMIIT

LNMIIT
The LNM Institute of Information Technology

INDEX

LNMIIT
The LNM Institute of Information Technology

Project Objective

Based on certain diagnostic tests used in the dataset, the purpose of the dataset is to forecast whether or not a patient has diabetes. The databases consist of several variables for medical prediction and one target variable, the outcome. Predictor variables include the number of births, their BMI, insulin level, age, and so on that the patient has had.

The Dataset Name and Origin

Name : Lung Cancer Dataset
Origin : www.kaggle.com/datasets
Dataset URL :
https://raw.githubusercontent.com/apoc4080/ids_project/aa62167f28081e976b6b11bbc14c299852637512/dataset2.csv

Dataset Specification

Abstract: This diabetes dataset is from AIM '94

Data Set Characteristics:	Multivariate, Time-Series	**Number of Instances:**	N/A	**Area:**	Life
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	20	**Date Donated**	N/A
Associated Tasks:	N/A	**Missing Values?**	N/A	**Number of Web Hits:**	524331

LNMIIT
The LNM Institute of Information Technology

Dataset Description

Diabetes files are made up of 4 fields per document. Each area is divided by a tab and a new line divides each record.

Format:

1. Gender: {11: Male 10: Female}
2. Age: {numerical}
3. Smoking: {1: No 2:Yes}
4. Yellow Fingers: {1: No 2: Yes}
5. Anxiety: {1: No 2: Yes}
6. Peer Pressure: {1: No 2: Yes}
7. Chronic Disease: {1: No 2: Yes}
8. Fatigue: {1: No 2: Yes}
9. Allergy: {1: No 2: Yes}
10. Wheezing: {1: No 2: Yes}
11. Alcohol: {1: No 2: Yes}
12. Coughing: {1: No 2: Yes}
13. Shortness of breath: {1: No 2: Yes}
14. Swallowing Difficulty: {1: No 2: Yes}
15. Chest Pain: {1: No 2: Yes}
16. Lung Cancer: {0: No 1: Yes}

Importing Libraries

```
[1]  import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import confusion_matrix,accuracy_score,plot_confusion_matrix
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import cross_val_score
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.preprocessing import LabelEncoder
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.naive_bayes import GaussianNB
     from sklearn.svm import SVC
     from sklearn.model_selection import GridSearchCV
```

Descriptive Statistics

```
import warnings
warnings.filterwarnings('ignore')
url = 'https://raw.githubusercontent.com/apoc4080/ids_project/aa62167f28081e976b6b11bbc14c299852637512/dataset2.csv'

dataset = pd.read_csv(url)
dataset.head()
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_CANCER
0	11	69	1	2	2	1	1	2	1	2	2	2	2	2	2	1
1	11	74	2	1	1	1	2	2	2	1	1	1	2	2	2	1
2	10	59	1	1	1	2	1	2	1	2	1	2	2	1	2	0
3	11	63	2	2	2	1	1	1	1	1	2	1	1	2	2	0
4	10	63	1	2	1	1	1	1	1	2	1	2	2	1	1	0

```
[2]  dataset.shape
```

```
(309, 16)
```

```
[3]  dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 309 entries, 0 to 308
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 309 non-null    int64
 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    int64
dtypes: int64(16)
memory usage: 38.8 KB
```

Statistical Summary:

```
[4]  dataset.describe().T
```

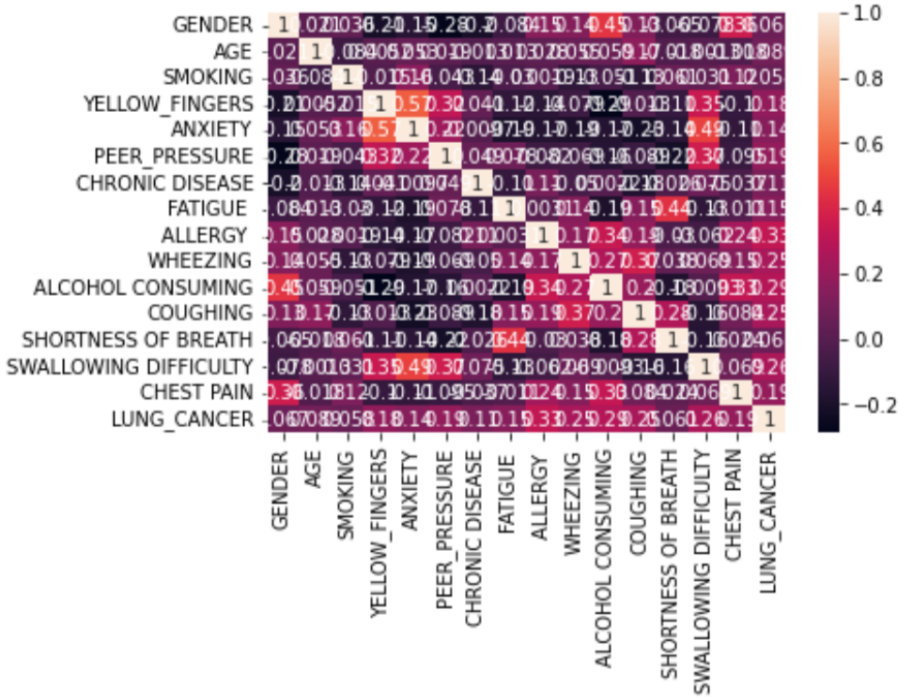
	count	mean	std	min	25%	50%	75%	max
GENDER	309.0	10.524272	0.500221	10.0	10.0	11.0	11.0	11.0
AGE	309.0	62.673139	8.210301	21.0	57.0	62.0	69.0	87.0
SMOKING	309.0	1.563107	0.496806	1.0	1.0	2.0	2.0	2.0
YELLOW_FINGERS	309.0	1.569579	0.495938	1.0	1.0	2.0	2.0	2.0
ANXIETY	309.0	1.498382	0.500808	1.0	1.0	1.0	2.0	2.0
PEER_PRESSURE	309.0	1.501618	0.500808	1.0	1.0	2.0	2.0	2.0
CHRONIC DISEASE	309.0	1.504854	0.500787	1.0	1.0	2.0	2.0	2.0
FATIGUE	309.0	1.673139	0.469827	1.0	1.0	2.0	2.0	2.0
ALLERGY	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0
WHEEZING	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0
ALCOHOL CONSUMING	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0
COUGHING	309.0	1.579288	0.494474	1.0	1.0	2.0	2.0	2.0
SHORTNESS OF BREATH	309.0	1.640777	0.480551	1.0	1.0	2.0	2.0	2.0
SWALLOWING DIFFICULTY	309.0	1.469256	0.499863	1.0	1.0	1.0	2.0	2.0
CHEST PAIN	309.0	1.556634	0.497588	1.0	1.0	2.0	2.0	2.0
LUNG_CANCER	309.0	0.873786	0.332629	0.0	1.0	1.0	1.0	1.0

Basic Observation :

1. The dataset has a total of 309 records and 16 functions.
2. Each function may have either an integer or a float form of data.
3. No attributes have any missing values/information.
4. 1 indicates the person has Lung Cancer whereas 0 indicates the person doesn’t have Lung Cancer.

Correlation Heatmap:

```
[6]  sns.heatmap(dataset.corr(),annot = True)
     plt.show()
```



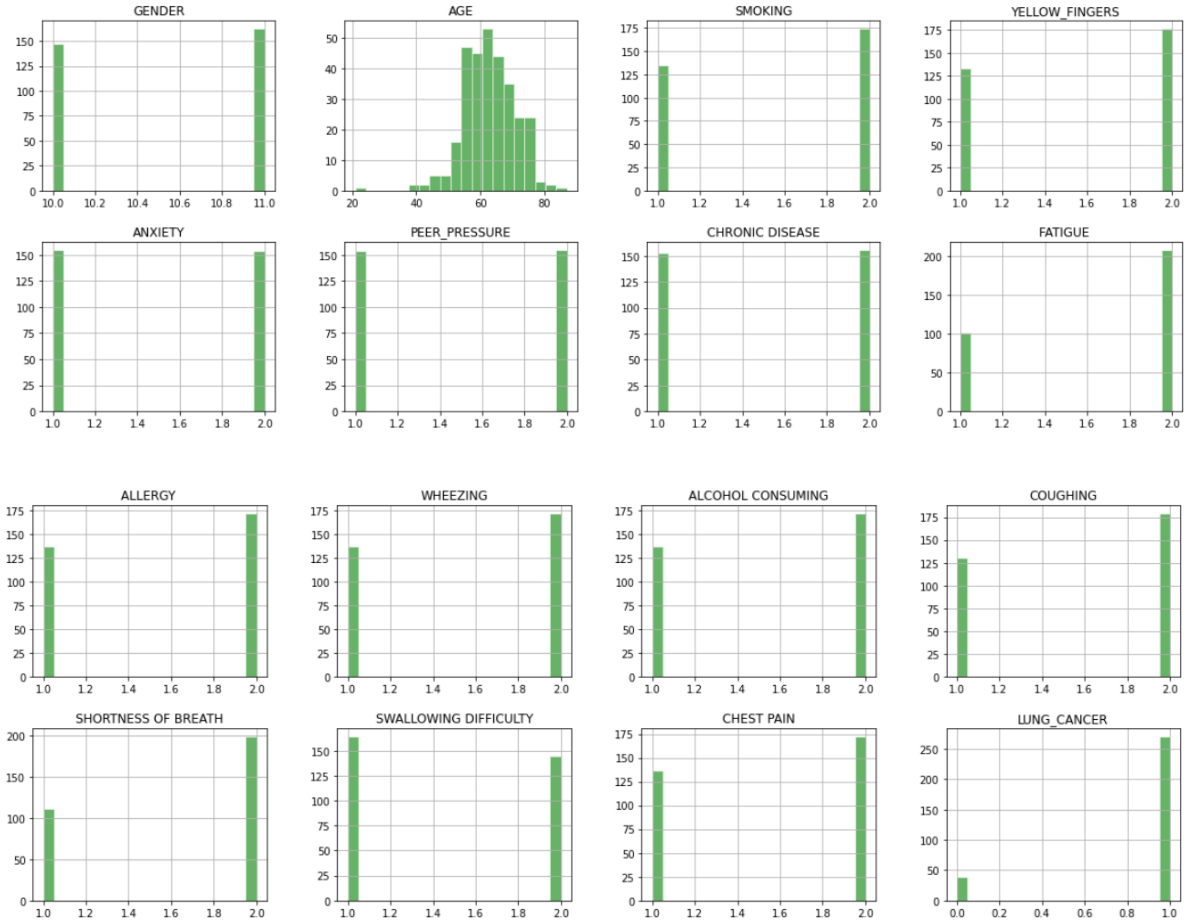
Inference from heatmap:

We can see that the correlation heat map has a high relationship between the outcome and [Alcohol Consumption, Peer Pressure, Yellow Fingers, Fatigue]. We can choose these characteristics to take feedback from the consumer and forecast the effect.

Data Visualisation

- First, we use **DataFrame.hist()** to plot a histogram. This histogram shows us the range of values for all our attributes in the dataset.

```
[7] dataset.hist(figsize=(20,15),ec='white',bins=20,color='green',alpha=0.6)
    plt.show()
```



Inference:

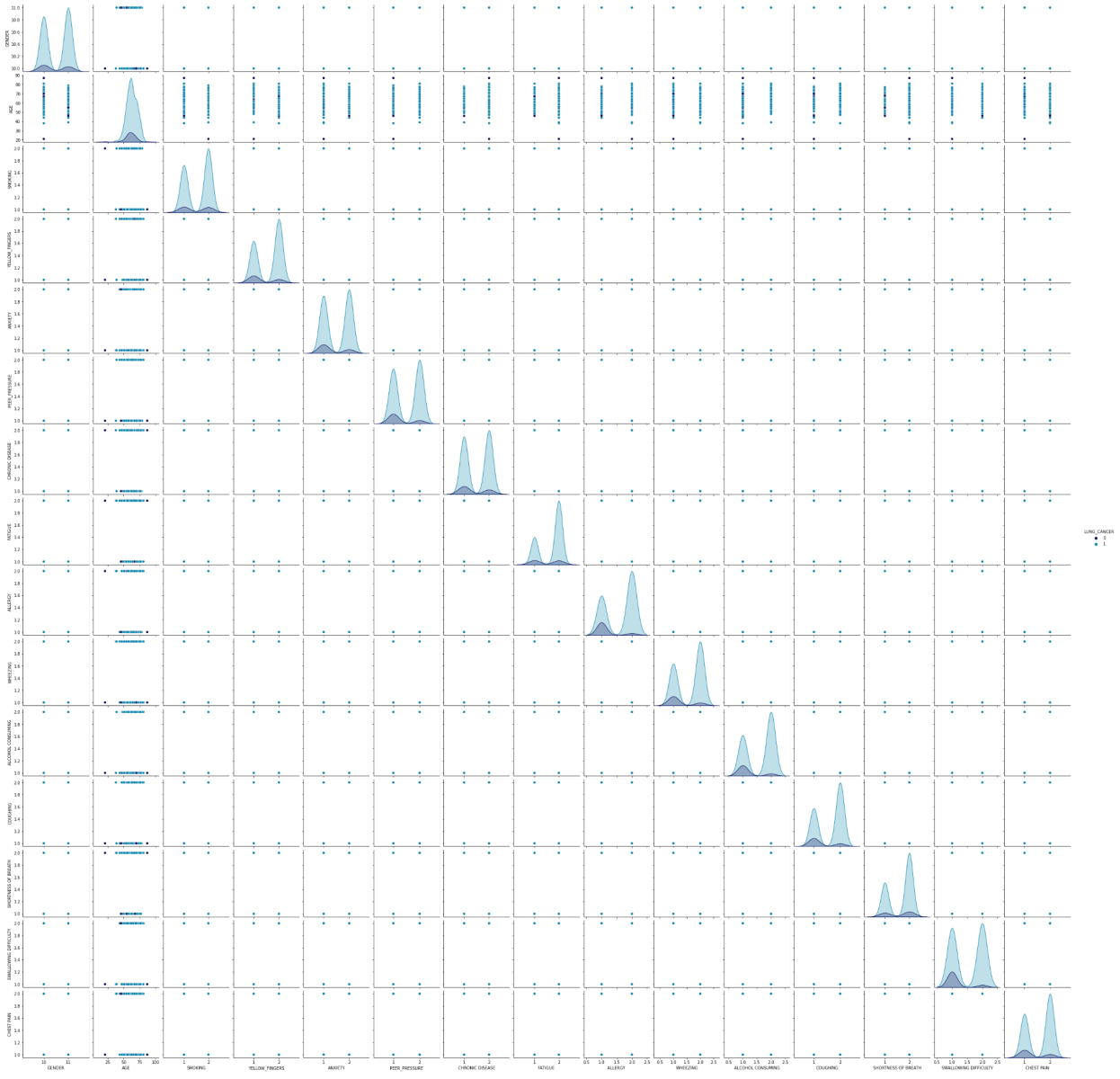
- This graph gives the histogram plots for each attribute in our dataset.
- It tells us the count of appearances of each value in each of the attributes.
- The x-values give the range of values of the particular attribute and the y-values give the count of each range.

Next, we use **seaborn.pairplot()** to pairwise plot our attributes.

- The Diagonal Plots Are Univariate. The Rest Are Bivariate Plots.
- It shows the (n,2) combination relationship of attributes of the dataset.

```
[8]  plt.figure(figsize=(10,5))
     sns.pairplot(dataset, hue="LUNG_CANCER", palette="ocean",diag_kind="kde")
```

```
<seaborn.axisgrid.PairGrid at 0x7f698aa49850>
<Figure size 720x360 with 0 Axes>
```



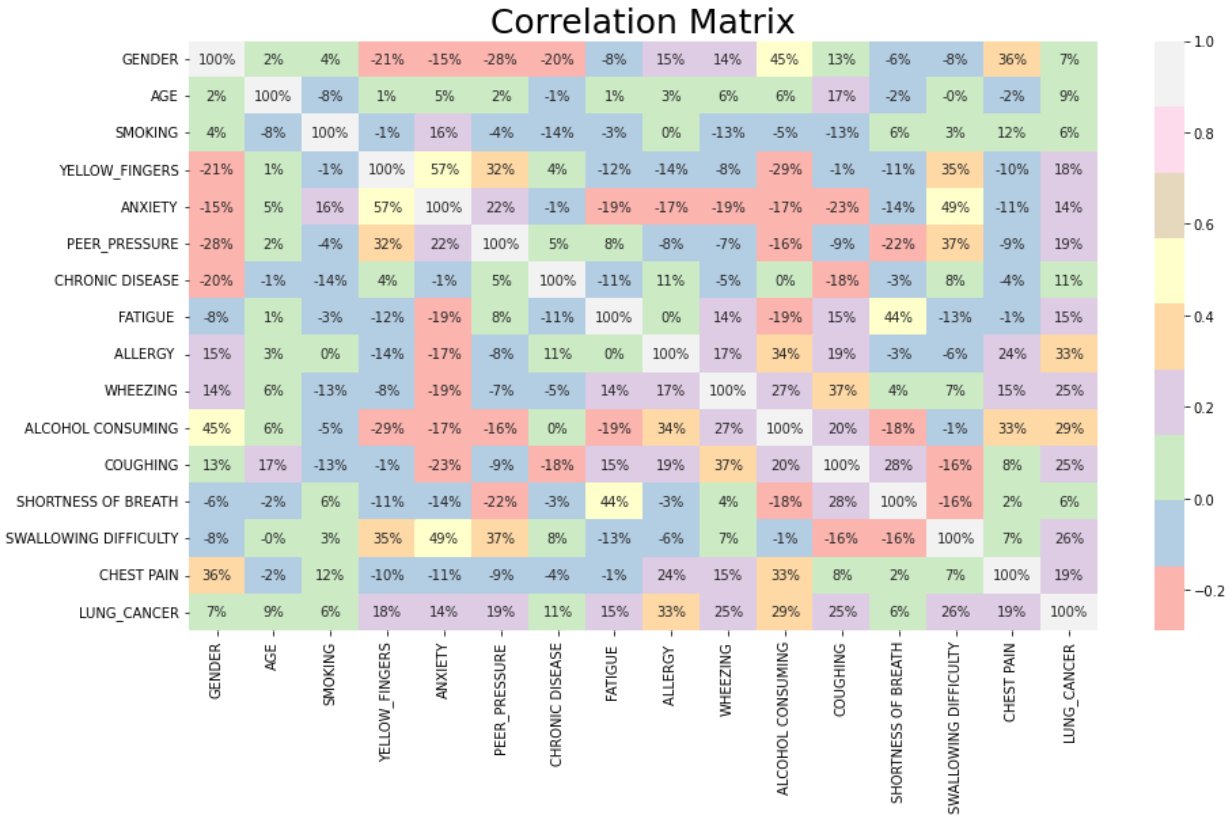
Inference:

- The dark blue squares represent No Lung Cancer or 0 and the light blue squares represent Lung Cancer or 1.
- The scatter plot gives the relation of two attributes one on the x-axis and other on the y.
- Each scatter plot can be studied thoroughly to gain more insights on attributes relationship.

After inferencing the dataset from our graphs, we move towards the mathematical values of correlation between the attributes.

For that, we plot the **Correlation Matrix,** using **DataFrame.corr()** in the **seaborn.heatmap()** function.

```
[9]  plt.figure(figsize=(15,8))
     sns.heatmap(dataset.corr(),annot=True,fmt=".0%",cmap='Pastel1')
     plt.title('Correlation Matrix',size=25)
     plt.show()
```



Data Pre-Processing

● After thoroughly analyzing the dataset through visuals (graph plots), we process our dataset a bit.
● We Display the Counts Of Our Class Variables.
● We apply **LabelEncoder()** to convert our labels("Yes", "No") into numeric form (as '1' and '0').
● This Is Used To Encode The Output Variable "y".

Data pre processing

```
[10] dataset['LUNG_CANCER'].value_counts()

     1    270
     0     39
     Name: LUNG_CANCER, dtype: int64
```

```
[11] labelencoder = LabelEncoder()
     dataset['LUNG_CANCER'] = labelencoder.fit_transform(dataset['LUNG_CANCER'])
```

● Now before applying any classification algorithms, we need to do the following tasks.

- ○ Split the dataset into **X**(input) and **Y**(output) as 2 separate variables.
- ○ Next, we split our X and Y variables into training data and testing data.
- ○ We use the **train_test_split()** function of sklearn.model_selection library to split the dataset.
  (80% training and 20% testing)

We also use **StandardScaler()** from the **sklearn.preprocessing** module to process our data.

StandardScaler() follows the **Standard Normal Distribution (SND).** It makes the mean(mu) = 0 and scales our data to unit variance.

We normalize our data so that it performs better with our classifiers.

```
[11] labelencoder = LabelEncoder()
     dataset['LUNG_CANCER'] = labelencoder.fit_transform(dataset['LUNG_CANCER'])
```

```
[12] X = dataset.drop(['LUNG_CANCER'], axis=1)
     y = dataset.loc[:,['LUNG_CANCER']]
```

```
[13] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 26)
```

```
[14] print("Shape of X_train: ",X_train.shape)
     print("Shape of X_test: ", X_test.shape)
     print("Shape of y_train: ",y_train.shape)
     print("Shape of y_test",y_test.shape)
```

```
Shape of X_train:  (247, 15)
Shape of X_test:  (62, 15)
Shape of y_train:  (247, 1)
Shape of y_test (62, 1)
```

```
[15] from sklearn.preprocessing import StandardScaler
     sc1 = StandardScaler()
     sc2 = StandardScaler()
     X_train = sc1.fit_transform(X_train)
     X_test = sc2.fit_transform(X_test)
```

Algorithm Application :

ML CLASSIFICATION ALGORITHMS

- After getting a clean Dataset, we can now apply our prediction algorithms, to predict whether the person has lung cancer or not.
- In our project, instead of applying just one, we use 5 different classification algorithms to predict the results.
- The motivation to apply all these algorithms was that we wanted to compare their accuracy results to see which algorithm works better on our dataset.

We applied the algorithms given below.

- Logistic regression
- K-Nearest Neighbour
- Decision Tree Classifier
- Random Forest Classifier
- Naive Bayes Classifier
- Support Vector Machine

1. Logistic regression:

Logistic regression is a classification algorithm used to assign a discrete set of categories to observations. Logistic regression transforms the output using the logistic sigmoid function to return a probability value that can then be mapped to two or three distinct groups, unlike linear regression that produces constant number values.

It is possible to forecast different things by logistic regression:

Logistic Regression will assist us in predicting whether the student has passed or failed or not. Predictions of discrete logistic regression are . Probability scores underlying the classifications of the model can also be viewed.

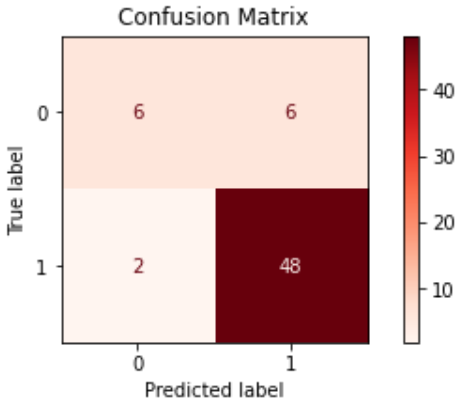
Logistic Regression

```
[16] model1 = LogisticRegression()
     model1.fit(X_train, y_train)
     y_pred = model1.predict(X_test)
```

```
[17] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))
```

```
     Accuracy Score = 87.09677419354838
```

```
[18] class_names = [0,1]
     fig, ax = plt.subplots(figsize=(7,3))
     plot_confusion_matrix(model1, X_test, y_test,cmap=plt.cm.Reds,labels=class_names,ax=ax,values_format = '.0f')
     plt.title('Confusion Matrix')
     plt.grid(False)
     plt.show()
```



It gets an Accuracy of **87.09%**

- Predicted Right-6+48=54
- Predicted Wrong-6+2=8

2. K-Nearest Neighbor (KNN):

K-Nearest Neighbor is one of Machine Learning's most simple and efficient classification algorithms. It belongs to the supervised field of learning and sees extreme use in pattern recognition, data mining and detection of intrusion.

In real-life situations, it is commonly true because it is non-parametric, which means it does not make any underlying conclusions regarding data delivery (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data).

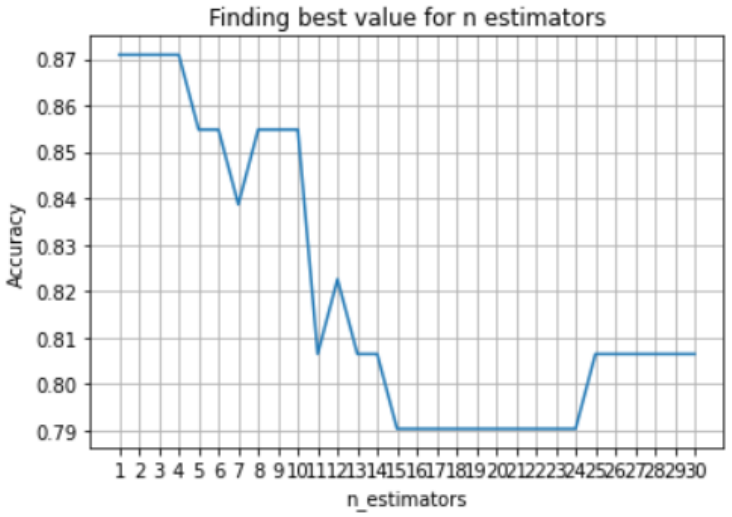
Any prior information (also called training data) is provided to us, which classifies coordinates into attribute-defined classes.

In a case where only two classes are available (e.g., Red/Blue), K can be held as an odd number, so that a simple majority can be calculated. With enhanced K, by different classifications, we get clearer, more defined distinctions. The accuracy of the classifier above also improves if we increase the number of data points in the training set.

Plot N - Neighbors:

KNN Plot N-Neighbors

```
[19] from sklearn import metrics
     from sklearn.neighbors import KNeighborsClassifier
     X_axis= list(range (1, 31))
     acc = pd.Series()
     x = range (1,31)
     for i in list (range (1, 31)):
       knn_model = KNeighborsClassifier (n_neighbors = i)
       knn_model.fit (X_train, y_train)
       prediction = knn_model.predict (X_test)
       acc = acc.append(pd. Series (metrics.accuracy_score (prediction, y_test)))
     plt.plot(X_axis, acc)
     plt.xticks(x)
     plt.title("Finding best value for n estimators")
     plt.xlabel("n_estimators")
     plt.ylabel("Accuracy")
     plt.grid()
     plt.show()
     print('Highest value: ',acc.values.max())
```




```
Highest value:  0.8709677419354839
```

Implementation of Algorithm:

implementation

```
[20] from sklearn.neighbors import KNeighborsClassifier

     knn = KNeighborsClassifier(n_neighbors = 24, metric = 'minkowski', p = 2)

     knn.fit (X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=24)
```

3. Support Vector Machine:

Support-vector machines are supervised learning models that use related learning algorithms, analyze data used for classification and regression analysis. A SVM training algorithm creates a model that assigns new examples to one or the other subclass, each classified as belonging to one or the other of two categories, making it a linear linear conditional classifier that is non-probabilistic.

An SVM model is a representation, mapped in such a way that a simple distance is as wide as possible, of the examples as space points. New cases, based on the side of the gap they land on, are then tracked and calculated to belong to a break.

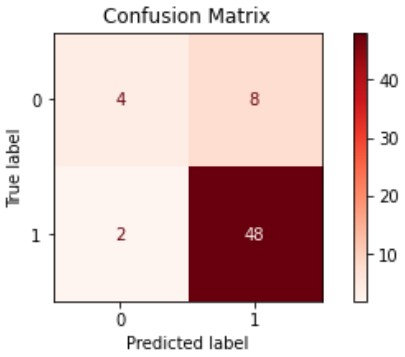
SVM

```
[21] model5 = SVC()
     model5.fit(X_train, y_train)
     y_pred = model5.predict(X_test)
```

```
[22] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))

     Accuracy Score = 83.87096774193549
```

```
[23] class_names = [0,1]
     fig, ax = plt.subplots(figsize=(7,3))
     plot_confusion_matrix(model5, X_test, y_test,cmap=plt.cm.Reds,labels=class_names,ax=ax,values_format = '.0f')
     plt.title('Confusion Matrix')
     plt.grid(False)
     plt.show()
```



It gets an Accuracy of **83.87%**

Predicted Right-4+48=52, Predicted Wrong-8+2=10

4. Naive Bayes:

The Naive Bayes algorithm is an algorithm for supervised learning based on the Bayes theorem and used to overcome classification problems. It is mostly used in text classification, which requires a high-dimensional training dataset. One of the easiest and most powerful classification algorithms is the Naïve Bayes Classifier, which helps to construct fast machine learning models that can predict easily.

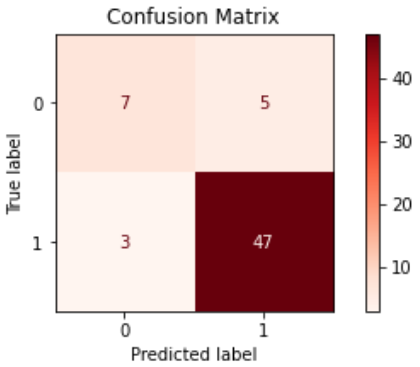
Naive Bayes Classifier

```
[24] model3 = GaussianNB()
     model3.fit(X_train, y_train)
     y_pred = model3.predict(X_test)
```

```
[25] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))

     Accuracy Score = 87.09677419354838
```

```
[26] class_names = [0,1]
     fig, ax = plt.subplots(figsize=(7,3))
     plot_confusion_matrix(model3, X_test, y_test,cmap=plt.cm.Reds,labels=class_names,ax=ax,values_format = '.0f')
     plt.title('Confusion Matrix')
     plt.grid(False)
     plt.show()
```



It gets an Accuracy of **87.09%**

- Predicted Right-7+47=54
- Predicted Wrong-5+3=8

5. Decision Tree:

Decision trees are the most powerful and popular tool for classification and prediction. A Decision tree is a tree-like flowchart where each internal node represents a query for an attribute, each branch represents a test outcome, and each leaf node (terminal node) has a name for the class.

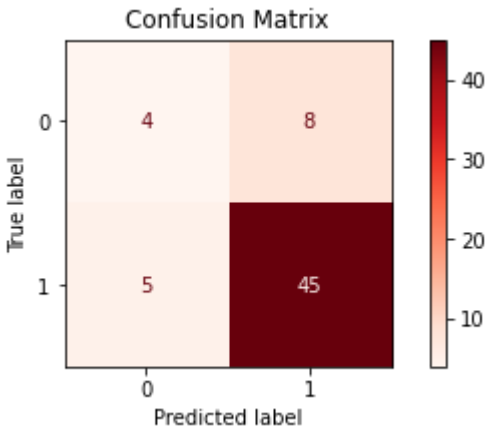
Decision Tree Classifier

```
[27] model2 = DecisionTreeClassifier()
     model2.fit(X_train, y_train)
     y_pred = model2.predict(X_test)
```

```
[28] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))

     Accuracy Score = 80.64516129032258
```

```
[29] class_names = [0,1]
     fig, ax = plt.subplots(figsize=(7,3))
     plot_confusion_matrix(model2, X_test, y_test,cmap=plt.cm.Reds,labels=class_names,ax=ax,values_format = '.0f')
     plt.title('Confusion Matrix')
     plt.grid(False)
     plt.show()
```



It gets an Accuracy of **79.03%**

- Predicted Right-45+5=50
- Predicted Wrong-8+5=13

6. Random Forest Classifier

This classifier follows an ensemble learning. Instead of one, multiple decision trees work as an **“ensemble”**.

It adds randomness to the ensemble by randomly creating a forest of decision trees.

Each decision tree produces an output and the final class for the input record is done by majority voting.

They are found to be more accurate than single decision trees, but have an extended time for training.

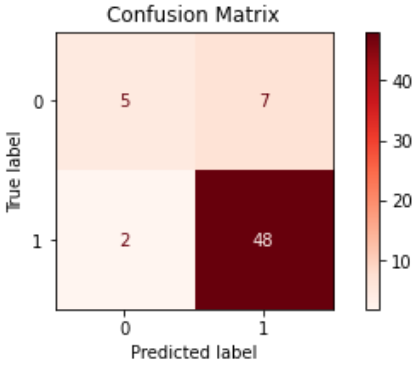
Random Forest

```
[30] model4 = RandomForestClassifier()
     model4.fit(X_train, y_train)
     y_pred = model4.predict(X_test)
```

```
[31] score = accuracy_score(y_test, y_pred)
     print('Accuracy Score = ' + str(score*100))

     Accuracy Score = 85.48387096774194
```

```
[32] class_names = [0,1]
     fig, ax = plt.subplots(figsize=(7,3))
     plot_confusion_matrix(model4, X_test, y_test,cmap=plt.cm.Reds,labels=class_names,ax=ax,values_format = '.0f')
     plt.title('Confusion Matrix')
     plt.grid(False)
     plt.show()
```



It gets an Accuracy of **85.48%**

- Predicted Right-5+48=53
- Predicted Wrong-7+2=9

Fine Tuning this model

Fine Tunning the Model

```
[34] n_estimators = [int(x) for x in np.linspace(start = 10, stop = 100, num = 10)]
     max_features = ['auto', 'sqrt']
     max_depth = [2,4]
     min_samples_split = [2, 5]
     min_samples_leaf = [1, 2]
     param_grid = {'n_estimators': n_estimators,
                   'max_features': max_features,
                   'max_depth': max_depth,
                   'min_samples_split': min_samples_split,
                   'min_samples_leaf': min_samples_leaf
                  }
```

```
[35] model = RandomForestClassifier()
```

Applying Grid Search on Random Forest

```
[36] grid = GridSearchCV(estimator = model, param_grid = param_grid, cv = 3, verbose=2)
     grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 160 candidates, totalling 480 fits
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=20; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=20; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=20; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=30; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=30; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=30; total time=   0.0s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=40; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=40; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=40; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=50; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=50; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=50; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=60; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=60; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=60; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=70; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=70; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=70; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=80; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=80; total time=   0.1s
[CV] END max_depth=2, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=80; total time=   0.2s
```

```
[37] # Getting the Best Parameters for Random Forest
     grid.best_params_
```

```
{'max_depth': 4,
 'max_features': 'auto',
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 10}
```

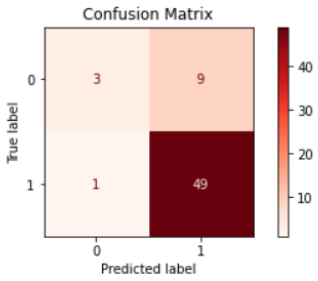
```
[38] # Creating the model with the best params
     finalmodel = RandomForestClassifier(max_depth=4,max_features='auto',min_samples_leaf=2,min_samples_split=5,n_estimators=30)
```

```
[39] finalmodel.fit(X_train,y_train)
     y_pred = finalmodel.predict(X_test)
```

```
[40] score = accuracy_score(y_test, y_pred)
     print('Final Accuracy Score = ' + str(score*100))
```

```
Final Accuracy Score = 83.87096774193549
```

```
[41] class_names = [0,1]
     fig, ax = plt.subplots(figsize=(7,3))
     plot_confusion_matrix(finalmodel, X_test, y_test,cmap=plt.cm.Reds,labels=class_names,ax=ax,values_format = '.0f')
     plt.title('Confusion Matrix')
     plt.grid(False)
     plt.show()
```



Cross Validation Score of the models

```
[42] def CrossValidationScore(model_list):
         global X,y

         mean = []
         modelname = []

         for model in model_list:
             modelname.append(type(model).__name__)

         for i in model_list:
             scores = cross_val_score(i, X, y, cv=5)
             mean.append(scores.mean())

         cvs = pd.DataFrame({"Model Name":modelname,"CVS":mean})
         return cvs.style.background_gradient("Reds")
```

```
[43] model_list=[model1,model2,model3,model4,model5,finalmodel]
```

```
[44] CrossValidationScore(model_list)
```

	Model Name	CVS
0	LogisticRegression	0.915918
1	DecisionTreeClassifier	0.873876
2	GaussianNB	0.903014
3	RandomForestClassifier	0.912745
4	SVC	0.873823
5	RandomForestClassifier	0.877049

CONCLUSION:

From the results, it's clear that **Logistic Regression** gives the highest accuracy of **87.09%** and CVS Score of 0.915918 on our dataset,thus we can conclude using Logistic Regression would be a better choice.And as Logistic Regression is easier to implement,interpret,very fast and can easily extend to multiple classes it would give good results.

Logistic Regression has a problem of overfitting but can be dealt with using Regularization(L1 and L2) technique.

We've uploaded the entire code on Google Collab as well. Link to the code -https://colab.research.google.com/drive/1_MYg4QNut2Sic3rDQ_XUbI343i-EZKE4?usp=sharing#scrollTo=TrbrRPoBXzYM

REFERENCES:

- The class presentations of IDS
- Official documentations of Seaborn, Matplotlib, scikit-learn, Pandas, NumPy
- https://www.kaggle.com/datasets