Sardar Patel Institute of Technology,Mumbai

Department of Electronics and Telecommunication Engineering

B.E. Sem-VII- PE-IV (2024-2025)

**IT 24 - AI in Healthcare**

**Exp 2-Experiment: Decision Tree (ID3) algorithm**

**Name: Adwait Purao UID : 2021300101** **Batch : D** **Date: 27/8/24**

**Objective: Write Python program to demonstrate the working of the decision tree based ID3 algorithm by using appropriate medical data set for building the decision tree and apply this knowledge to forecast.**

**Outcomes:**

1. Find entropy of data and follow steps of the algorithm to construct a tree.
2. Representation of hypothesis using decision tree.
3. Apply Decision Tree algorithm to classify the given data.
4. Interpret the output of Decision Tree.

**System Requirements:**

Linux OS with Python and libraries or R or windows with MATLAB

**Theory:**

The decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Entropy

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

E(S) is the Entropy of the entire set, while the second term E(S, A) relates to an Entropy of an attribute A.





Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).



**Dataset Description:**

**Code:**

**#Importing Libraries**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

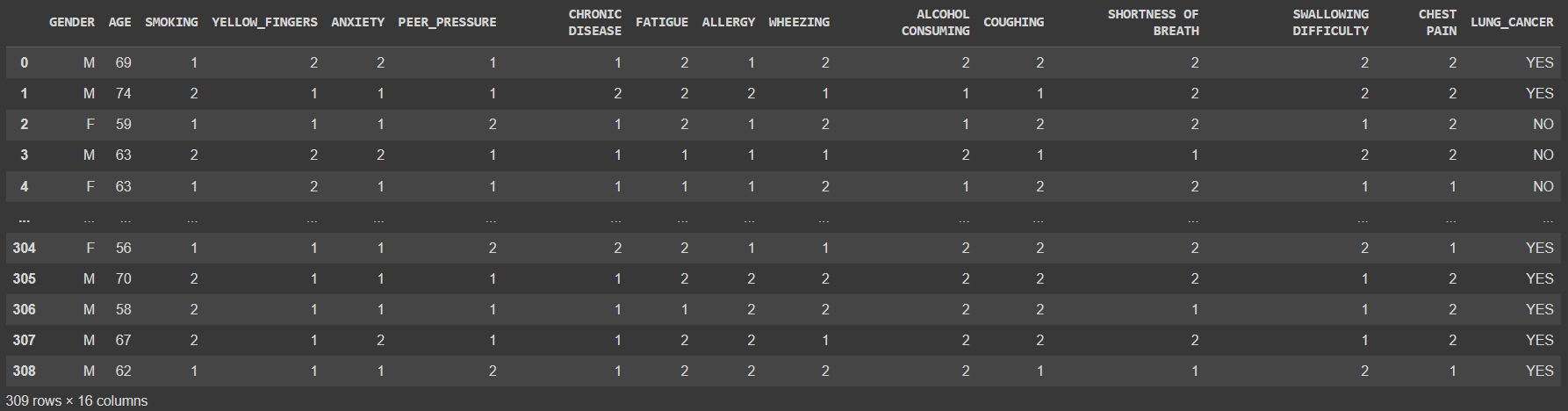
**#For ignoring warning**

**import warnings**

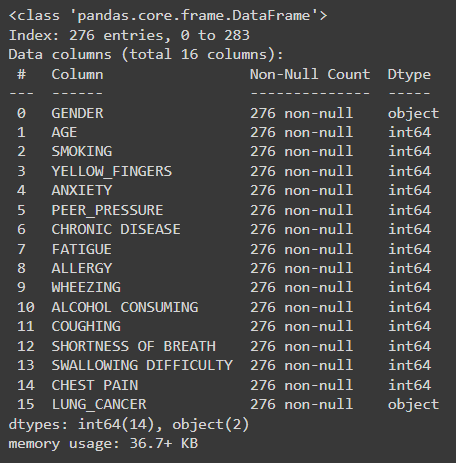
**warnings.filterwarnings("ignore")**

**df=pd.read\_csv('./survey lung cancer.csv')**

**df**

****

**df.info()**

****

**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'])**

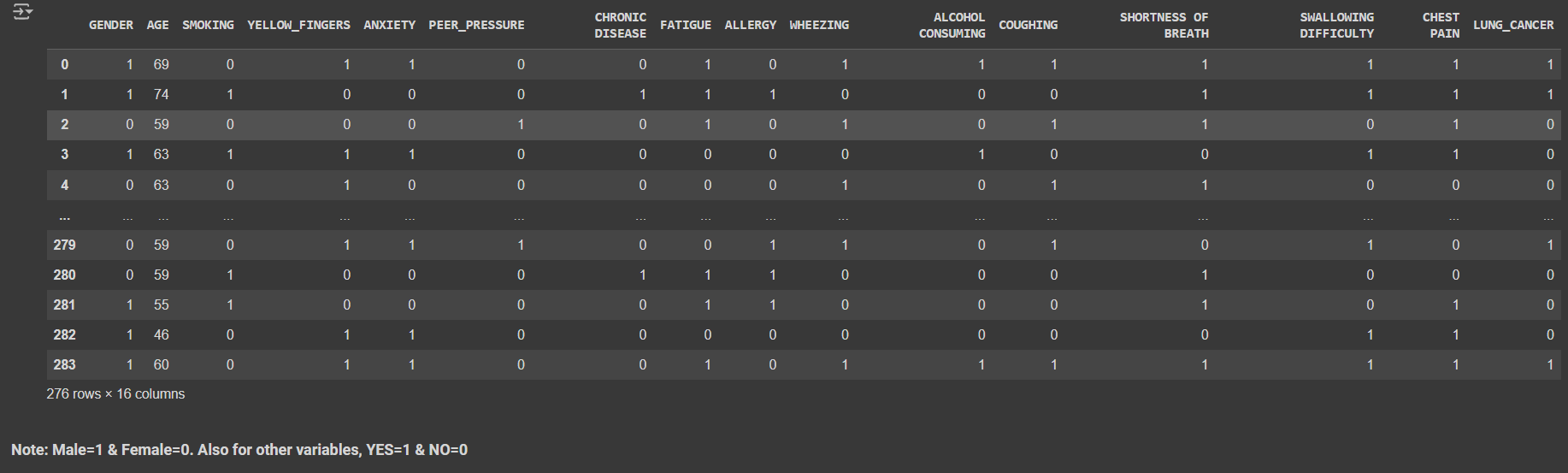
**df['SHORTNESS OF BREATH']=le.fit\_transform(df['SHORTNESS OF BREATH'])**

**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'])**

**df**

****

**import matplotlib.pyplot as plt**

**# Define a list of columns you want to plot**

**columns = ['GENDER', 'AGE', 'SMOKING', 'YELLOW\_FINGERS', 'ANXIETY', 'PEER\_PRESSURE',**

**'CHRONIC DISEASE', 'FATIGUE ', 'ALLERGY ', 'WHEEZING', 'ALCOHOL CONSUMING',**

**'COUGHING', 'SHORTNESS OF BREATH', 'SWALLOWING DIFFICULTY', 'CHEST PAIN']**

**# Create a figure with a grid of subplots**

**fig, axes = plt.subplots(nrows=5, ncols=3, figsize=(20, 25)) # Adjust the grid size as needed**

**fig.tight\_layout(pad=5.0) # Adjust padding between plots**

**# Flatten the axes array for easy iteration**

**axes = axes.flatten()**

**# Iterate over columns and plot each in its subplot**

**for i, col in enumerate(columns):**

**df.groupby(col)['LUNG\_CANCER'].value\_counts(normalize=True).unstack().plot(kind='bar', ax=axes[i])**

**axes[i].set\_title(f'Distribution of LUNG\_CANCER by {col}')**

**axes[i].set\_xlabel(col)**

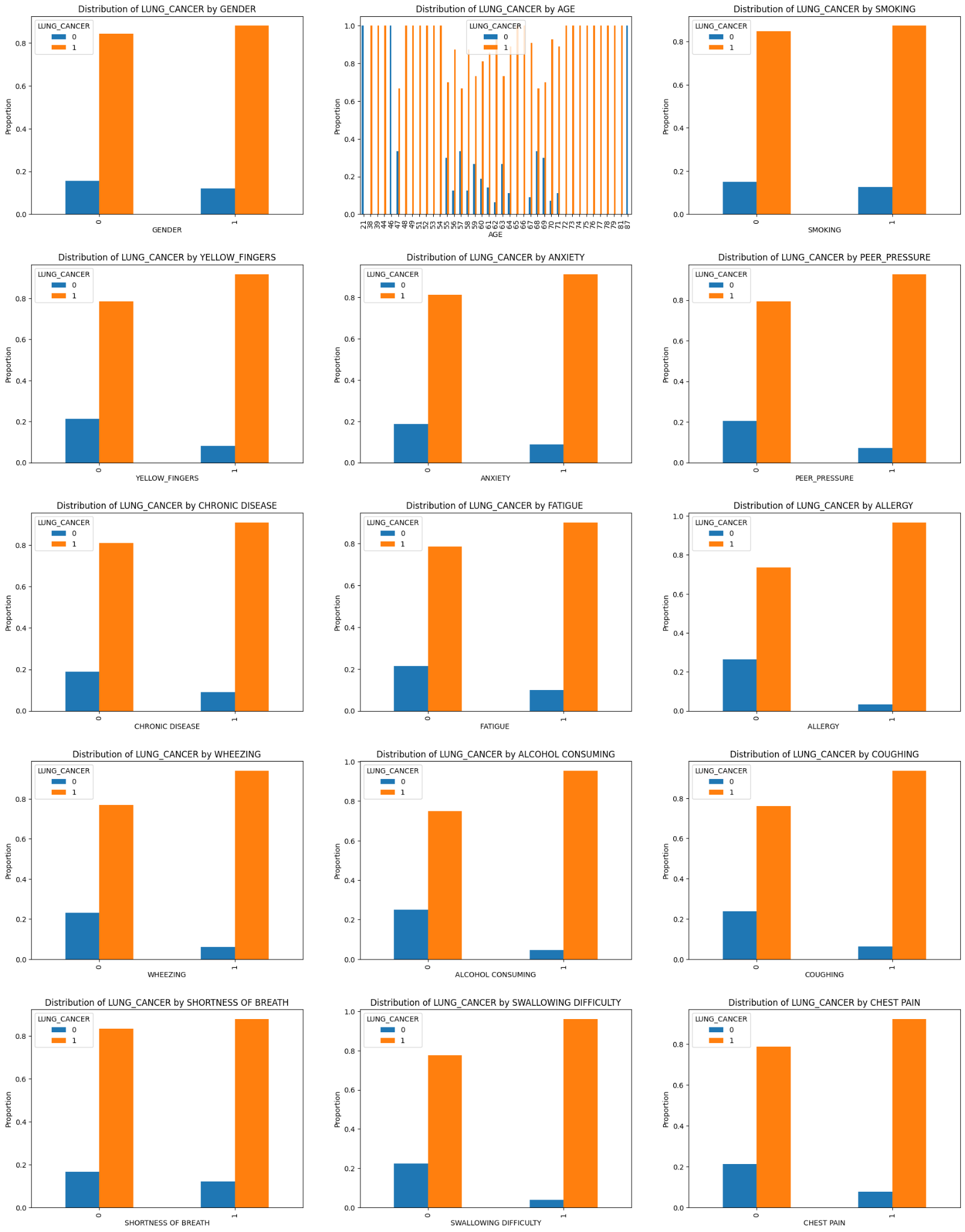
**axes[i].set\_ylabel('Proportion')**

**# Remove any empty subplots (if the grid is larger than the number of plots)**

**for j in range(i + 1, len(axes)):**

**fig.delaxes(axes[j])**

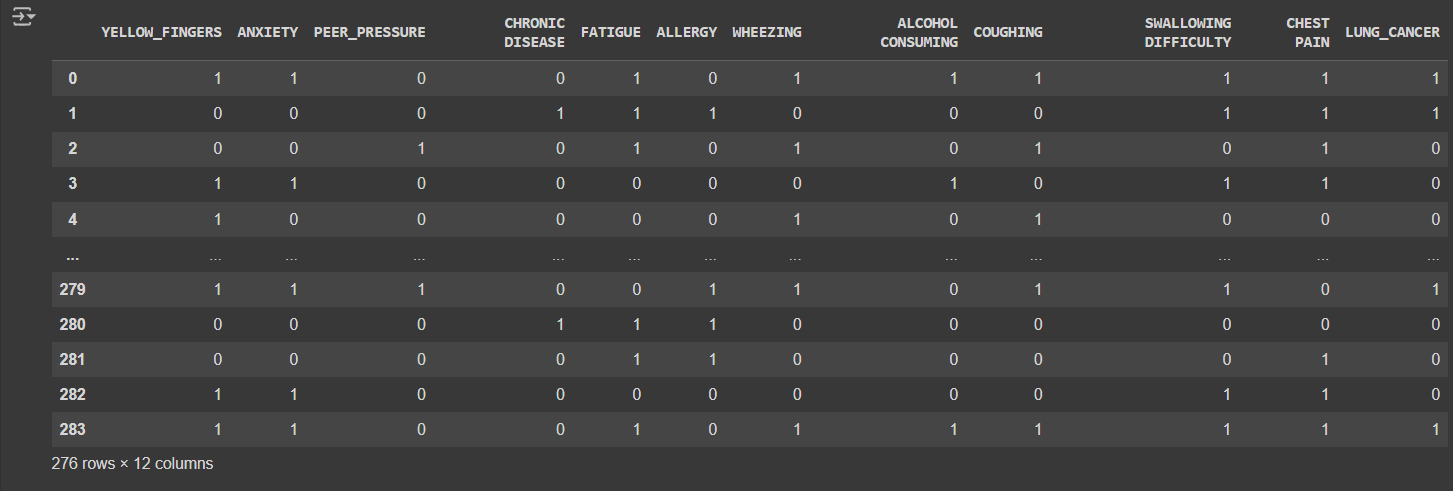
**plt.show()**

****

**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**

****

**#Correlation**

**cmap=sns.diverging\_palette(260,-10,s=50, l=75, n=6,**

**as\_cmap=True)**

**plt.subplots(figsize=(18,18))**

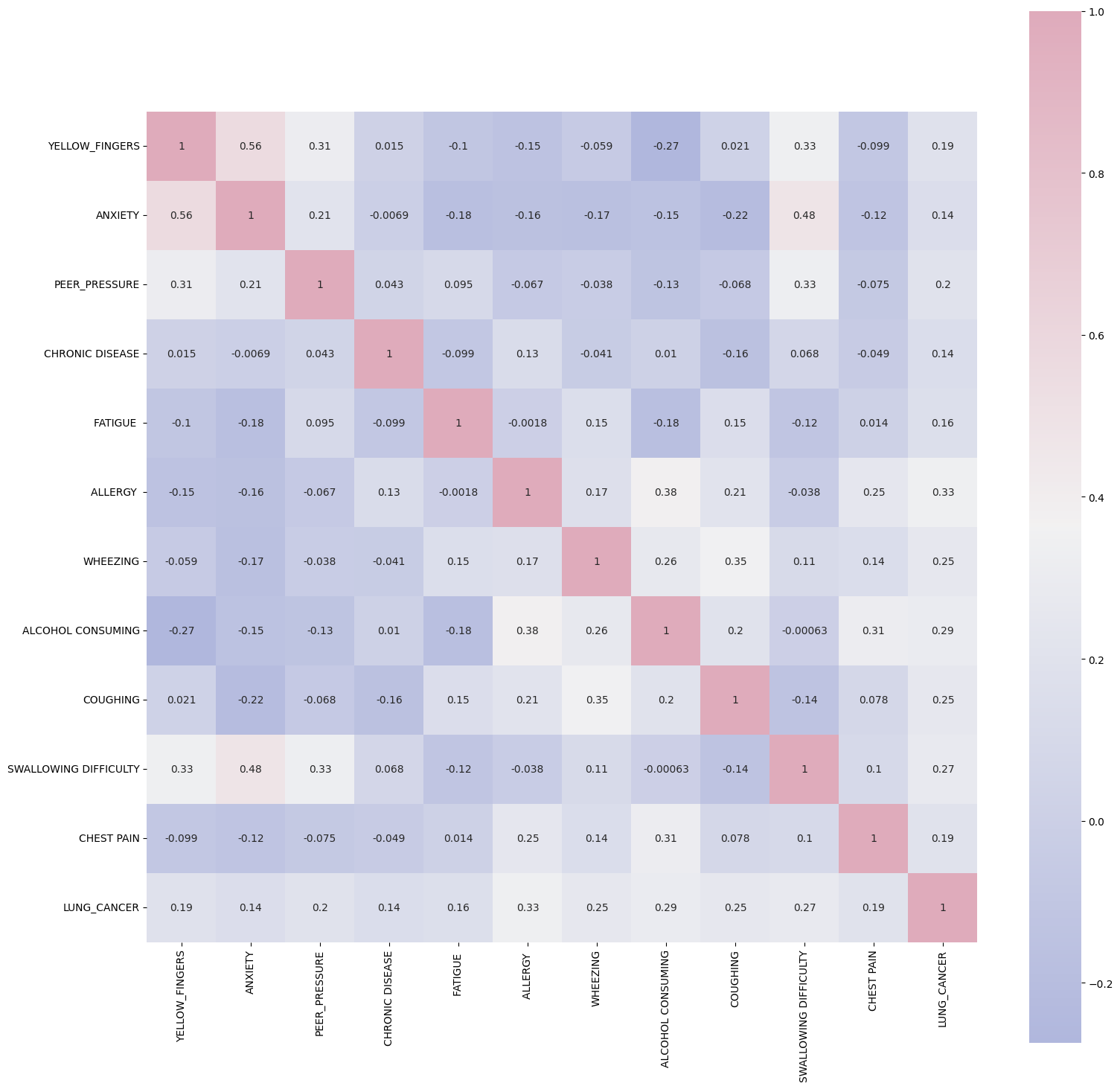
**sns.heatmap(cn,cmap=cmap,annot=True, square=True)**

**plt.show()**

**kot = cn[cn>=.40]**

**plt.figure(figsize=(12,8))**

**sns.heatmap(kot, cmap="Blues")**

****

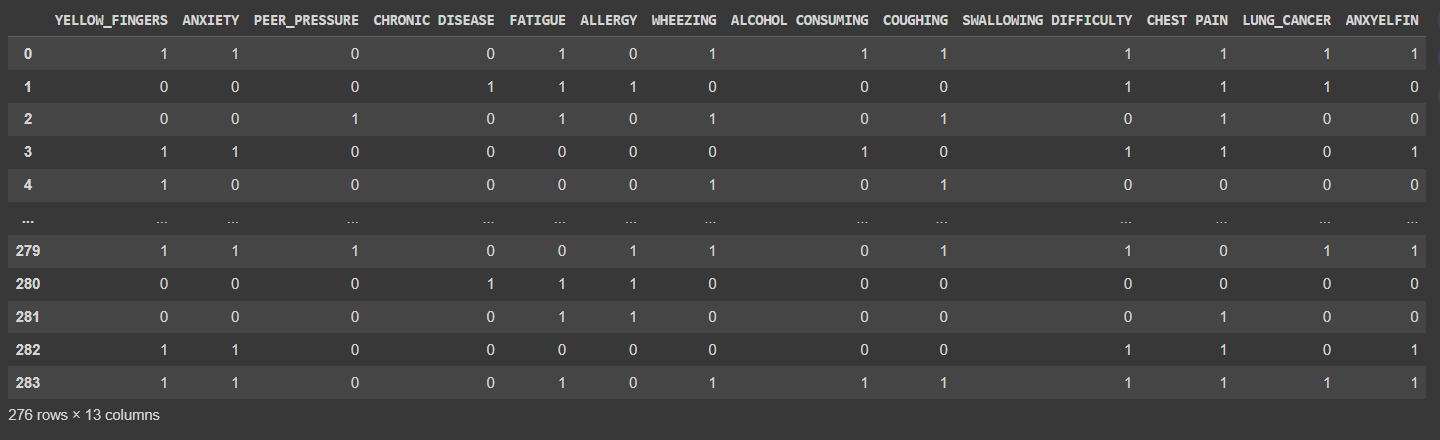
**Feature Engineering**

**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**

****

**#Splitting independent and dependent variables**

**X = df\_new.drop('LUNG\_CANCER', axis = 1)**

**y = df\_new['LUNG\_CANCER']**

**from imblearn.over\_sampling import ADASYN**

**adasyn = ADASYN(random\_state=42)**

**X, y = adasyn.fit\_resample(X, y)**

**len(X)**

****

# **Decision Tree**

**#Splitting data for training and testing**

**from sklearn.model\_selection import train\_test\_split**

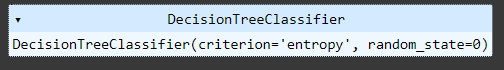
**X\_train, X\_test, y\_train, y\_test= train\_test\_split(X, y, test\_size= 0.25, random\_state=0)**

**#Fitting training data to the model**

**from sklearn.tree import DecisionTreeClassifier**

**dt\_model= DecisionTreeClassifier(criterion='entropy', random\_state=0)**

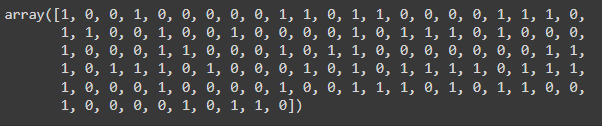
**dt\_model.fit(X\_train, y\_train)**

****

**#Predicting result using testing data**

**y\_dt\_pred= dt\_model.predict(X\_test)**

**y\_dt\_pred**

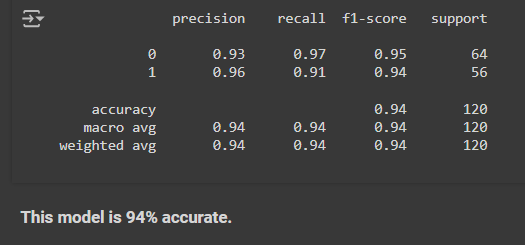
****

**#Model accuracy**

**from sklearn.metrics import classification\_report, accuracy\_score, f1\_score**

**dt\_cr=classification\_report(y\_test, y\_dt\_pred)**

**print(dt\_cr)**

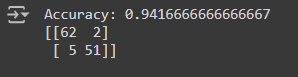
****

**from sklearn import metrics**

**print("Accuracy:",metrics.accuracy\_score(y\_test, y\_dt\_pred))**

**confusion\_matrix = metrics.confusion\_matrix(y\_test, y\_dt\_pred)**

**print(confusion\_matrix)**

****

**# Forecasting**

**sample\_index = 2 # change this index to predict different samples**

**# Access the row using .iloc for integer-location based indexing**

**sample = X\_test.iloc[sample\_index].values.reshape(1, -1)**

**prediction = dt\_model.predict(sample)**

**print(f"Predicted class for sample {sample\_index}: {prediction[0]}")**

****

**Conclusion:**

* We learned to calculate the entropy of the dataset and information gain of each attribute to decide the root node and subsequently the leaf nodes.
* We also used scikit to run the Decision Tree algorithm on a larger dataset and estimate the accuracy f the model created.
* In Decision Tree as the depth of the tree increases the model overfits the data and accuracy reduces to avoid these, parameters for pruning the tree should be passed to the classifier.