Lung Cancer Prediction

```
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#For ignoring warning
import warnings
warnings.filterwarnings("ignore")

import cufflinks as cf
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)  #  link pandas to plotly and add the iplot method
```

```
df=pd.read_csv('survey lung cancer.csv')
df
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGU
0	M	69	1	2	2	1	1	
1	M	74	2	1	1	1	2	
2	F	59	1	1	1	2	1	
3	M	63	2	2	2	1	1	
4	F	63	1	2	1	1	1	
...	...	...	...	...	...	...	...	
304	F	56	1	1	1	2	2	
305	M	70	2	1	1	1	1	
306	M	58	2	1	1	1	1	
307	M	67	2	1	2	1	1	
308	M	62	1	1	1	2	1	

309 rows × 16 columns

Note: In this dataset, YES=2 & NO=1

```
df.shape
```

```
(309, 16)
```

```
#Checking for Duplicates
df.duplicated().sum()
```

```
33
```

```
#Removing Duplicates
df=df.drop_duplicates()
```

```
#Checking for null values
df.isnull().sum()
```

```
GENDER                 0
AGE                    0
SMOKING                0
YELLOW_FINGERS         0
ANXIETY                0
PEER_PRESSURE          0
CHRONIC DISEASE        0
FATIGUE                0
```

```
ALLERGY                  0
WHEEZING                 0
ALCOHOL CONSUMING        0
COUGHING                 0
SHORTNESS OF BREATH      0
SWALLOWING DIFFICULTY    0
CHEST PAIN               0
LUNG_CANCER              0
dtype: int64
```

```
df.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    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
dtypes: int64(14), object(2)
memory usage: 38.8+ KB
```

```
df.describe()
```

	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	C
count	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	30
mean	62.673139	1.563107	1.569579	1.498382	1.501618	1.504854	1.673139	1.556634	1.556634	1.556634	1
std	8.210301	0.496806	0.495938	0.500808	0.500808	0.500787	0.469827	0.497588	0.497588	0.497588	0
min	21.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1
25%	57.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1
50%	62.000000	2.000000	2.000000	1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2
75%	69.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2
max	87.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2

Processing data

```
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
df['GENDER']=le.fit_transform(df['GENDER'])
df['LUNG_CANCER']=le.fit_transform(df['LUNG_CANCER'])
df['SMOKING']=le.fit_transform(df['SMOKING'])
df['YELLOW_FINGERS']=le.fit_transform(df['YELLOW_FINGERS'])
df['ANXIETY']=le.fit_transform(df['ANXIETY'])
df['PEER_PRESSURE']=le.fit_transform(df['PEER_PRESSURE'])
df['CHRONIC DISEASE']=le.fit_transform(df['CHRONIC DISEASE'])
df['FATIGUE ']=le.fit_transform(df['FATIGUE '])
df['ALLERGY ']=le.fit_transform(df['ALLERGY '])
df['WHEEZING']=le.fit_transform(df['WHEEZING'])
df['ALCOHOL CONSUMING']=le.fit_transform(df['ALCOHOL CONSUMING'])
df['COUGHING']=le.fit_transform(df['COUGHING'])
df['SHORTNESS OF BREATH']=le.fit_transform(df['SHORTNESS OF BREATH'])
df['SWALLOWING DIFFICULTY']=le.fit_transform(df['SWALLOWING DIFFICULTY'])
```

```
df['CHEST PAIN']=le.fit_transform(df['CHEST PAIN'])
df['LUNG_CANCER']=le.fit_transform(df['LUNG_CANCER'])
```

```
#Let's check what's happened now
df
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SH OF
0	1	69	0	1	1	0	0	1	0	1	1	1	
1	1	74	1	0	0	0	1	1	1	0	0	0	
2	0	59	0	0	0	1	0	1	0	1	0	1	
3	1	63	1	1	1	0	0	0	0	0	1	0	
4	0	63	0	1	0	0	0	0	0	1	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
304	0	56	0	0	0	1	1	1	0	0	1	1	
305	1	70	1	0	0	0	0	1	1	1	1	1	
306	1	58	1	0	0	0	0	0	1	1	1	1	
307	1	67	1	0	1	0	0	1	1	0	1	1	
308	1	62	0	0	0	1	0	1	1	1	1	0	

309 rows × 16 columns

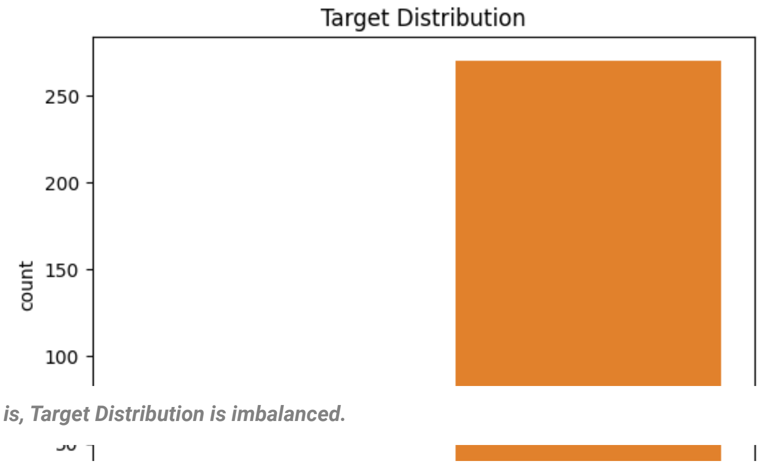
Note: Male=1 & Female=0. Also for other variables, YES=1 & NO=0

```
df.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
```

Data Visualizations

```
#Let's check the distributaion of Target variable.
sns.countplot(x='LUNG_CANCER', data=df,)
plt.title('Target Distribution');
```



That is, Target Distribution is imbalanced.

```
df['LUNG_CANCER'].value_counts()
```

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

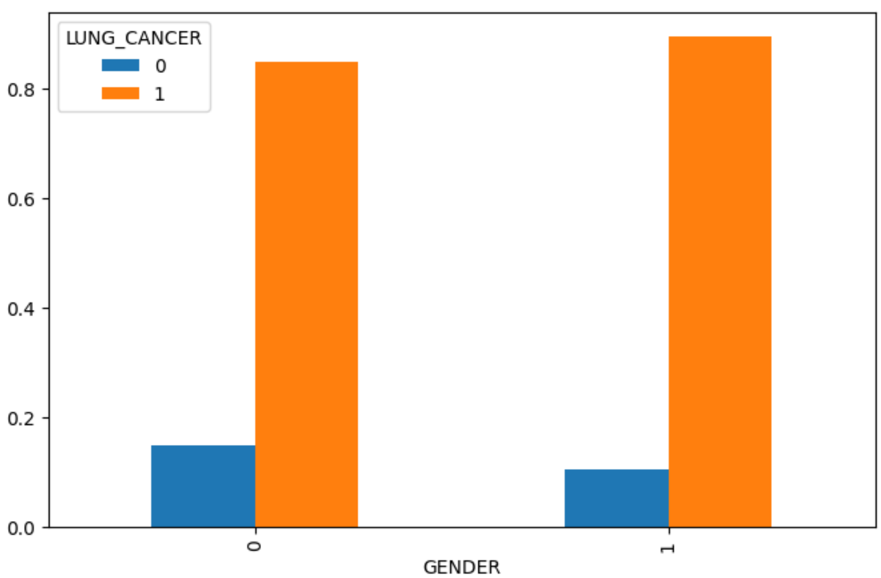
We will handle this imbalance before applyig algorithm.

Now let's do some Data Visualizations for the better understanding of how the independent features are related to the target variable..

```
# function for plotting
def plot(col, df=df):
    return df.groupby(col)['LUNG_CANCER'].value_counts(normalize=True).unstack().plot(kind='bar', figsize=(8,5))
```

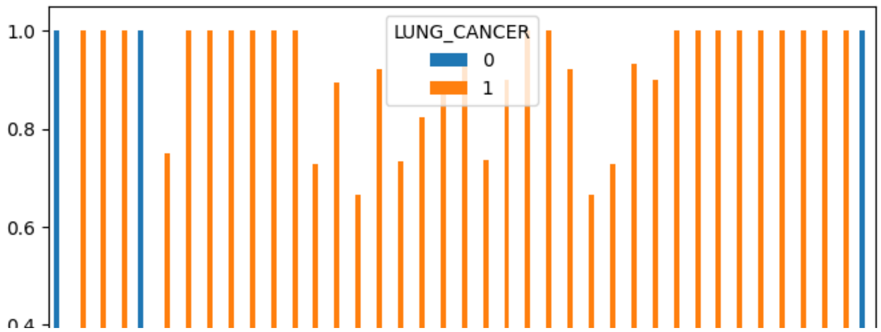
```
plot('GENDER')
```

```
<Axes: xlabel='GENDER'>
```



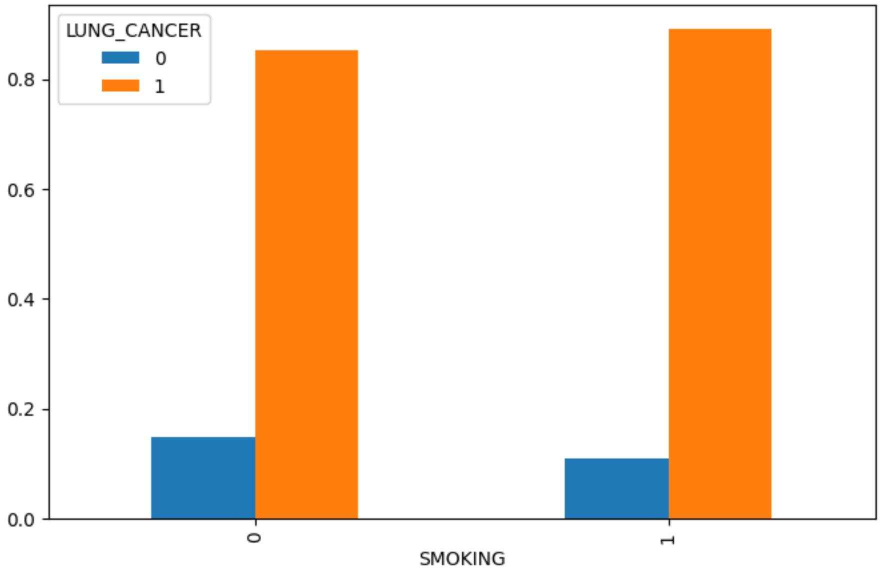

```
plot('AGE')
```

```
<Axes: xlabel='AGE'>
```



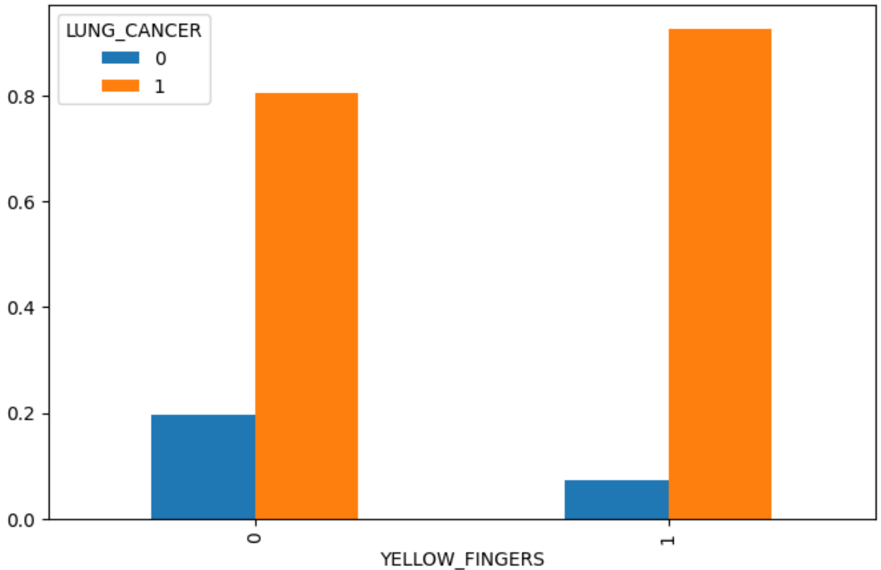

```
plot('SMOKING')
```

```
<Axes: xlabel='SMOKING'>
```



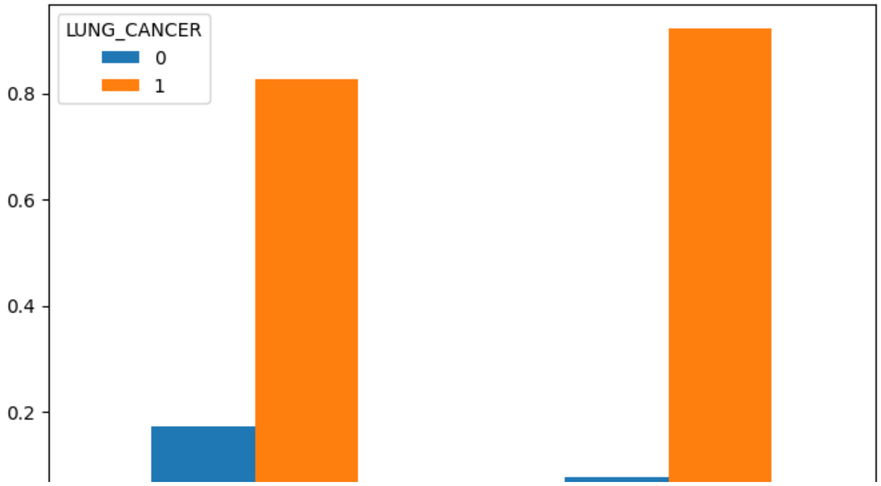

```
plot('YELLOW_FINGERS')
```

```
<Axes: xlabel='YELLOW_FINGERS'>
```



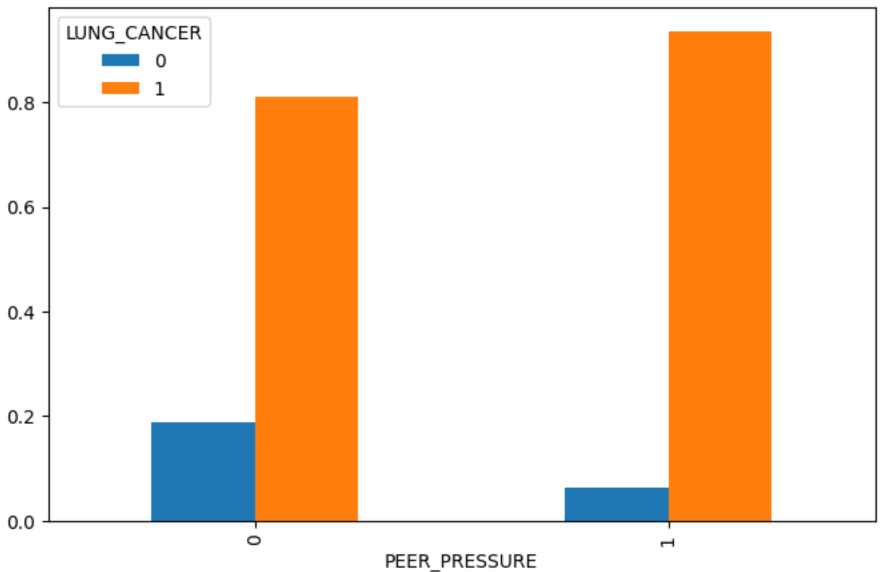

```
plot('ANXIETY')
```

```
<Axes: xlabel='ANXIETY'>
```



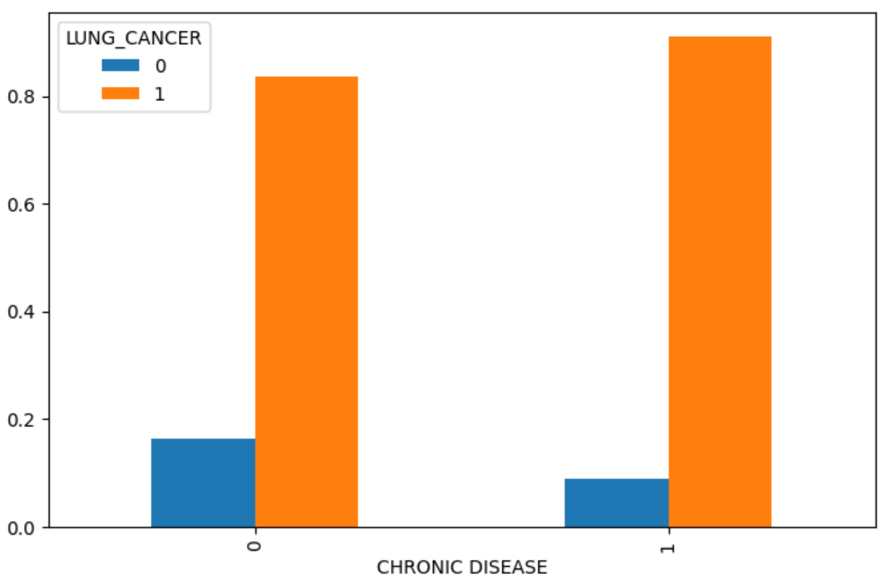

```
plot('PEER_PRESSURE')
```

```
<Axes: xlabel='PEER_PRESSURE'>
```




```
plot('CHRONIC DISEASE')
```

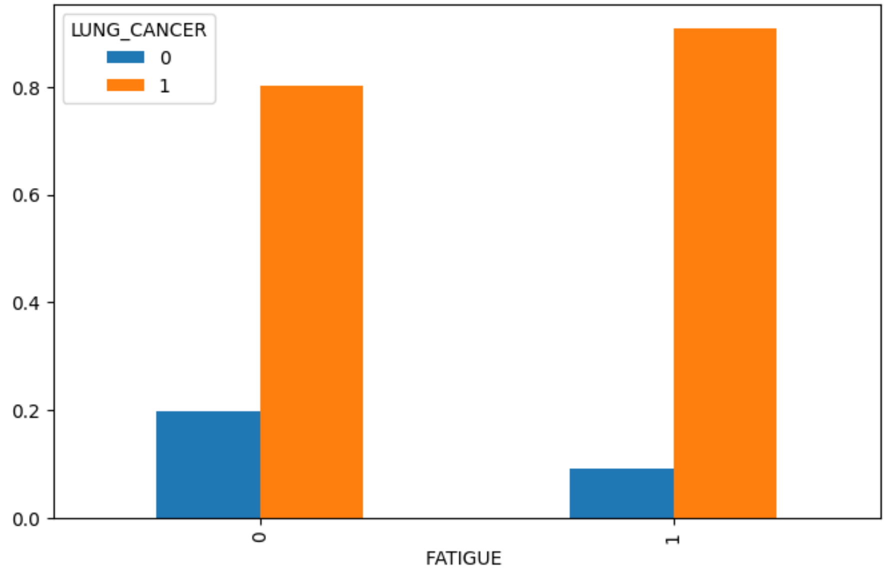
```
<Axes: xlabel='CHRONIC DISEASE'>
```




```
plot('FATIGUE ')
```

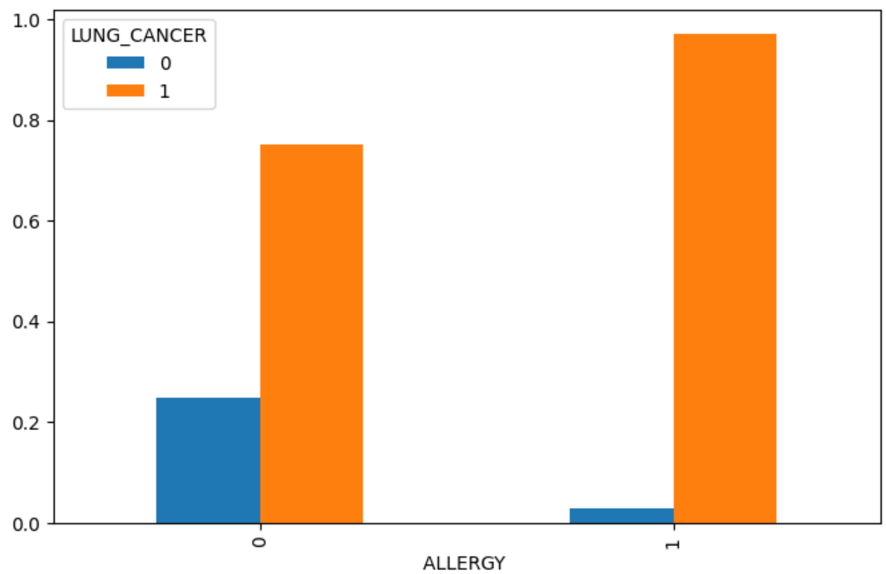
```
plot('FATIGUE ')
```

```
<Axes: xlabel='FATIGUE '>
```




```
plot('ALLERGY ')
```

```
<Axes: xlabel='ALLERGY '>
```

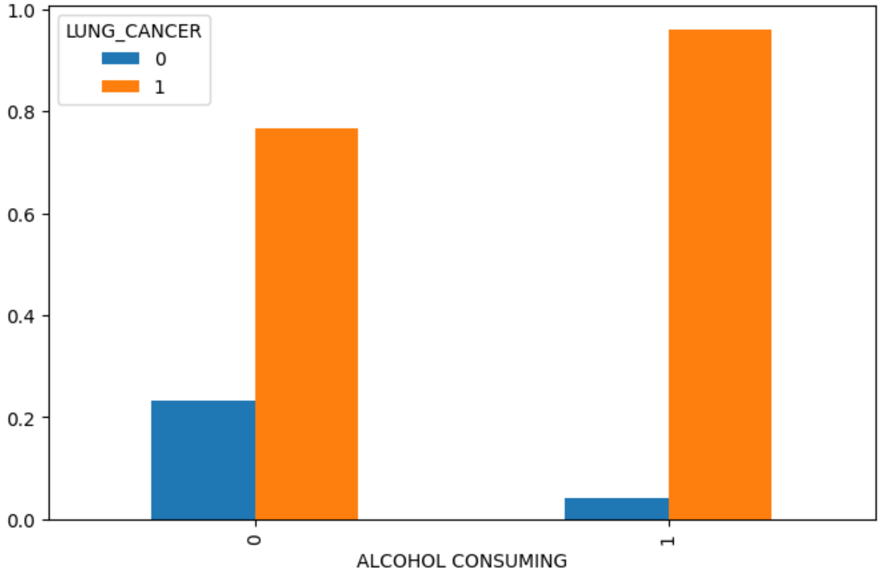



```
plot('WHEEZING')
```

```
<Axes: xlabel='WHEEZING'>
```

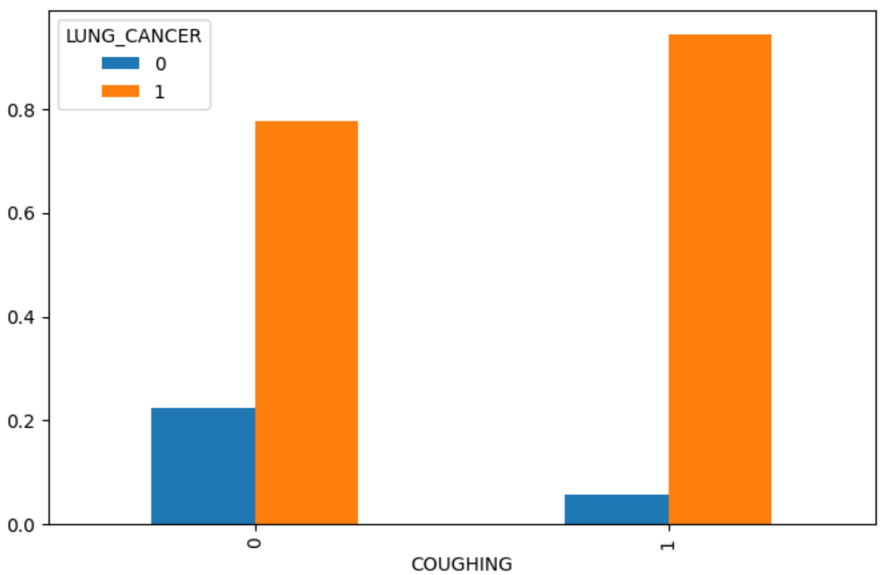

```
plot('ALCOHOL CONSUMING')
```

```
<Axes: xlabel='ALCOHOL CONSUMING'>
```




```
plot('COUGHING')
```

```
<Axes: xlabel='COUGHING'>
```

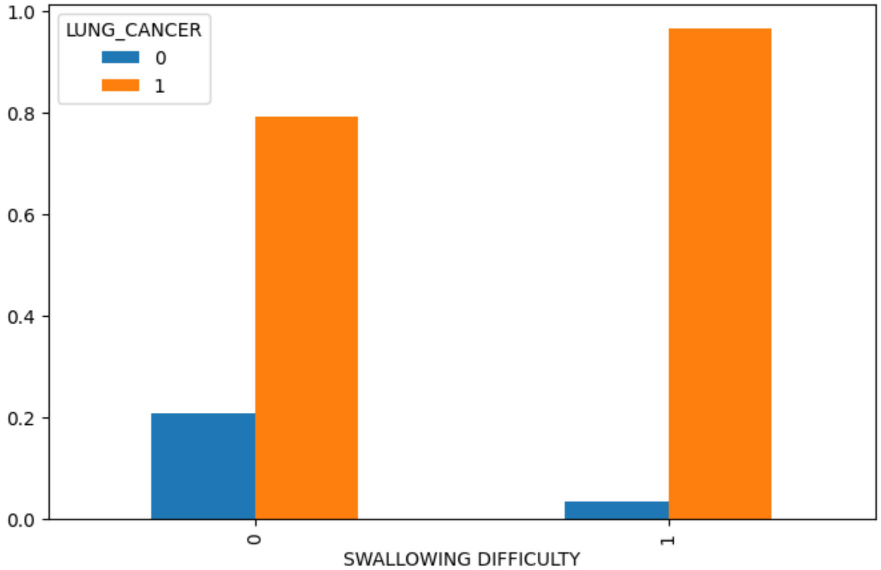



```
plot('SHORTNESS OF BREATH')
```

```
<Axes: xlabel='SHORTNESS OF BREATH'>
```

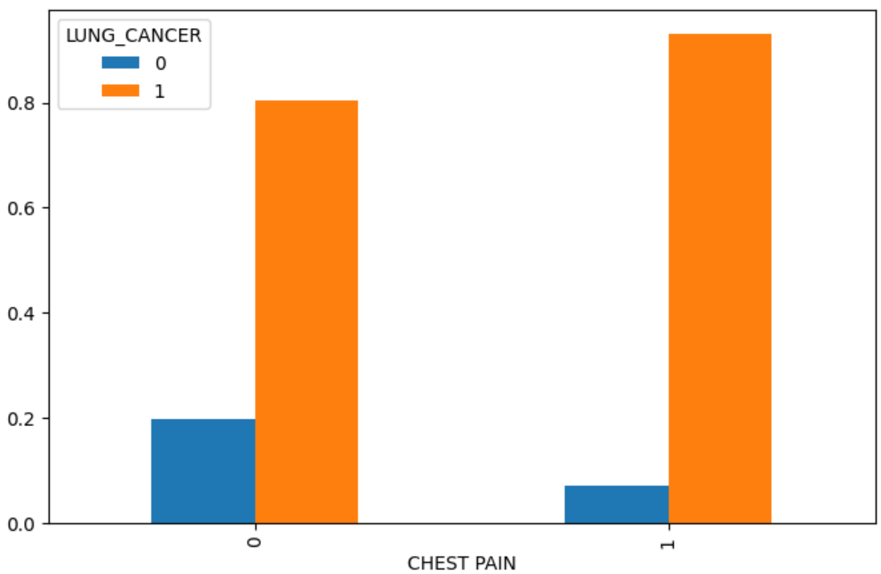

```
plot('SWALLOWING DIFFICULTY')
```

```
<Axes: xlabel='SWALLOWING DIFFICULTY'>
```




```
plot('CHEST PAIN')
```

```
<Axes: xlabel='CHEST PAIN'>
```



From the visualizations, it is clear that in the given dataset, the features GENDER, AGE, SMOKING and SHORTNESS OF BREATH don't have that much relationship with LUNG CANCER. So let's drop those features to make this dataset more clean.

```
df_new=df.drop(columns=['GENDER','AGE', 'SMOKING', 'SHORTNESS OF BREATH'])
df_new
```

	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_
0	1	1	0	0	1	0	1	1	1	1	1	
1	0	0	0	1	1	1	0	0	0	1	1	
2	0	0	1	0	1	0	1	0	1	0	1	
3	1	1	0	0	0	0	0	1	0	1	1	
4	1	0	0	0	0	0	1	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	
304	0	0	1	1	1	0	0	1	1	1	0	
305	0	0	0	0	1	1	1	1	1	0	1	
306	0	0	0	0	0	1	1	1	1	0	1	
307	0	1	0	0	1	1	0	1	1	0	1	
308	0	0	1	0	1	1	1	1	0	1	0	

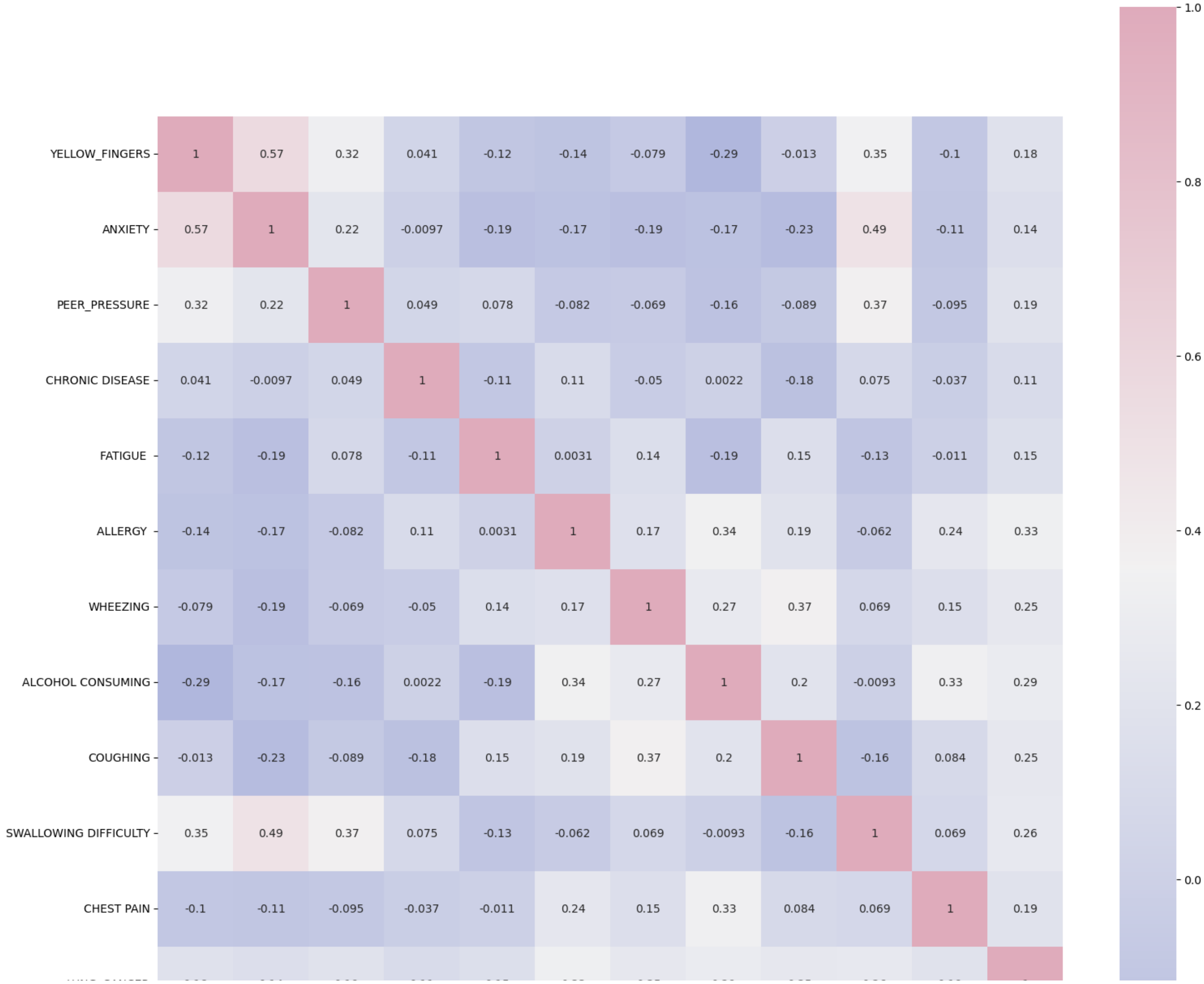
309 rows × 12 columns

Correlation

```
#Finding Correlation
cn=df_new.corr()
cn
```

	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SWALLOWING DIFFICULTY
YELLOW_FINGERS	1.000000	0.565829	0.323083	0.041122	-0.118058	-0.144300	-0.078515	-0.289025	-0.012640	0.345904
ANXIETY	0.565829	1.000000	0.216841	-0.009678	-0.188538	-0.165750	-0.191807	-0.165750	-0.225644	0.489403
PEER_PRESSURE	0.323083	0.216841	1.000000	0.048515	0.078148	-0.081800	-0.068771	-0.159973	-0.089019	0.366590
CHRONIC DISEASE	0.041122	-0.009678	0.048515	1.000000	-0.110529	0.106386	-0.049967	0.002150	-0.175287	0.075176
FATIGUE	-0.118058	-0.188538	0.078148	-0.110529	1.000000	0.003056	0.141937	-0.191377	0.146856	-0.132790
ALLERGY	-0.144300	-0.165750	-0.081800	0.106386	0.003056	1.000000	0.173867	0.344339	0.189524	-0.061508
WHEEZING	-0.078515	-0.191807	-0.068771	-0.049967	0.141937	0.173867	1.000000	0.265659	0.374265	0.069027
ALCOHOL CONSUMING	-0.289025	-0.165750	-0.159973	0.002150	-0.191377	0.344339	0.265659	1.000000	0.202720	-0.009294
COUGHING	-0.012640	-0.225644	-0.089019	-0.175287	0.146856	0.189524	0.374265	0.202720	1.000000	-0.157586
SWALLOWING DIFFICULTY	0.345904	0.489403	0.366590	0.075176	-0.132790	-0.061508	0.069027	-0.009294	-0.157586	1.000000
CHEST PAIN	-0.104829	-0.113634	-0.094828	-0.036938	-0.010832	0.239433	0.147640	0.331226	0.083958	0.069027
LUNG_CANCER	0.181339	0.144947	0.186388	0.110891	0.150673	0.327766	0.249300	0.288533	0.248570	0.259730

```
#Correlation
cmap=sns.diverging_palette(260,-10,s=50, l=75, n=6,
as_cmap=True)
plt.subplots(figsize=(18,18))
sns.heatmap(cn,cmap=cmap,annot=True, square=True)
plt.show()
```




```
kot = cn[cn>=.40]
plt.figure(figsize=(12,8))
sns.heatmap(kot, cmap="Blues")
```

```
<Axes: >
```

Double-click (or enter) to edit

Feature Engineering

```
df_new['ANXYELFIN']=df_new['ANXIETY']*df_new['YELLOW_FINGERS']
df_new
```

	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_
0	1	1	0	0	1	0	1	1	1	1	1	
1	0	0	0	1	1	1	0	0	0	1	1	
2	0	0	1	0	1	0	1	0	1	0	1	
3	1	1	0	0	0	0	0	1	0	1	1	
4	1	0	0	0	0	0	1	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	
304	0	0	1	1	1	0	0	1	1	1	0	
305	0	0	0	0	1	1	1	1	1	0	1	
306	0	0	0	0	0	1	1	1	1	0	1	
307	0	1	0	0	1	1	0	1	1	0	1	
308	0	0	1	0	1	1	1	1	0	1	0	

309 rows × 13 columns

```
#Splitting independent and dependent variables
X = df_new.drop('LUNG_CANCER', axis = 1)
y = df_new['LUNG_CANCER']
```

Target Distribution Imbalance Handling

```
from imblearn.over_sampling import ADASYN
adasyn = ADASYN(random_state=42)
X, y = adasyn.fit_resample(X, y)
```

```
len(X)
```

```
545
```

Model building

Logistic Regression

```
#Splitting data for training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size= 0.25, random_state=0)
```

```
#Fitting training data to the model
from sklearn.linear_model import LogisticRegression
lr_model=LogisticRegression(random_state=0)
lr_model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0)
```

```
#Predicting result using testing data
y_lr_pred= lr_model.predict(X_test)
y_lr_pred
```

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

```
#Model accuracy
from sklearn.metrics import classification_report, accuracy_score, f1_score
lr_cr=classification_report(y_test, y_lr_pred)
print(lr_cr)
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.95        74
           1       0.95      0.94      0.94        63

    accuracy                           0.95       137
   macro avg       0.95      0.95      0.95       137
weighted avg       0.95      0.95      0.95       137
```

This model is almost 97% accurate.

Decision Tree

```
#Fitting training data to the model
from sklearn.tree import DecisionTreeClassifier
dt_model= DecisionTreeClassifier(criterion='entropy', random_state=0)
dt_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
#Predicting result using testing data
y_dt_pred= dt_model.predict(X_test)
y_dt_pred
```

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

```
#Model accuracy
dt_cr=classification_report(y_test, y_dt_pred)
```

```
print(dt_cr)
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93        74
           1       0.95      0.89      0.92        63

    accuracy                           0.93       137
   macro avg       0.93      0.92      0.93       137
weighted avg       0.93      0.93      0.93       137
```

This model is 94% accurate.

K Nearest Neighbor

```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
knn_model= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
knn_model.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
#Predicting result using testing data
y_knn_pred= knn_model.predict(X_test)
y_knn_pred
```

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

```
#Model accuracy
knn_cr=classification_report(y_test, y_knn_pred)
print(knn_cr)
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92        74
           1       0.96      0.84      0.90        63

    accuracy                           0.91       137
   macro avg       0.92      0.91      0.91       137
weighted avg       0.92      0.91      0.91       137
```

This model is 96% accurate.

Gaussian Naive Bayes

```
#Fitting Gaussian Naive Bayes classifier to the training set
from sklearn.naive_bayes import GaussianNB
gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
#Predicting result using testing data
y_gnb_pred= gnb_model.predict(X_test)
y_gnb_pred
```

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

```
#Model accuracy
gnb_cr=classification_report(y_test, y_gnb_pred)
print(gnb_cr)
```

```
              precision    recall  f1-score   support

           0       0.97      0.77      0.86        74
           1       0.78      0.97      0.87        63

    accuracy                           0.86       137
   macro avg       0.87      0.87      0.86       137
weighted avg       0.88      0.86      0.86       137
```

This model is 92% accurate.

Cross Validation

```
# K-Fold Cross Validation

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

k = 10
kf = KFold(n_splits=k, shuffle=True, random_state=42)


# Logistic regerssion model
lr_model_scores = cross_val_score(lr_model,X, y, cv=kf)

# Decision tree model
dt_model_scores = cross_val_score(dt_model,X, y, cv=kf)

# KNN model
knn_model_scores = cross_val_score(knn_model,X, y, cv=kf)

# Gaussian naive bayes model
gnb_model_scores = cross_val_score(gnb_model,X, y, cv=kf)

print("Logistic regression models' average accuracy:", np.mean(lr_model_scores))
print("Decision tree models' average accuracy:", np.mean(dt_model_scores))
print("KNN models' average accuracy:", np.mean(knn_model_scores))
print("Gaussian naive bayes models' average accuracy:", np.mean(gnb_model_scores))
```

```
Logistic regression models' average accuracy: 0.9449831649831649
Decision tree models' average accuracy: 0.943131313131313
KNN models' average accuracy: 0.8932323232323233
Gaussian naive bayes models' average accuracy: 0.9044781144781144
```

So the K-Fold cross validation is showing Decision Tree model gives the most accuracy of 95.4%, and also gives almost same accuracy, while Gaussian naive bayes model gives the least accuarcy of 88.45%.