Lung Cancer Prediction Using 10 models

Lung cancer prediction using 10 machine learning classification models using Scikit-learn library in Python is a code implementation that aims to develop a predictive model for detecting lung cancer in patients. The code uses 10 different machine learning algorithms, including logistic regression, decision tree, k-nearest neighbor, Gaussian naive Bayes, multinomial naive Bayes, support vector classifier, random forest, XGBoost, multi-layer perceptron, and gradient boosting classifier, to predict the likelihood of lung cancer based on a range of variables. The dataset used in the code includes various columns such as gender, age, smoking, yellow fingers, anxiety, peer pressure, chronic disease, fatigue, allergy, wheezing, alcohol consuming, coughing, shortness of breath, swallowing difficulty, chest pain, and lung cancer. By analyzing these variables and using machine learning algorithms to identify patterns and correlations, the predictive models can provide accurate assessments of a patient's risk of developing lung cancer.

```
#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")


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

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIG
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()
```

```
np.int64(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'>
Index: 276 entries, 0 to 283
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 276 non-null    object
 1   AGE                    276 non-null    int64
 2   SMOKING                276 non-null    int64
 3   YELLOW_FINGERS         276 non-null    int64
 4   ANXIETY                276 non-null    int64
 5   PEER_PRESSURE          276 non-null    int64
 6   CHRONIC DISEASE        276 non-null    int64
 7   FATIGUE                276 non-null    int64
 8   ALLERGY                276 non-null    int64
 9   WHEEZING               276 non-null    int64
 10  ALCOHOL CONSUMING      276 non-null    int64
 11  COUGHING               276 non-null    int64
 12  SHORTNESS OF BREATH    276 non-null    int64
 13  SWALLOWING DIFFICULTY  276 non-null    int64
 14  CHEST PAIN             276 non-null    int64
 15  LUNG_CANCER            276 non-null    object
dtypes: int64(14), object(2)
memory usage: 36.7+ KB
```

```
df.describe()
```

	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE
count	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000
mean	62.909420	1.543478	1.576087	1.496377	1.507246	1.521739
std	8.379355	0.499011	0.495075	0.500895	0.500856	0.500435
min	21.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	57.750000	1.000000	1.000000	1.000000	1.000000	1.000000
50%	62.500000	2.000000	2.000000	1.000000	2.000000	2.000000
75%	69.000000	2.000000	2.000000	2.000000	2.000000	2.000000
max	87.000000	2.000000	2.000000	2.000000	2.000000	2.000000

In this dataset, GENDER & LUNG_CANCER attributes are in object data type. So, let's convert them to numerical values using LabelEncoder from sklearn. LabelEncoder is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1. It can also

be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels. Also let's make every other attributes as YES=1 & NO=0.

```
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	FATIG
0	1	69	0	1	1	0	0	
1	1	74	1	0	0	0	1	
2	0	59	0	0	0	1	0	
3	1	63	1	1	1	0	0	
4	0	63	0	1	0	0	0	
...	...	...	...	...	...	...	...	
279	0	59	0	1	1	1	0	
280	0	59	1	0	0	0	1	
281	1	55	1	0	0	0	0	
282	1	46	0	1	1	0	0	
283	1	60	0	1	1	0	0	

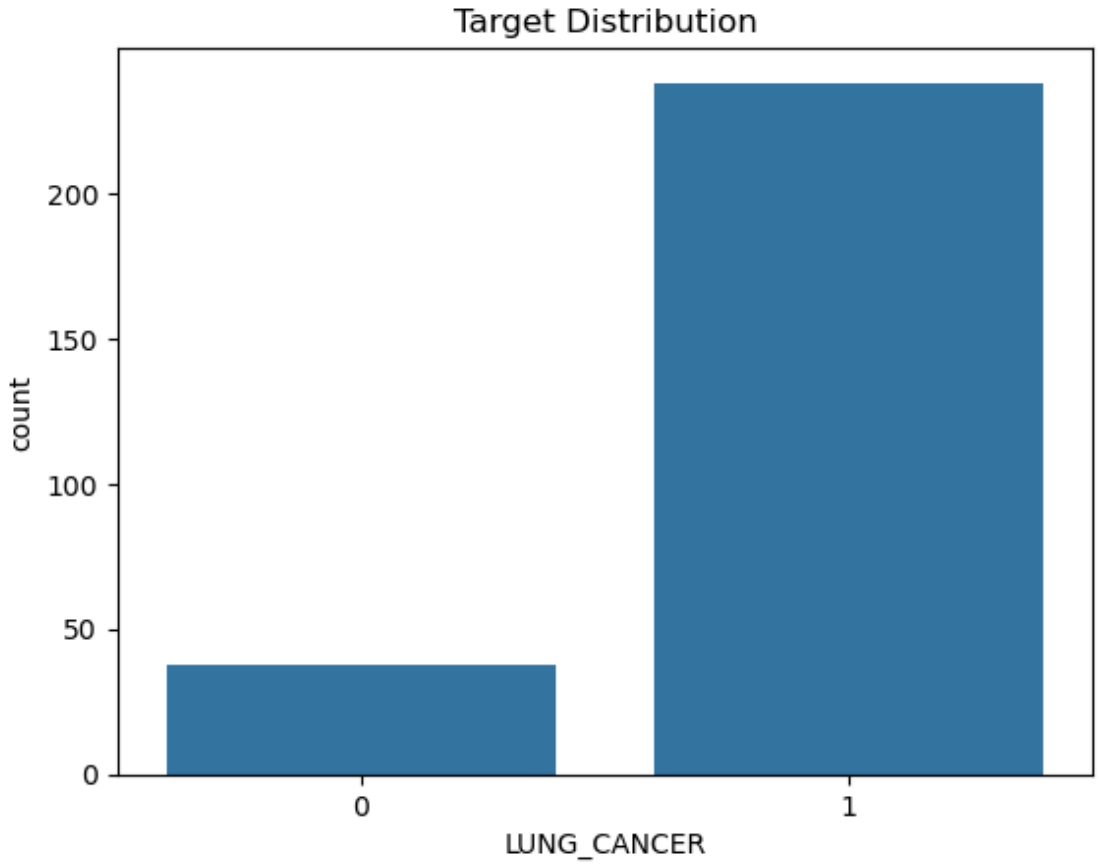
276 rows × 16 columns

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 276 entries, 0 to 283
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 276 non-null    int64
 1   AGE                    276 non-null    int64
 2   SMOKING                276 non-null    int64
 3   YELLOW_FINGERS         276 non-null    int64
 4   ANXIETY                276 non-null    int64
 5   PEER_PRESSURE          276 non-null    int64
 6   CHRONIC DISEASE        276 non-null    int64
 7   FATIGUE                276 non-null    int64
 8   ALLERGY                276 non-null    int64
 9   WHEEZING               276 non-null    int64
 10  ALCOHOL CONSUMING      276 non-null    int64
 11  COUGHING               276 non-null    int64
 12  SHORTNESS OF BREATH    276 non-null    int64
 13  SWALLOWING DIFFICULTY  276 non-null    int64
 14  CHEST PAIN             276 non-null    int64
 15  LUNG_CANCER            276 non-null    int64
dtypes: int64(16)
memory usage: 36.7 KB
```

```
#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()
```

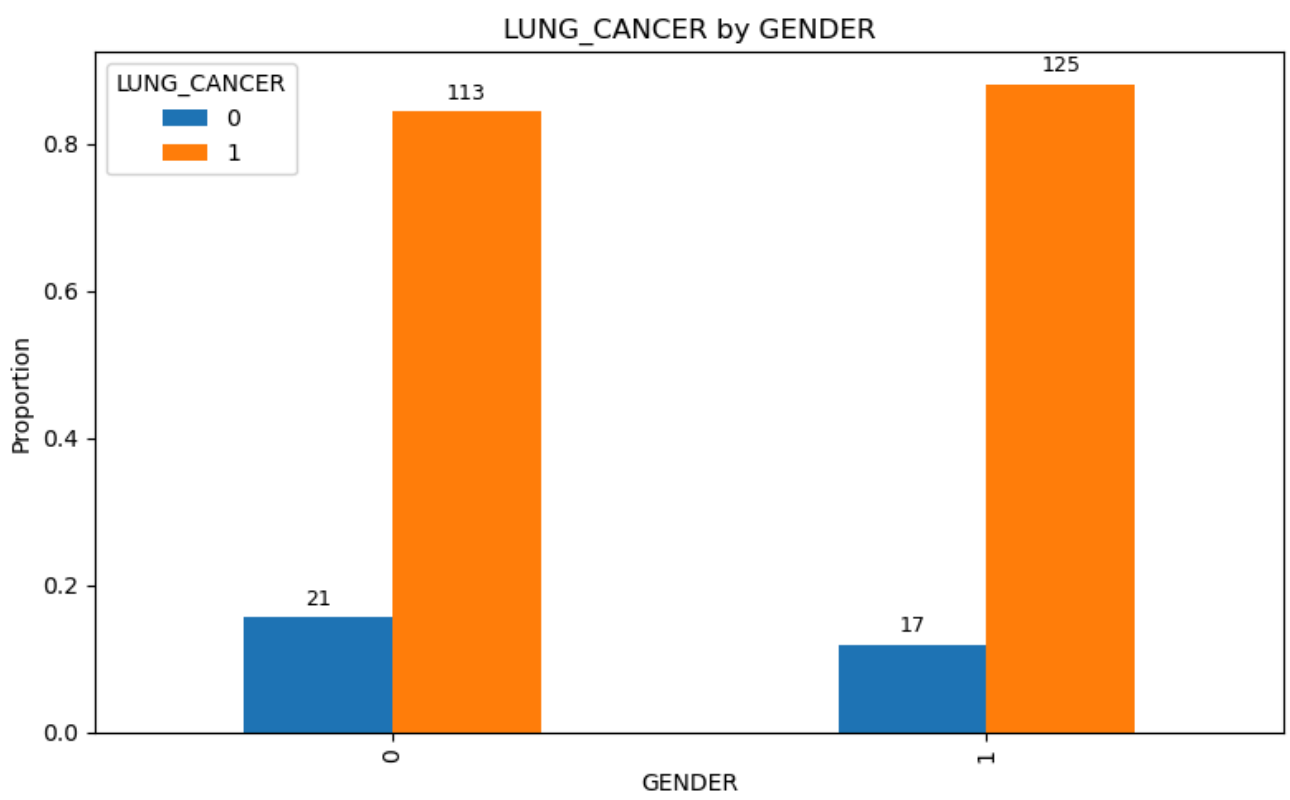
```
LUNG_CANCER
1    238
0     38
Name: count, dtype: int64
```

We will handle this imbalance before applyig algorithm.

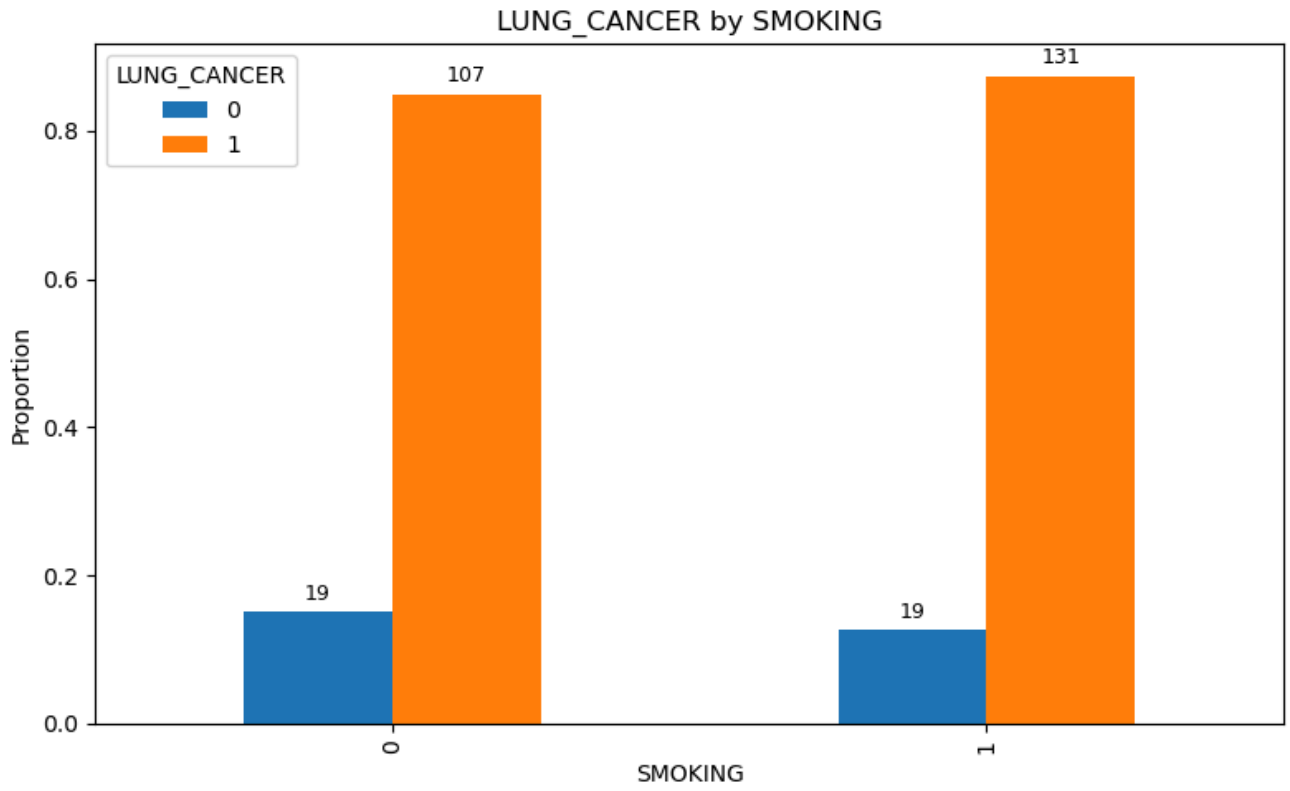
Double-click (or enter) to edit

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

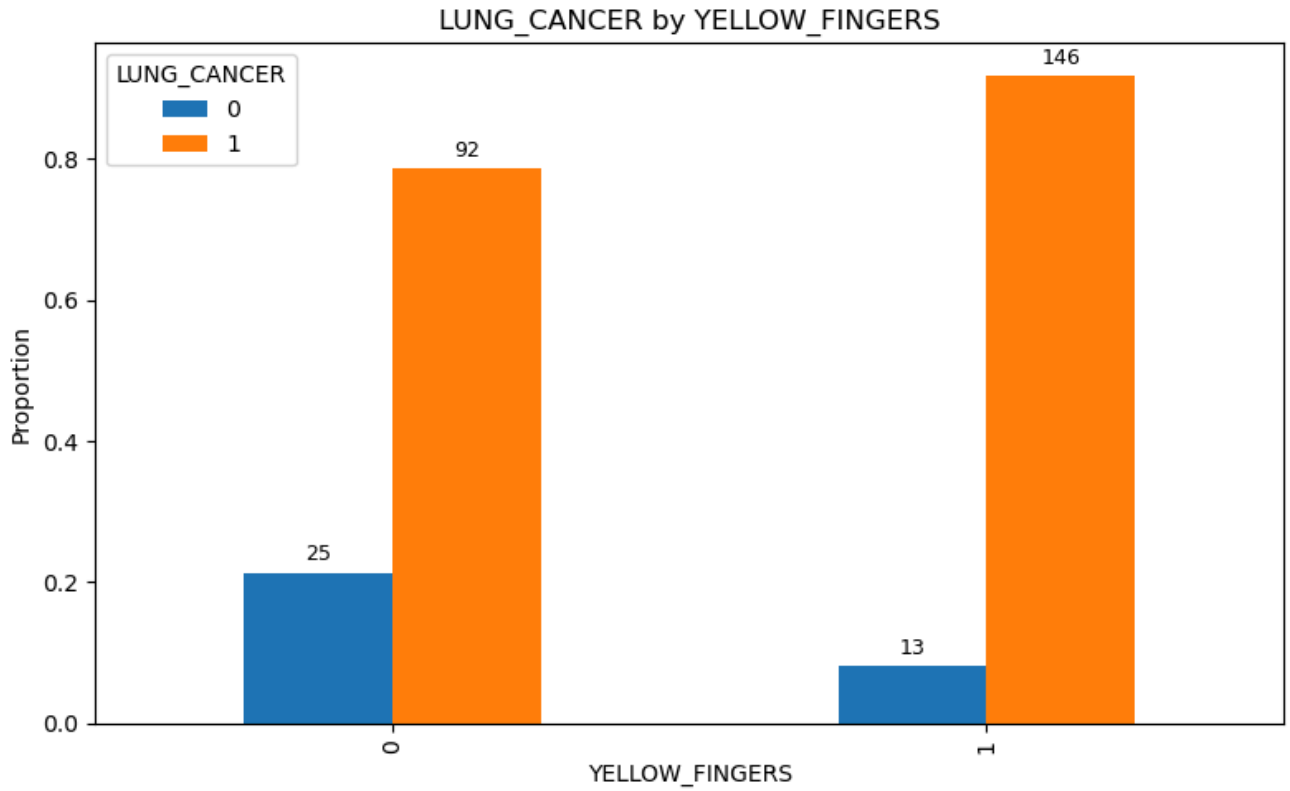
```
import matplotlib.pyplot as plt

def plot(col, df=df):
    # Proportion data for plotting
    prop = df.groupby(col)['LUNG_CANCER'].value_counts(normalize=True).unstack()
    # Actual counts for labeling
    counts = df.groupby(col)['LUNG_CANCER'].value_counts().unstack()

    ax = prop.plot(kind='bar', figsize=(8, 5))

    # Add count labels
    for bars, count_row in zip(ax.containers, counts.T.values):
        for bar, count in zip(bars, count_row):
            ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
                    str(int(count)), ha='center', va='bottom', fontsize=9)

    plt.title(f'LUNG_CANCER by {col}')
    plt.ylabel('Proportion')
    plt.xlabel(col)
    plt.legend(title='LUNG_CANCER')
    plt.tight_layout()
    plt.show()



plot('GENDER')
```



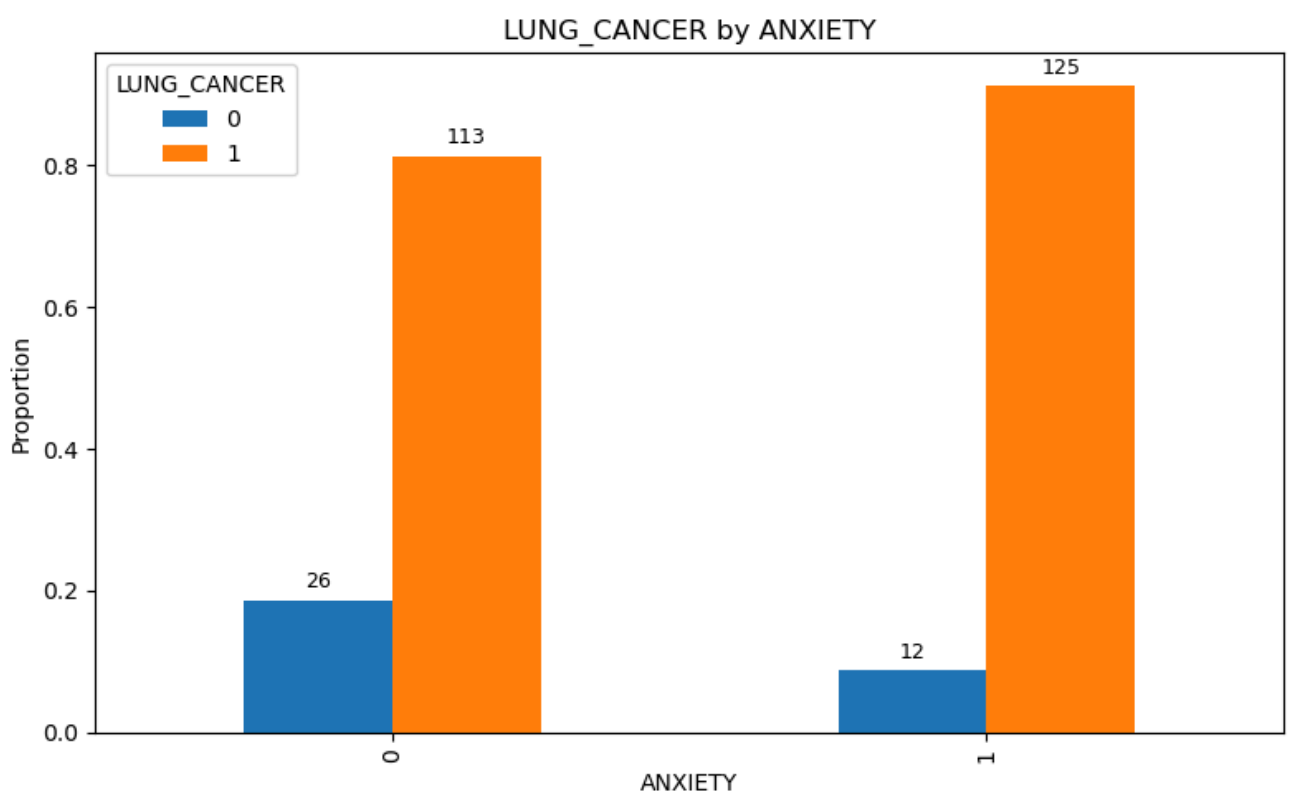

```
plot('SMOKING')
```



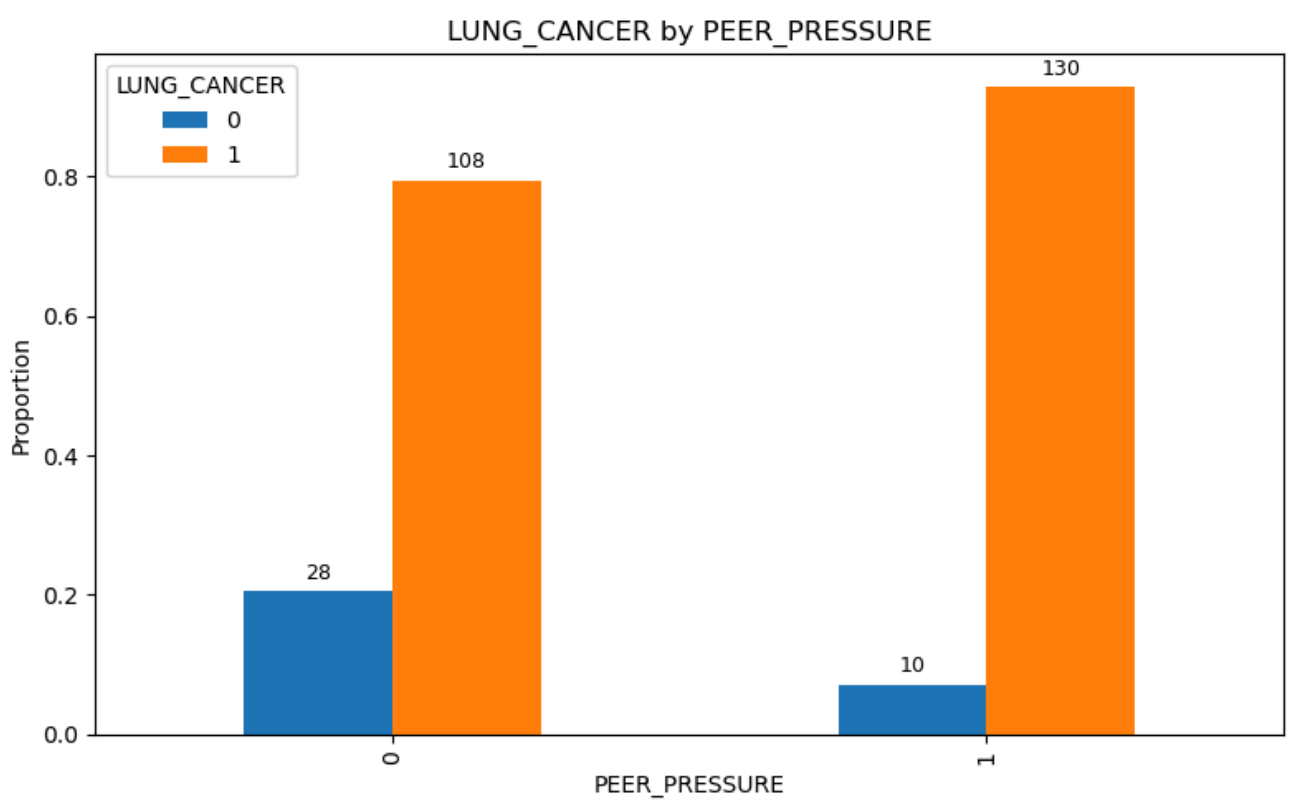

```
plot('YELLOW_FINGERS')
```



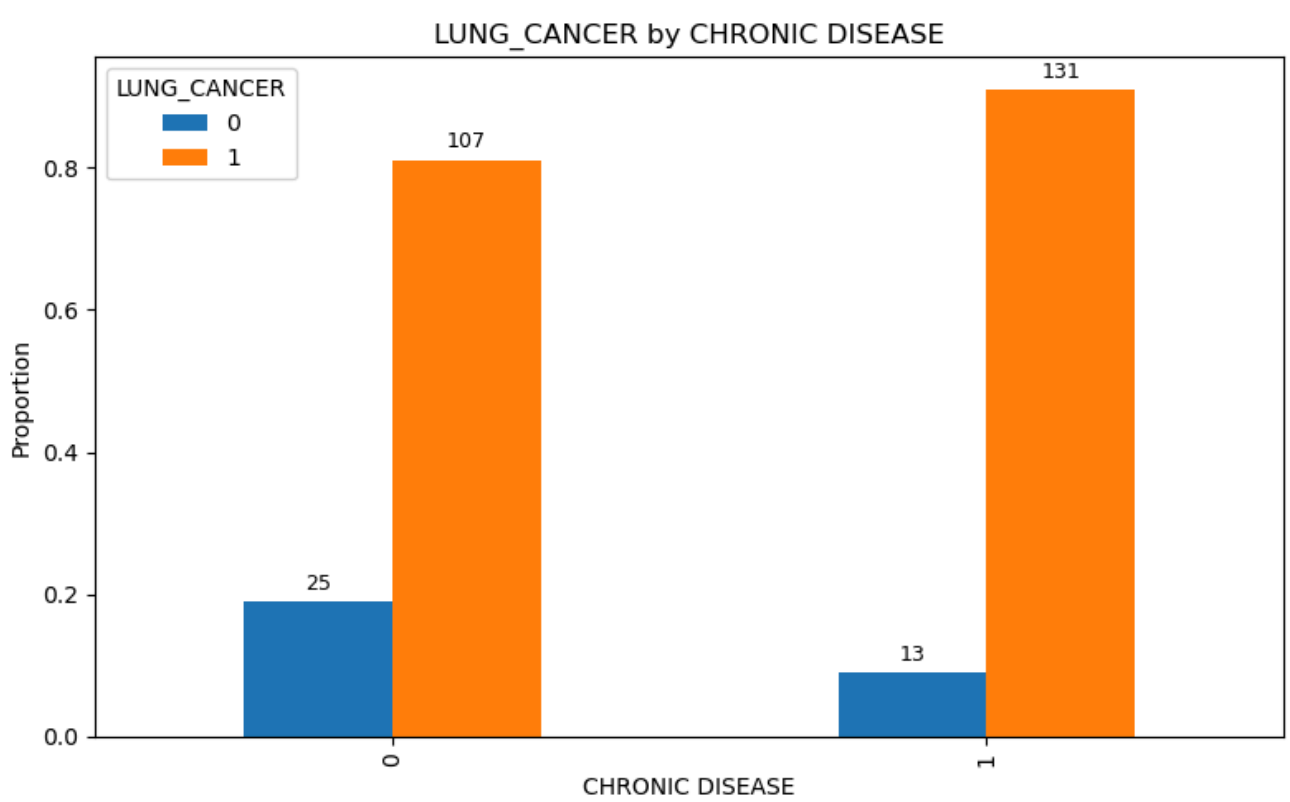

```
plot('ANXIETY')
```



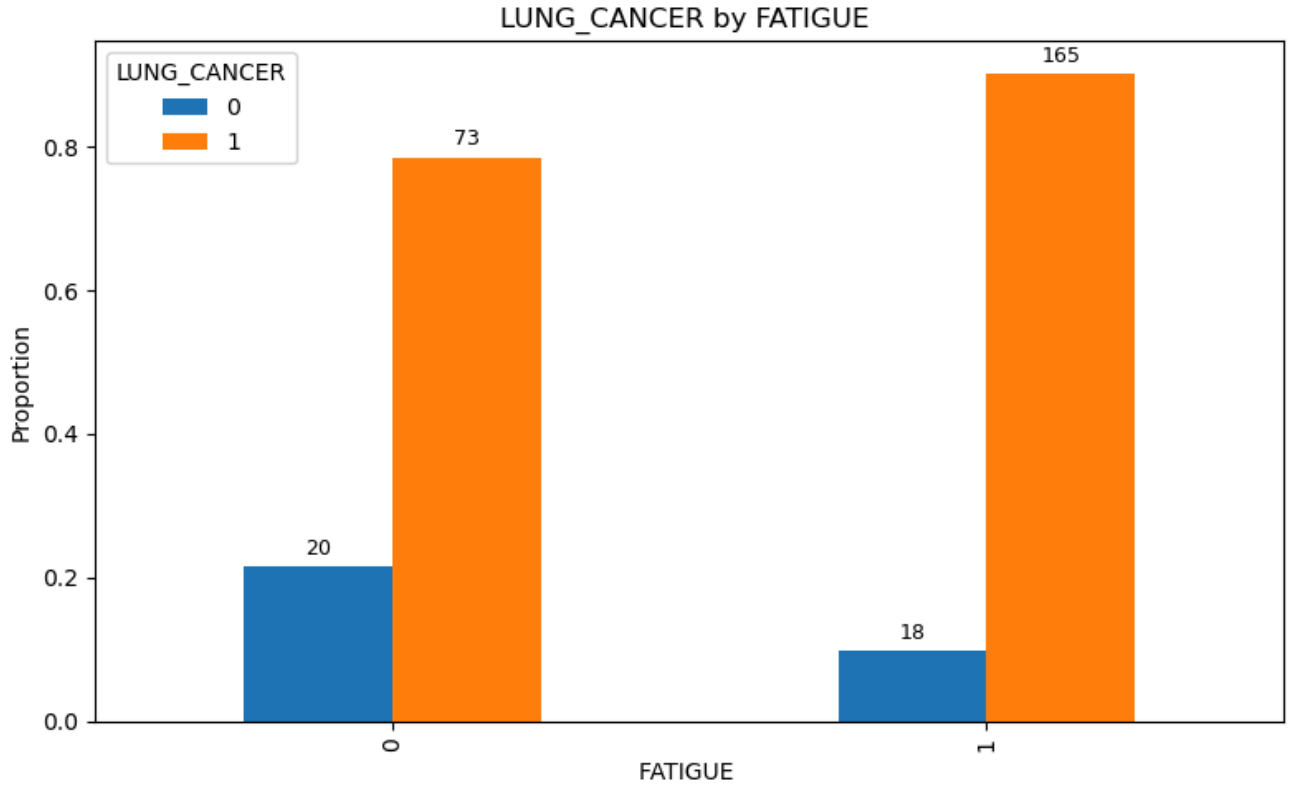

```
plot('PEER_PRESSURE')
```



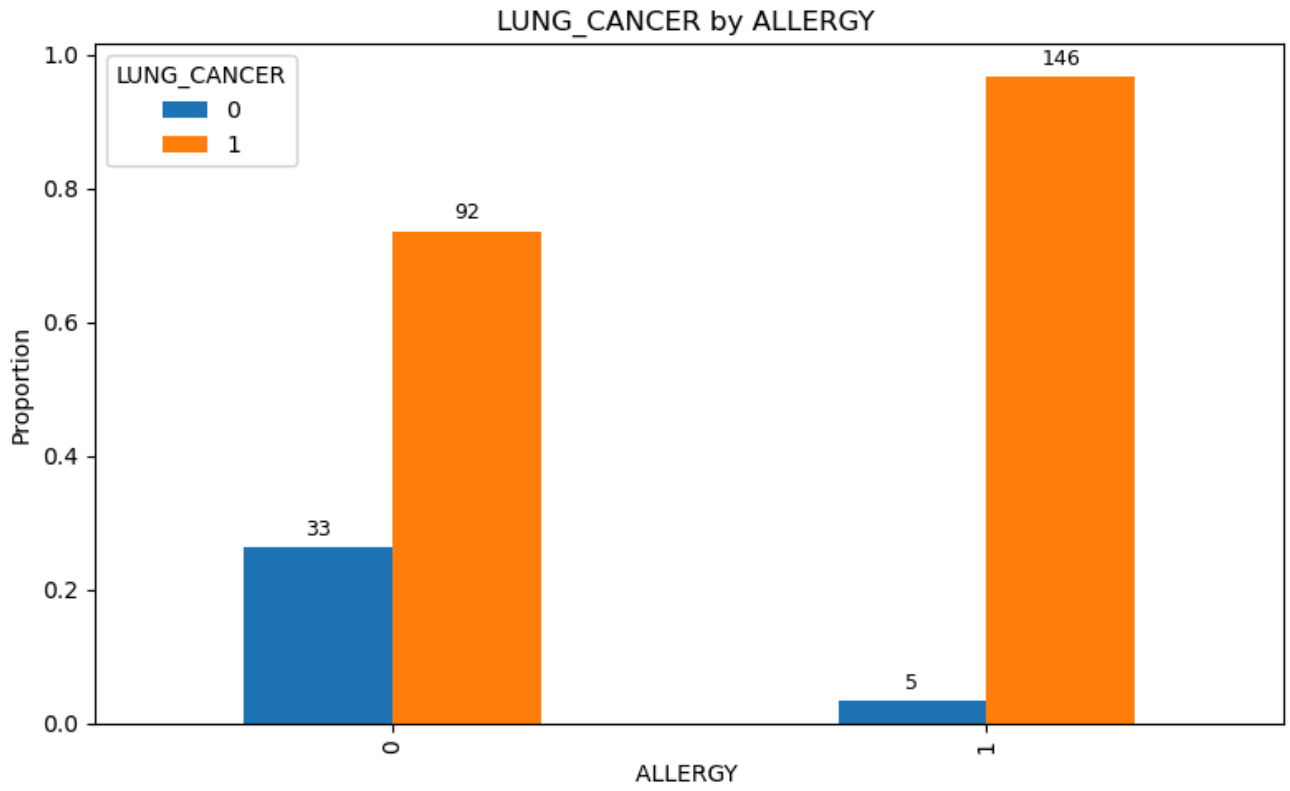

```
plot('CHRONIC DISEASE')
```



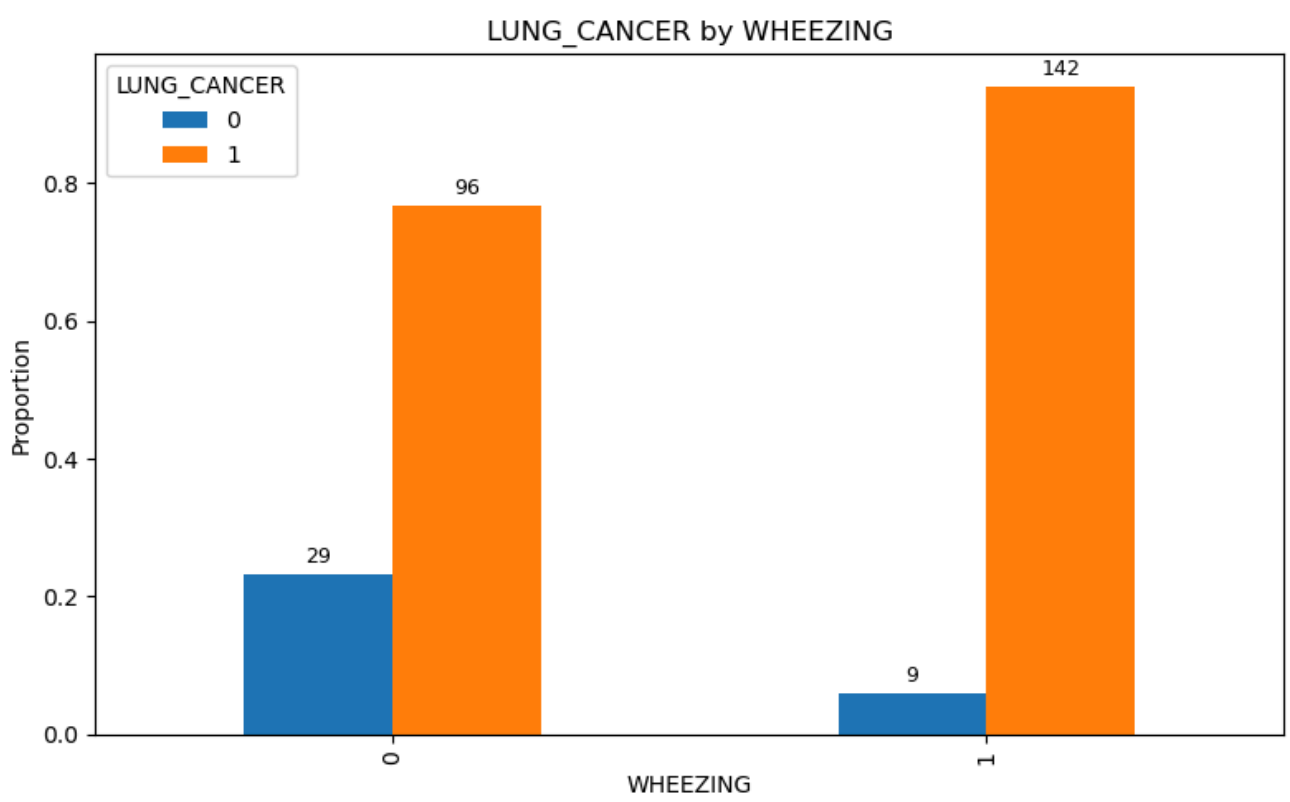

```
plot('FATIGUE ')
```



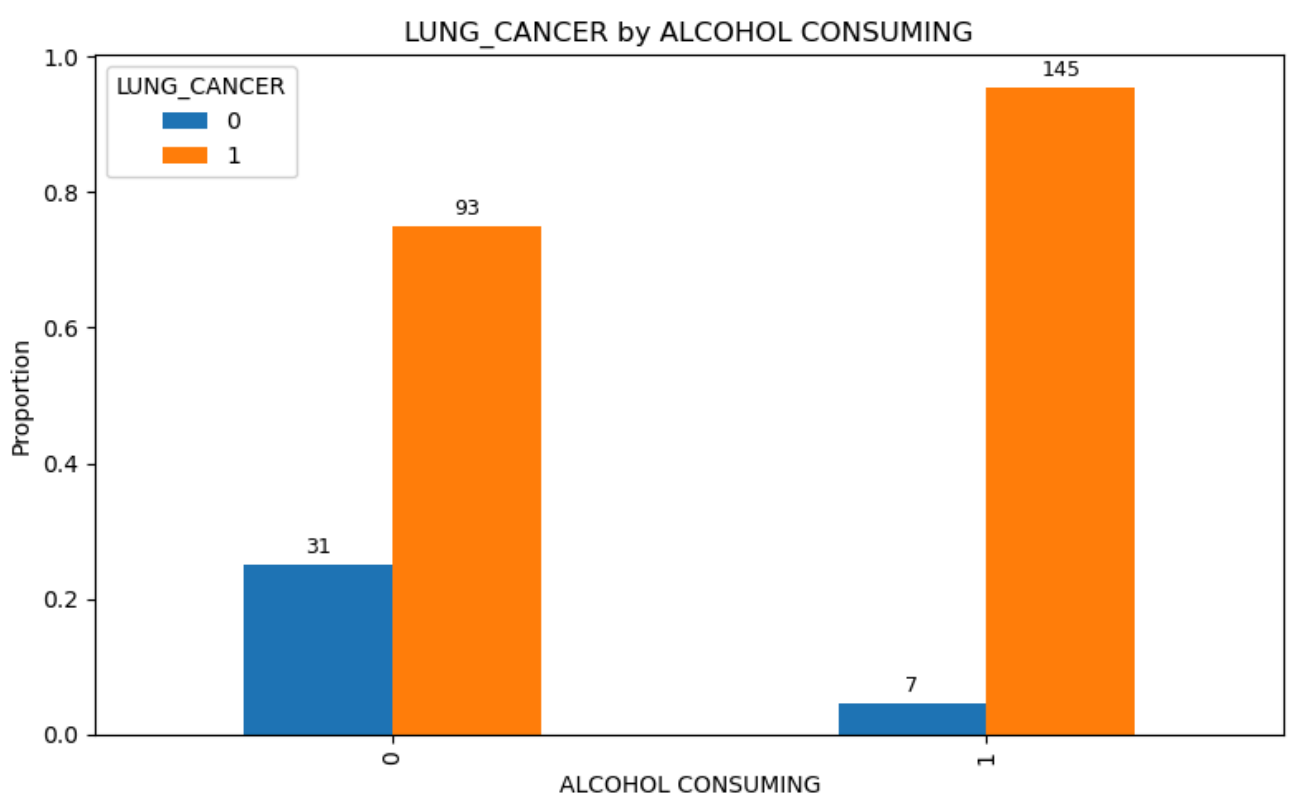

```
plot('ALLERGY ')
```



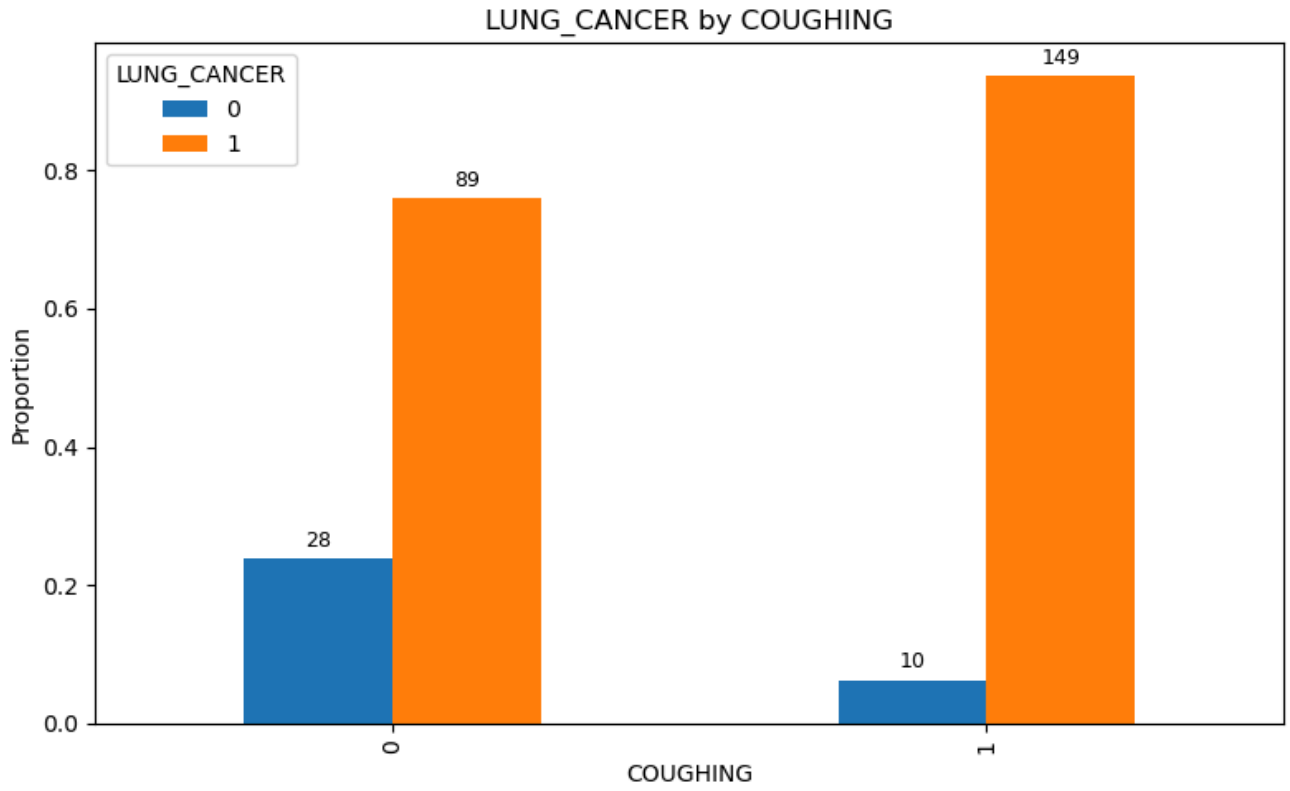

```
plot('WHEEZING')
```



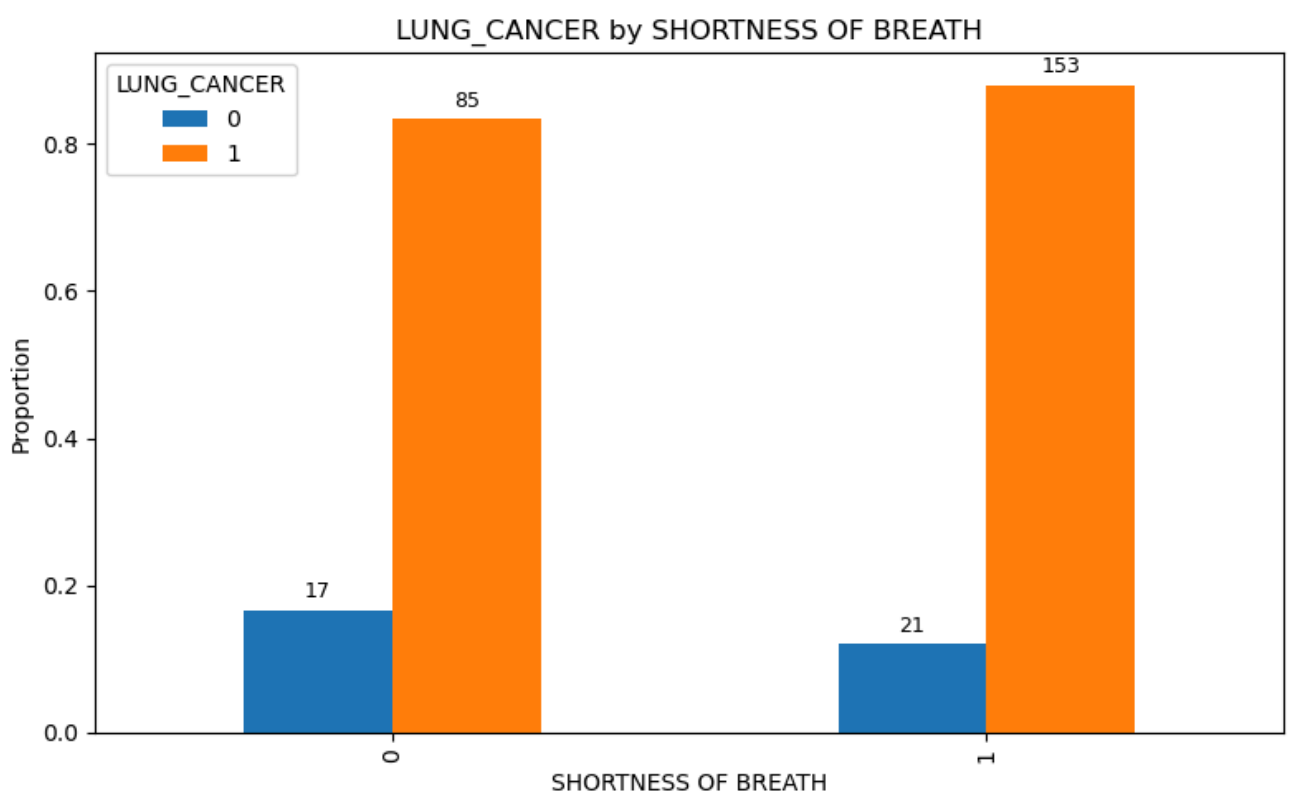

```
plot('ALCOHOL CONSUMING')
```



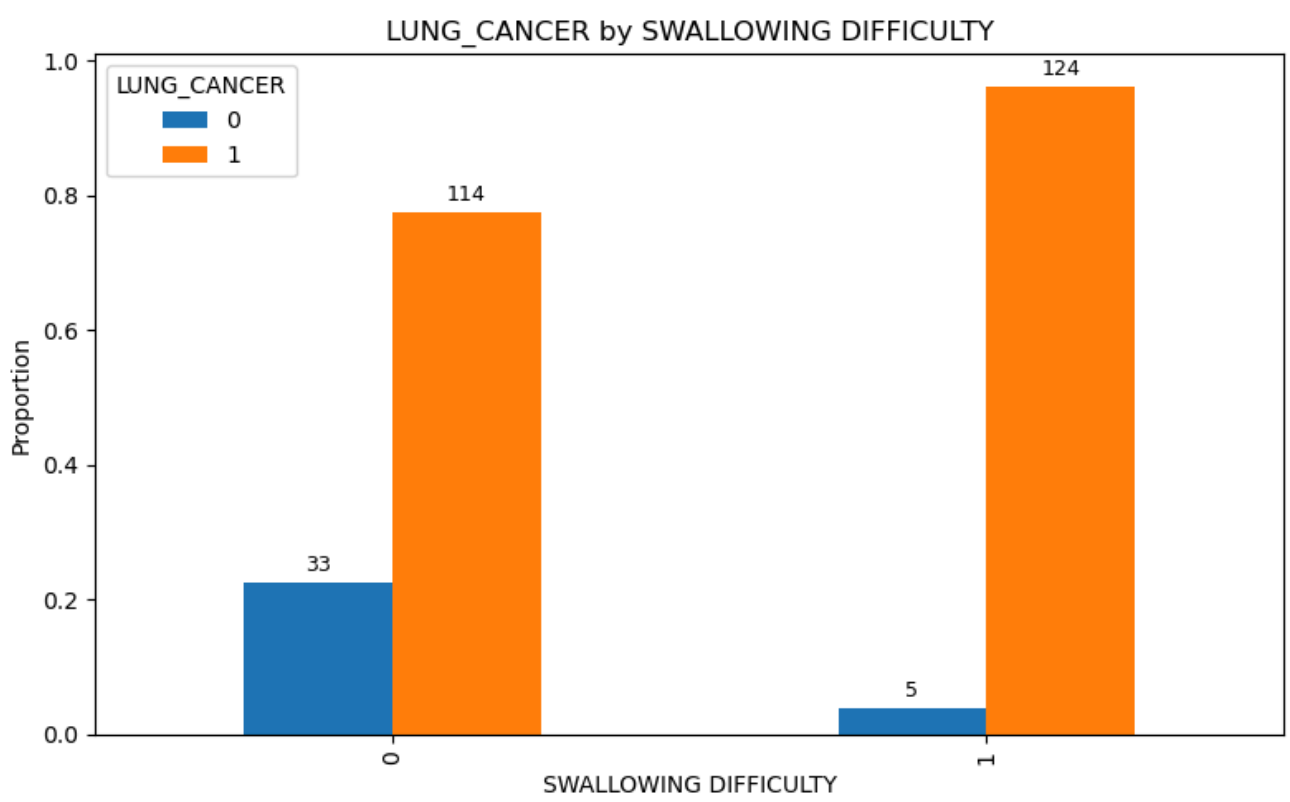

```
plot('COUGHING')
```



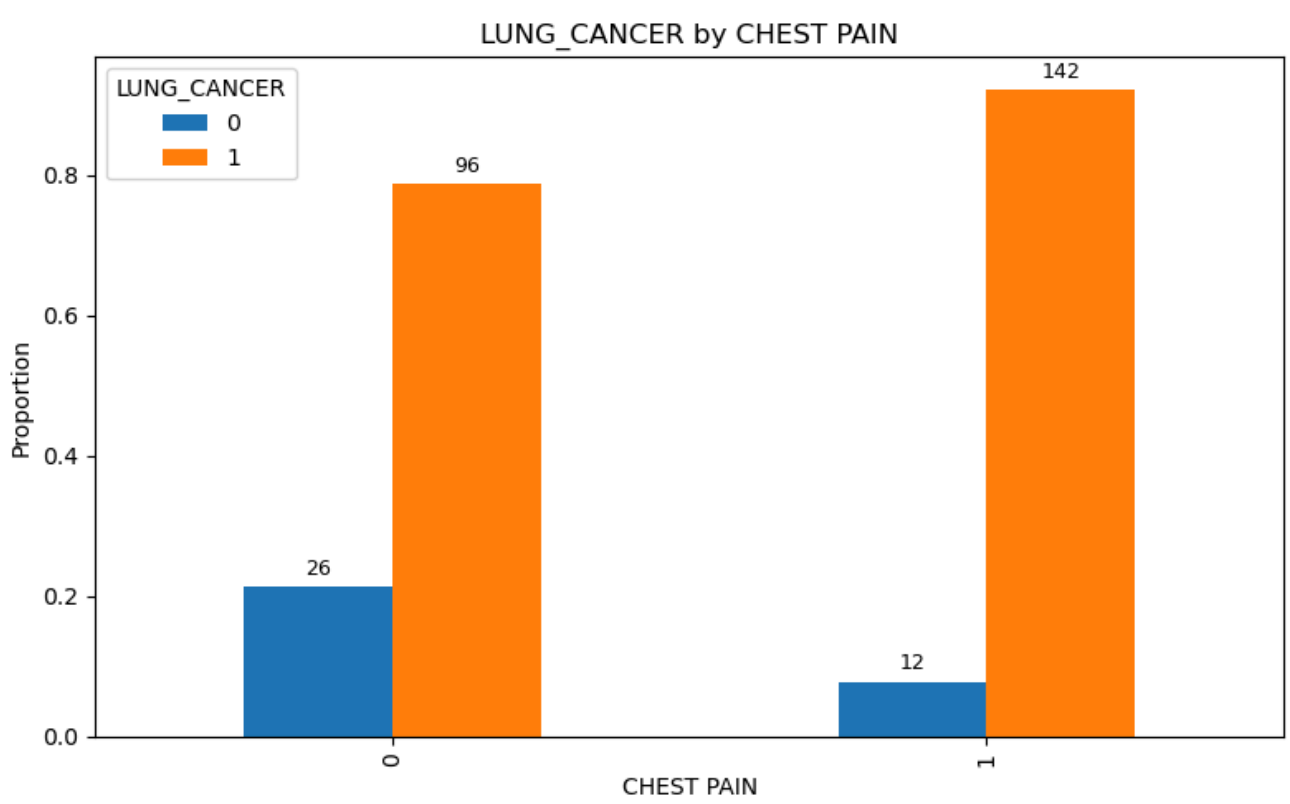

```
plot('SHORTNESS OF BREATH')
```




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




```
plot('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
0	1	1	0	0	1	0	1
1	0	0	0	1	1	1	0
2	0	0	1	0	1	0	1
3	1	1	0	0	0	0	0
4	1	0	0	0	0	0	1
...	...	...	...	...	...	...	...
279	1	1	1	0	0	1	1
280	0	0	0	1	1	1	0
281	0	0	0	0	1	1	0
282	1	1	0	0	0	0	0
283	1	1	0	0	1	0	1

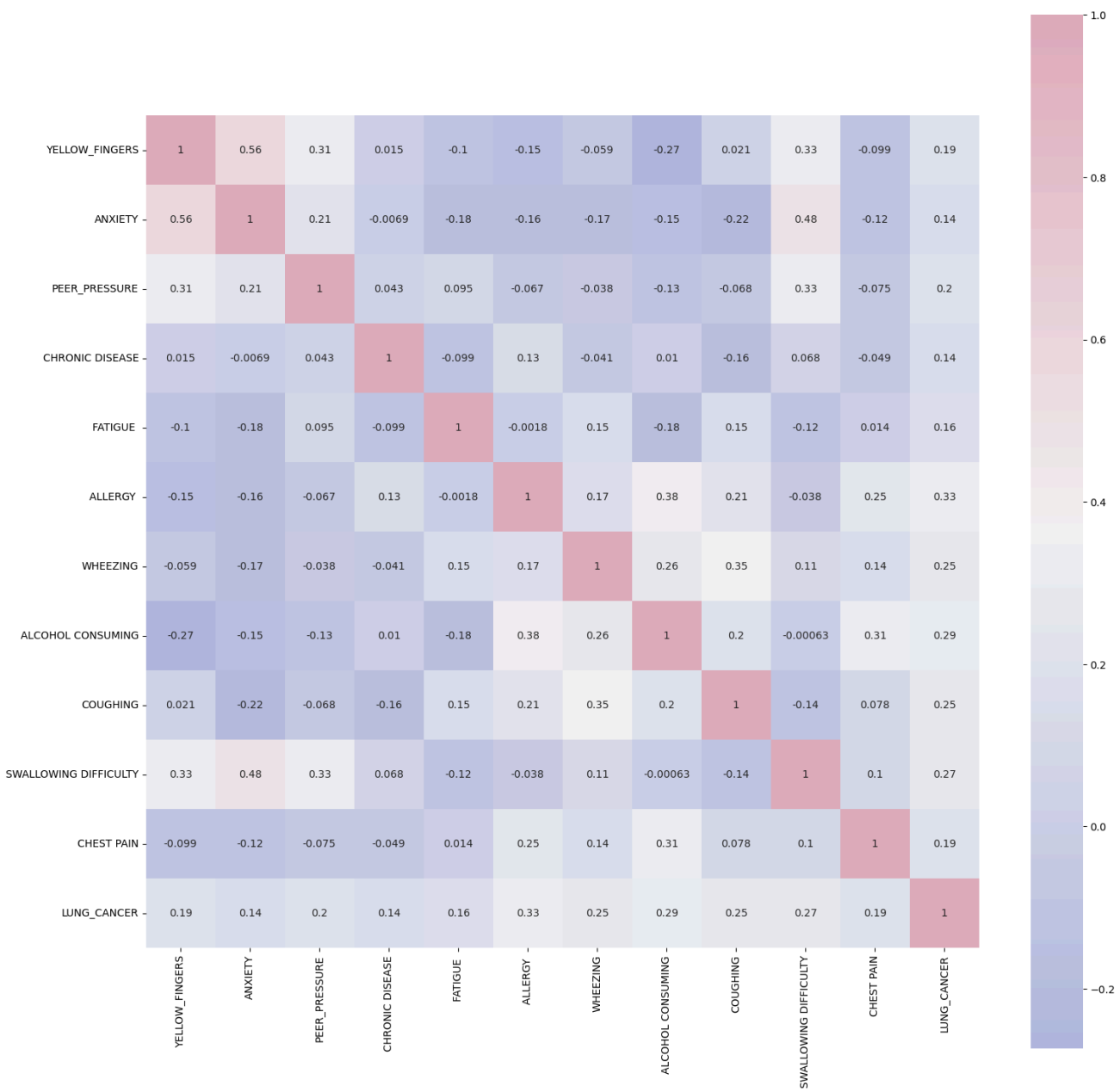
276 rows × 12 columns

CORRELATION

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

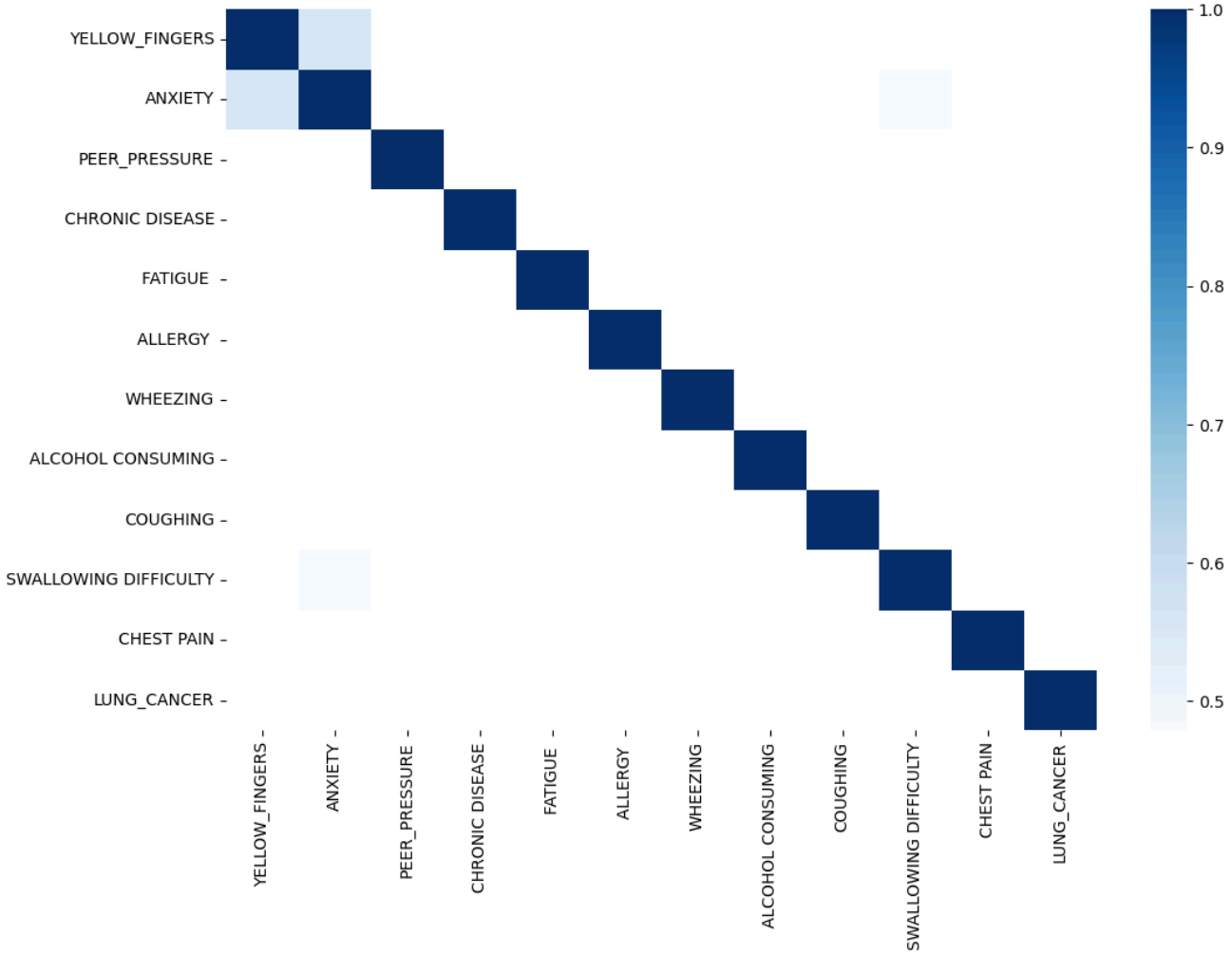
	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALL
YELLOW_FINGERS	1.000000	0.558344	0.313067	0.015316	-0.099644	-0.14
ANXIETY	0.558344	1.000000	0.210278	-0.006938	-0.181474	-0.15
PEER_PRESSURE	0.313067	0.210278	1.000000	0.042893	0.094661	-0.06
CHRONIC DISEASE	0.015316	-0.006938	0.042893	1.000000	-0.099411	0.13
FATIGUE	-0.099644	-0.181474	0.094661	-0.099411	1.000000	-0.00
ALLERGY	-0.147130	-0.159451	-0.066887	0.134309	-0.001841	1.00
WHEEZING	-0.058756	-0.174009	-0.037769	-0.040546	0.152151	0.16
ALCOHOL CONSUMING	-0.273643	-0.152228	-0.132603	0.010144	-0.181573	0.37
COUGHING	0.020803	-0.218843	-0.068224	-0.160813	0.148538	0.20
SWALLOWING DIFFICULTY	0.333349	0.478820	0.327764	0.068263	-0.115727	-0.03
CHEST PAIN	-0.099169	-0.123182	-0.074655	-0.048895	0.013757	0.24
LUNG_CANCER	0.189192	0.144322	0.195086	0.143692	0.160078	0.33

```
#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

Feature Engineering is the process of creating new features using existing features.

The correlation matrix shows that ANXIETY and YELLOW_FINGERS are correlated more than 50%. So, lets create a new feature combining them.

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

	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING
0	1	1	0	0	1	0	1
1	0	0	0	1	1	1	0
2	0	0	1	0	1	0	1
3	1	1	0	0	0	0	0
4	1	0	0	0	0	0	1
...	...	...	...	...	...	...	...
279	1	1	1	0	0	1	1
280	0	0	0	1	1	1	0
281	0	0	0	0	1	1	0
282	1	1	0	0	0	0	0
283	1	1	0	0	1	0	1

276 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)
```

```
483
```

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_st
```

```
#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, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 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      0.95      0.95        60
           1       0.95      0.95      0.95        61

    accuracy                           0.95       121
   macro avg       0.95      0.95      0.95       121
weighted avg       0.95      0.95      0.95       121
```

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, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0,
```

```
       1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0])
```

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

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95        60
           1       0.98      0.92      0.95        61

    accuracy                           0.95       121
   macro avg       0.95      0.95      0.95       121
weighted avg       0.95      0.95      0.95       121
```

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, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0])
```

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

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.94        60
           1       1.00      0.89      0.94        61

    accuracy                           0.94       121
   macro avg       0.95      0.94      0.94       121
weighted avg       0.95      0.94      0.94       121
```

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()
```

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

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

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

```
              precision    recall  f1-score   support

           0       0.95      0.87      0.90        60
           1       0.88      0.95      0.91        61

    accuracy                           0.91       121
   macro avg       0.91      0.91      0.91       121
weighted avg       0.91      0.91      0.91       121
```

Multinomial Naive Bayes

```
#Fitting Multinomial Naive Bayes classifier to the training set
from sklearn.naive_bayes import MultinomialNB
mnb_model = MultinomialNB()
mnb_model.fit(X_train, y_train)
```

```
MultinomialNB()
```

```
#Predicting result using testing data
y_mnb_pred= mnb_model.predict(X_test)
```

```
y_mnb_pred
```

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

```
#Model accuracy
mnb_cr=classification_report(y_test, y_mnb_pred)
print(mnb_cr)
```

```
              precision    recall  f1-score   support

           0       0.78      0.85      0.82        60
           1       0.84      0.77      0.80        61

    accuracy                           0.81       121
   macro avg       0.81      0.81      0.81       121
weighted avg       0.81      0.81      0.81       121
```

Support Vector Classifier

```
#Fitting SVC to the training set
from sklearn.svm import SVC
svc_model = SVC()
svc_model.fit(X_train, y_train)
```

```
SVC
SVC()
```

```
#Predicting result using testing data
y_svc_pred= svc_model.predict(X_test)
y_svc_pred
```

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

```
#Model accuracy
svc_cr=classification_report(y_test, y_svc_pred)
print(svc_cr)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98        60
           1       0.98      0.97      0.98        61
```

```
          accuracy                           0.98       121
         macro avg       0.98      0.98      0.98       121
      weighted avg       0.98      0.98      0.98       121
```

Random Forest

```
#Training
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
#Predicting result using testing data
y_rf_pred= rf_model.predict(X_test)
y_rf_pred
```

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

```
#Model accuracy
rf_cr=classification_report(y_test, y_rf_pred)
print(rf_cr)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98        60
           1       0.98      0.97      0.98        61

    accuracy                           0.98       121
   macro avg       0.98      0.98      0.98       121
weighted avg       0.98      0.98      0.98       121
```

XGBoost

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /opt/anaconda3/lib/python3.13/site-pa
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.13/site-pack
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.13/site-pack
Note: you may need to restart the kernel to use updated packages.
```

```
from xgboost import XGBClassifier
xgb_model = XGBClassifier()
xgb_model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None
              n_jobs=None, num_parallel_tree=None, ...)
```

```
#Predicting result using testing data
y_xgb_pred= xgb_model.predict(X_test)
y_xgb_pred
```

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

```
#Model accuracy
xgb_cr=classification_report(y_test, y_xgb_pred)
print(xgb_cr)
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        60
           1       0.97      0.97      0.97        61

    accuracy                           0.97       121
   macro avg       0.97      0.97      0.97       121
weighted avg       0.97      0.97      0.97       121
```

Multi-layer Perceptron classifier

```
#Training a neural network model
from sklearn.neural_network import MLPClassifier
mlp_model = MLPClassifier()
mlp_model.fit(X_train, y_train)
```

```
▾ MLPClassifier
MLPClassifier()
```

```
#Predicting result using testing data
y_mlp_pred= mlp_model.predict(X_test)
y_mlp_pred
```

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

```
#Model accuracy
mlp_cr=classification_report(y_test, y_mlp_pred)
print(mlp_cr)
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97        60
           1       0.98      0.95      0.97        61

    accuracy                           0.97       121
   macro avg       0.97      0.97      0.97       121
weighted avg       0.97      0.97      0.97       121
```

Gradient Boosting

```
#Training
from sklearn.ensemble import GradientBoostingClassifier
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
#Predicting result using testing data
y_gb_pred= gb_model.predict(X_test)
y_gb_pred
```

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

```
#Model accuracy
gb_cr=classification_report(y_test, y_gb_pred)
print(gb_cr)
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98        60
           1       0.98      0.97      0.98        61

    accuracy                           0.98       121
   macro avg       0.98      0.98      0.98       121
weighted avg       0.98      0.98      0.98       121
```

From the above calculated accuracies, it is clear that the SVC, Random Forest, Multi-layer Perceptron and Gradient Boost models performed atmost level while the worst performed one is Multinomial Naive Bayes. However, I'm interested in a more efficient way of evaluating these models. Let's go for the Cross Validation methods using both K-Fold and Stratified K-Fold

Cross Validation

```
# Standardn 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)

# Multinomial naive bayes model
mnb_model_scores = cross_val_score(mnb_model,X, y, cv=kf)

# Support Vector Classifier model
svc_model_scores = cross_val_score(svc_model,X, y, cv=kf)

# Random forest model
rf_model_scores = cross_val_score(rf_model,X, y, cv=kf)

# XGBoost model
```

```
    xgb_model_scores = cross_val_score(xgb_model,X, y, cv=kf)

    # Multi-layer perceptron model
    mlp_model_scores = cross_val_score(mlp_model,X, y, cv=kf)

    # Gradient boost model
```