Swami Ramanand Teerth Marathwada University,Nanded

Department of Statistics

School of Mathematical Sciences

A Project Report

On

Exploration and Classification of Lung Cancer dataset Using Python

Submitted By:

Patil Prajakta Chandrakant

Satpute Shubhangi Santosh

Bhawar Ajinkya Raju

Wadepalli Harish Prakash

Submitted To:

Department of Statistics

School of Mathematical Science

Swami Ramanand Teerth Marathwada University,Nanded

Under the Guidance of

Dr. A.A. Muley

(2023-2024)

April, 2024

स्वामी रामानंद तीर्थ मराठवाडा विद्यापीठ, नांदेड.

Swami Ramanand Teerth Marathwada University,Nanded

Department Of Statistics

School Of Mathematical Science

CERTIFICATE

This is certify that,

Submitted By:

Patil Prajakta Chandrakant

Satpute Shubhangi Santosh

Bhawar Ajinkya Raju

Wadepalli Harish Prakash

have successfully completed their project report on "Exploration and Classification of Lung Cancer dataset Using Python" for the partial fulfilment of M.Sc. Statistics Second Year, in Semester –IV during the year 2023-2024.

Dr. A.A. Muley	External Examiner	Dr. D.D. Pawar
(Project Guide)		(Director)

DECLARATION

We hereby declare that as, the project entitled, "Exploration and Classification of Lung Cancer dataset Using Python" submitted by us, to the Swami Ramanand Teerth Marathwada University, Nanded is an original work. The Contents of the Project Report, in full or in parts have not been submitted to any other institute or university for the fulfilment of any other degree.

Name:

Patil Prajakta Chandrakant

Satpute Shubhangi Santosh

Bhawar Ajinkya Raju

Wadepalli Harish Prakash

DATE: April, 2024

ACKNOWLEDGEMENT

We wish to thank School of Mathematical Sciences, S.R.T.M. University, Nanded for giving us an opportunity to do a project. This report has been prepared under the guidance of Dr. A.A. Muley. We would like to express our profound gratitude towards him who has been a constant source of inspiration and also keen interest in the project development. Also, We are grateful to Dr. D.D. Pawar, Professor and Director, School of Mathematical Sciences for their encouragement in shaping idea and valuable suggestions in making it a reality. Finally, We would like to thank Research Student, non-teaching staff and of our department for their valuable co-operation in this project.

Name:

Patil Prajakta Chandrakant

Satpute Shubhangi Santosh

Bhawar Ajinkya Raju

Wadepalli Harish Prakash

CONTENTS

1. INTRODUCTION 4

2. METHOD OF DATA COLLECTION 5

3. OBJECTIVES 6

4. METHODOLOGY 6-15

4.1 Basic concepts of python

4.2 Subplot

4.3 Logistic Regression Model

4.4 Gaussian Naive Bayes Model

4.5 Bernoulli Naive Bayes Model

4.6 Support Vector Machine Model

4.7 Random Forest Model

4.8 K-Nearest Neighbour Model

4.9 Extreme Gradiant Boosting Model

4.10 Neural Network Architecture

4.11 Gradient Boosting Classifier

4.12 Adaboost Classifier

4.13 Decision Tree Classifier

4.14 Chi-Square Test

4.15 Proportion Test

5. RESULT 49

6. CONCLUSION 49

7. REFERENCES 50

1 INTRODUCTION

In this study, we have studied lung cancer data analysis. Cancer is a disease in which cells in the body grow out of control. When cancer starts in the lungs, it is called lung cancer. Lung cancer begins in the lungs and may spread to lymph nodes or other organs in the body, such as the brain. In this project, we have analysed the lung cancer data by using Python.

To begin, we have started with importing the necessary libraries such as Pandas, NumPy, and Matplotlib. Then, we have imported a dataset to analyze and visualize data using python i.e. to explore the data using descriptive statistics and data visualization tools such as histograms and subplot .

People who smoke have the greatest risk of lung cancer. Apart from that it can also occur in people who have never smoked. Its risk may increases with the length of time and number of cigarettes smoked by someone else. If someone may quit smoking can significantly reduce the chances of developing lung cancer.

Stage II ("stage 2"): The disease may have spread to your lymph nodes near your lungs. Stage III ("stage 3"): It has spread further into your lymph nodes and the middle of your chest. Stage IV ("stage 4"): Cancer has spread widely around your body. It may have spread to your brain, bones, or liver.

Fatigue, nausea, weakness, loss of appetite, weight loss, and vomiting were the most frequently mentioned symptoms that impaired daily functioning in patients with recently diagnosed lung cancer.

Globocan estimate of lung cancer in India would indicate that incidence of lung cancer in India is 70,275 (for all ages and both genders) with an age standardized incidence rate being 6.9 per 100,000 of our population.

People with non-small cell lung cancer can be treated with surgery, chemotherapy, radiation therapy, targeted therapy, or a combination of these treatments. People with small cell lung cancer are usually treated with radiation therapy and chemotherapy.

The drug Ingrid was given was approved in November 2023 by the Food and Drug Administration (FDA) to treat advanced non-small cell lung cancer with a ROS1 fusion. The approval was based on findings from the trial in which Ingrid was one of the very first patients.

By the end of this project, the effectiveness of cancer prediction system helps the people to know their cancer risk with low cost and it also helps the people to take the appropriate decision based on their cancer risk status.

2 Method of data collection

For the project titled "Exploration and Classification of Lung Cancer dataset Using Python" we have collected secondary data from the Kaggle link:
"https://www.kaggle.com/code/hasibalmuzdadid/lung-cancer-analysis-accuracy-96-4/input"
We study 309 sample observations (Male and Female) with various symptoms of lung cancer.

3 OBJECTIVES

The objectives of this study is as follows:

1. To apply unsupervised learning techniques on lung cancer dataset. Graphical Techniques like Subplot, Histogram and Piechart.
2. To apply supervised learning techniques on lung cancer dataset like various classifiers as Random Forest Classifier, Kneighbour Classifier, Support Vector Machine, Gradient Boosting Classifier,Adaboost Classifier and Decision Tree Classifier.
3. To perform the comparative study of various classifiers and To check the accuracy.
4. To predicting the best classifier on the basis of higher accuracy among all classifiers.
5. To test the genderwise significance with various symptoms of lung cancer.
6. To perform proportion test to check the significance of male and female patients having lung cancer.

By achieving these objectives,The effectiveness of cancer prediction system helps the people to know their cancer risk with low cost and it also helps the people to take the appropriate decision based on their cancer risk status.

4 METHODOLOGY

4.1 Basic concepts of python

Statsmodel: Statsmodels is a Python library for statistical modeling and analysis. It provides a range of tools for statistical analysis, including linear regression, generalized linear models, time series analysis, and more.

Matplotlib: Matplotlib is a popular plotting library for Python. It allows users to create a wide variety of static, animated, and interactive visualizations in Python.

Sklearn: scikit-learn, also known as sklearn, is a popular machine learning library for Python. It provides a wide range of machine learning algorithms and tools for tasks such as classification, regression, clustering, and dimensionality reduction.

train test split: train test split is a function from the model selection module in scikitlearn library. It is used to split a given dataset into two parts, namely the training set and the test set.

The purpose of splitting a dataset is to train a machine learning model on the training set and evaluate its performance on the test set. This helps to estimate how well the model will perform on new, unseen data.

Seaborn: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics.

scipy: scipy is a Python library that provides a range of scientific computing tools and algorithms. It is built on top of numpy, another popular scientific computing library for Python.

Random import seed, random, randint: The random module is a built-in Python library that allows you to generate pseudo-random numbers. It provides several functions for generating random numbers, including integers, floats, and sequences.

The seed() function is used to initialize the random number generator with a starting value, which ensures that the same sequence of random numbers is generated every time the code is run with the same seed value.

The random() function generates a random float number between 0 and 1, while the randint() function generates a random integer between a specified range of values.

Overall, the random module is a useful tool for many applications that require randomization, such as simulations, games, and statistical analysis.

Seed: The seed() function is a built-in function in the random module of Python. It is used to initialize the pseudo-random number generator with a starting value, known as a seed.

4.2 Subplot

In statistics, a subplot refers to a smaller, individual plot within a larger graphical representation, such as a figure or chart. Subplots are often used to display multiple sets of data or different aspects of a dataset in a clear and organized manner. Subplots can also be used to compare different groups or categories within a dataset, allowing for easier comparison and analysis.

Histogram

A histogram is a graphical representation of the distribution of numerical data. It consists of a series of contiguous rectangles (bars), where the area of each rectangle corresponds to the frequency or proportion of observations within a particular interval (bin) of the data. It provide a visual summary of the underlying distribution, making it easier to understand key characteristics of the dataset.

4.3 Logistic Regression Model

Logistic regression is a statistical method used for analyzing binary or categorical dependent variables. It is a type of generalized linear model that models the probability of an event occurring, given certain input variables. The logistic regression model takes a set of input variables and calculates a weighted sum of these variables. This weighted sum is then passed through the logistic function, which maps the output to a probability value between 0 and 1. The logistic function is defined as:

$$p = 1/(1 + e^{-z})$$

Overall, logistic regression is a powerful tool for modeling binary or categorical data, but its assumptions and limitations should be carefully considered when using it for predictive modeling.

4.4 Gaussian Naive Bayes Model

The Gaussian Naive Bayes model is a variant of the Naive Bayes algorithm, which is a probabilistic classification method based on Bayes' theorem with the assumption of independence between features. In the Gaussian Naive Bayes model, it is assumed that the features follow a Gaussian (normal) distribution. Naive Bayes assumes that the features are conditionally independent given the class. Gaussian Naive Bayes is commonly used for classification tasks, especially when dealing with continuous or real-valued features that can be assumed to follow a Gaussian distribution. However, it may not perform well if the independence assumption is violated or if the data distribution is significantly different from a Gaussian distribution.

4.5 Bernoulli Naive Bayes Model –

In statistics and probability theory, the Bernoulli distribution is a discrete probability distribution of a random variable which takes the value 1 with probability p and the value 0 with probability q = 1 p, where 0 < p < 1. It's often used for modeling binary outcomes, such as success/failure or yes/no. It's a probabilistic classifier based on Bayes' theorem with the assumption of independence between features. Naive Bayes classifiers are commonly used for text classification tasks, such as spam filtering or document categorization. Bernoulli Naive Bayes model is a specific type of Naive Bayes classifier where features are assumed to be binary (Bernoulli distributed), often used in text classification tasks where the presence or absence of certain words is considered.

4.6 Support Vector Machine Model-

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. Its primary objective is to find the optimal hyperplane that separates data points of different classes with the largest possible margin.

4.7 Random Forest Model-

A Random Forest is a powerful ensemble learning technique used for both classification and regression tasks. It operates by constructing multiple decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forest uses a technique called bagging, which involves sampling the training data with replacement to create multiple subsets. Each decision tree is then trained on one of these subsets, which helps to reduce overfitting and improve generalization. Random Forests are known for their high performance, scalability, and ability to handle high-dimensional data with noisy features.

4.8 K-Nearest Neighbour Model-

In statistics and machine learning, the k-nearest neighbors (KNN) algorithm is a non-parametric method used for both classification and regression tasks. It is based on the idea that data points with similar characteristics tend to belong to the same class or have similar outcomes. The "k" in KNN refers to the number of nearest neighbors considered when making predictions. To classify a new data point, KNN finds the "k" nearest neighbors in the training dataset based on a distance metric (e.g., Euclidean distance) and assigns the class label that is most common among those neighbors (for classification tasks).

4.9 Extreme Gradiant Boosting Model

Extreme Gradient Boosting (XGBoost) is a powerful ensemble learning technique primarily used for regression and classification tasks. It belongs to the gradient boosting family of algorithms, which iteratively builds an ensemble of weak learners (typically decision trees) and combines their predictions to improve overall performance. XGBoost builds an ensemble of decision trees sequentially, where each tree corrects the errors made by the previous ones. It minimizes a loss function by adding trees that reduce the residuals of the previous predictions. XGBoost can automatically handle missing values by learning the best direction to go when a value is missing during training. XGBoost is known for its speed and performance, often outperforming other algorithms in various machine learning competitions and real-world applications. It is widely used in industry for tasks such as click-through rate prediction, customer churn prediction, and fraud detection.

4.10 Neural Network Architecture

Neural network architecture refers to the structure or layout of a neural network, which consists of multiple layers of interconnected neurons. In statistics and machine learning, neural networks are used for various tasks, including classification, regression, and pattern recognition.

Key components of neural network architecture:

1. *Input Layer*: This layer consists of neurons that receive the input data. Each neuron corresponds to one feature of the input data.

2. *Hidden Layers*: Hidden layers are layers of neurons between the input and output layers. They perform computations on the input data through a series of weighted connections and activation functions. Deep neural networks have multiple hidden layers, hence the term "deep learning."

3. *Output Layer*: The output layer consists of neurons that produce the final output of the neural network. The number of neurons in the output layer depends on the type of task. For example, in binary classification tasks, there is typically one output neuron representing the probability of belonging to one class, while in multi-class classification tasks, there is one output neuron per class.

4. *Activation Functions*: Activation functions introduce non-linearity into the neural network, allowing it to learn complex patterns in the data. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax.

5. *Connections and Weights*: Each neuron in a neural network is connected to every neuron in the adjacent layers, forming a network of weighted connections. During training, these weights are adjusted iteratively using optimization algorithms such as gradient descent to minimize the loss function and improve the network's performance.

6. *Architecture Variants*: There are various neural network architectures tailored for specific tasks, such as convolutional neural networks (CNNs) for image recognition, recurrent neural networks (RNNs) for sequential data, and transformer architectures for natural language processing tasks.

Neural network architecture design involves choosing the appropriate number of layers, number of neurons in each layer, activation functions, and optimization techniques to achieve the desired performance on the task at hand.

4.11 Gradient Boosting Classifier-

A Gradient Boosting Classifier is a type of ensemble learning method used for classification tasks. It belongs to the family of gradient boosting algorithms, which sequentially build a strong predictive model by combining the predictions of multiple weaker models, typically decision trees. It focuses on reducing the residuals at each step. Gradient Boosting Classifiers are known for their high accuracy and robustness, and it perform well on a wide range of classification tasks. Popular implementations include XGBoost, LightGBM, and CatBoost, which offer efficient and scalable implementations of gradient boosting algorithms.

4.12 Adaboost Classifier-

AdaBoost, short for Adaptive Boosting, is a popular ensemble learning algorithm used for classification tasks. It belongs to the family of boosting algorithms, which combine multiple weak learners (often simple decision trees or "stumps") to create a strong classifier. The final prediction of the AdaBoost classifier is determined by the weighted majority vote of the weak learners. One advantage of AdaBoost is its ability to handle complex datasets and achieve high accuracy with relatively simple weak learners. However, it can be sensitive to noisy data and outliers. Popular implementations of AdaBoost include those found in Python libraries such as scikit-learn.

4.13 Decision Tree Classifier-

A Decision Tree Classifier is a supervised machine learning algorithm used for classification tasks. It works by recursively partitioning the feature space into regions and assigning class labels to each region based on the majority class of the training examples within that region.

Decision Tree Classifiers have several advantages, including: - *Interpretability*: Decision trees are easy to understand and interpret, making them useful for explaining the reasoning behind predictions. - *Nonlinear Relationships*: They can capture nonlinear relationships between features and class labels without the need for feature engineering. - *Robustness to Outliers*: Decision trees are robust to outliers and can handle noisy data.

However, they also have limitations such as overfitting, especially when the tree depth is not controlled or when the dataset is imbalanced. Techniques such as pruning, limiting the tree depth, and using ensemble methods like Random Forests can help mitigate overfitting and improve performance

4.14 Chi-Square Test-

The Chi-Square test, also known as the χ^2 (chi-squared) test, is a statistical test used to determine whether there is a significant association between categorical variables. It's commonly used to analyze contingency tables, which are tables that display the frequency distribution of two or more categorical variables.

Here's how the Chi-Square test works: 1. *Hypotheses*: The Chi-Square test typically involves two hypotheses:
Null Hypothesis (H_0): There is no significant association between the categorical variables.
Alternative Hypothesis (H_1): There is a significant association between the categorical variables.

2. *Contingency Table*: The data is organized into a contingency table, also known as a cross-tabulation table, which displays the frequency counts for each combination of categories of the variables being studied.

3. *Expected Frequencies*: The Chi-Square test calculates the expected frequency for each cell in the contingency table under the assumption of independence between the variables. The expected frequency is calculated based on the marginal totals of the table.

4. *Chi-Square Statistic*: The Chi-Square statistic is calculated as the sum of the squared differences between the observed and expected frequencies, divided by the expected frequencies. It measures the discrepancy between the observed and expected frequencies and quantifies how well the observed frequencies fit the expected frequencies under the null hypothesis.

5. *P-value*: The Chi-Square test generates a p-value, which represents the probability of observing the observed frequencies (or more extreme frequencies) if the null hypothesis were true. A small p-value indicates that the observed frequencies are significantly different from the expected frequencies, leading to the rejection of the null hypothesis in favor of the alternative hypothesis.

6. *Conclusion*: Based on the p-value and a chosen significance level (typically 0.05), you can decide whether to reject the null hypothesis and conclude that there is a significant association between the categorical variables.

The Chi-Square test is widely used in various fields, including social sciences, biology, and market research, to analyze categorical data and determine whether there are relationships or dependencies between categorical variables.

4.15 Proportion Test-

A proportion test, also known as a test of proportions or a binomial test, is a statistical test used to determine whether the proportion of successes in a sample is significantly different from a specified value or from the proportion of successes in another sample.

Here's how a proportion test works: 1. *Hypotheses*: The proportion test typically involves two hypotheses: - Null Hypothesis (H): The proportion of successes in the sample is equal to a specified value (e.g., a population proportion) or the proportion of successes in another sample. - Alternative Hypothesis (H): The proportion of successes in the sample is not equal to the specified value or the proportion of successes in another sample.

2. *Test Statistic*: The test statistic for a proportion test is often based on the binomial distribution, which describes the distribution of the number of successes in a fixed number of independent Bernoulli trials (binary outcomes). The most commonly used test statistic is the Z-test statistic, which is calculated as the difference between the sample proportion and the hypothesized proportion, divided by the standard error of the sample proportion. 3. *Standard Error*: The standard error of the sample proportion is calculated based on the formula for the standard deviation of a binomial distribution, taking into account the sample size and the hypothesized proportion.

4. *P-value*: The proportion test generates a p-value, which represents the probability of observing the sample proportion (or a more extreme proportion) if the null hypothesis were true. A small p-value indicates that the observed proportion is significantly different from the hypothesized proportion, leading to the rejection of the null hypothesis in favor of the alternative hypothesis.

5. *Conclusion*: Based on the p-value and a chosen significance level (typically 0.05), you can decide whether to reject the null hypothesis and conclude that there is a significant difference in proportions.

Proportion tests are commonly used in various fields, including epidemiology, marketing research, and quality control, to compare proportions between different groups or to test whether a proportion deviates from an expected value. They are particularly useful when dealing with categorical data with two possible outcomes (e.g., success or failure, yes or no).

1. Random Sampling Step: The first step in RANSAC is to randomly select a subset of the data points. This subset is known as the "inliers" and is used to fit a model. The size of the subset is determined by the user-defined parameter "min samples". The number of iterations is also determined by the user-defined parameter "max iter”.

2. Model Fitting Step: The second step in RANSAC is to fit a model to the selected inliers. The specific model used depends on the problem being solved. For linear regression problems, the model is a straight line. For non-linear regression problems, the model may be a polynomial or another non-linear function.

Once the model has been fitted to the inliers, it is used to classify the remaining data points as either inliers or outliers. This is done by calculating the residuals for each data point, which is the vertical distance between the data point and the model. If the residual is less than a user-defined threshold, the data point is classified as an inlier. Otherwise, it is classified as an outlier.

The RANSAC algorithm repeats these two steps multiple times, and the model with the most inliers is selected as the final model. The algorithm terminates when either the maximum number of iterations has been reached or the number of inliers exceeds a user-defined threshold.

In summary, the RANSAC algorithm is a robust regression algorithm that can fit a model to a set of data points containing outliers. It works by iteratively selecting random subsets of the data and using them to fit a model. The best model is selected based on the number of inliers it correctly identifies.

#

Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
C:\Users\Acerr\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version 1.26.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

#

About Dataset

- **GENDER :** M [Male] , F [Female]
- **AGE :** Age of patients
- **SMOKING :** 2 [Yes] , 1 [No]
- **YELLOW_FINGERS :** 2 [Yes] , 1 [No]

- **ANXIETY :** 2 [Yes] , 1 [No]
- **PEER_PRESSURE :** 2 [Yes] , 1 [No]
- **CHRONIC DISEASE :** 2 [Yes] , 1 [No]
- **FATIGUE :** 2 [Yes] , 1 [No]
- **ALLERGY :** 2 [Yes] , 1 [No]
- **WHEEZING :** 2 [Yes] , 1 [No]
- **ALCOHOL CONSUMING :** 2 [Yes] , 1 [No]
- **COUGHING :** 2 [Yes] , 1 [No]
- **SHORTNESS OF BREATH :** 2 [Yes] , 1 [No]
- **SWALLOWING DIFFICULTY :** 2 [Yes] , 1 [No]
- **CHEST PAIN :** 2 [Yes] , 1 [No]
- **LUNG_CANCER :** YES [Positive] , NO [Negative]

[2]:
```
data = pd.read_csv("E:/survey_lung_cancer.csv")
```

#

Basic Exploration

Let's have a glimpse of the dataset.

[3]:
```
print(f"Shape of The Dataset : {data.shape}")
print(f"\nGlimpse of The Dataset :")
data.head().style.set_properties(**{"background-color": "#2a9d8f","color":
 ↪"white","border": "1.5px  solid black"})
```

```
Shape of The Dataset : (309, 16)
```

```
Glimpse of The Dataset :
```

```
[3]: <pandas.io.formats.style.Styler at 0x26feaf8f250>
```

```
[4]: print(f"Informations About The Dataset :\n")
     print(data.info())
```

```
Informations About The Dataset :

<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
None
```

#

Dataset Summary

```
[5]: print(f"Summary of This Dataset :")
     data.describe().T.style.set_properties(**{"background-color": "#2a9d8f","color":
      ↪"white","border": "1.5px  solid black"})
```

```
Summary of This Dataset :
```

```
[5]: <pandas.io.formats.style.Styler at 0x26feaf87fd0>
```

```
[6]: data.describe(include=object).T.style.set_properties(**{"background-color":␣
      ↪"#2a9d8f","color":"white","border": "1.5px  solid black"})
```

```
[6]: <pandas.io.formats.style.Styler at 0x26f86329a90>
```

```
[7]: data.isna().sum().to_frame().T.style.set_properties(**{"background-color":␣
      ↪"#2a9d8f","color":"white","border": "1.5px  solid black"})
```

```
[7]: <pandas.io.formats.style.Styler at 0x26f862f1b20>
```

Here, we can see there is **no null value** exists in this dataset. Let's check if there exists any duplicate entry in this dataset. If exists then we will remove them from the dataset. After that we will initialize the visualization style and custom pallete for visualization.

```
[8]: dup = data[data.duplicated()].shape[0]
     print(f"There are {dup} duplicate entries among {data.shape[0]} entries in this␣
      ↪dataset.")

     data.drop_duplicates(keep='first',inplace=True)
     print(f"\nAfter removing duplicate entries there are {data.shape[0]} entries in␣
      ↪this dataset.")
```

```
There are 33 duplicate entries among 309 entries in this dataset.

After removing duplicate entries there are 276 entries in this dataset.
```

#

Digging Deeper

Let's replace all the **numeric values** into **categorical values** for better and efficient visualization. All the **2**'s and **1**'s will be replaced by **"YES"** and **"NO"** accordingly. And replace **"M"** and **"F"** by **"Male"** and **"Female"** accordingly in **"GENDER"** column.

```
[9]: data_temp = data.copy()
     data_temp["GENDER"] = data_temp["GENDER"].replace({"M" : "Male" , "F" :␣
      ↪"Female"})

     for column in data_temp.columns:
         data_temp[column] = data_temp[column].replace({2: "Yes" , 1 : "No"})

     data_temp.head().style.set_properties(**{"background-color": "#2a9d8f","color":
      ↪"white","border": "1.5px  solid black"})
```

```
[9]: <pandas.io.formats.style.Styler at 0x26f86302940>
```

#

Custom Palette For Visualization

```
[10]: sns.set_style("whitegrid")
      sns.set_context("poster",font_scale = .7)

      palette = ["#1d7874","#679289","#f4c095","#ee2e31","#ffb563","#918450","#f85e00",
      "#a41623","#9a031e","#d6d6d6","#ffee32","#ffd100","#333533","#202020"]
```

```
# sns.palplot(sns.color_palette(palette))
# plt.show()
```

#

Positive Lung Cancer Cases

Let's create a new dataframe containing only positive cases data.

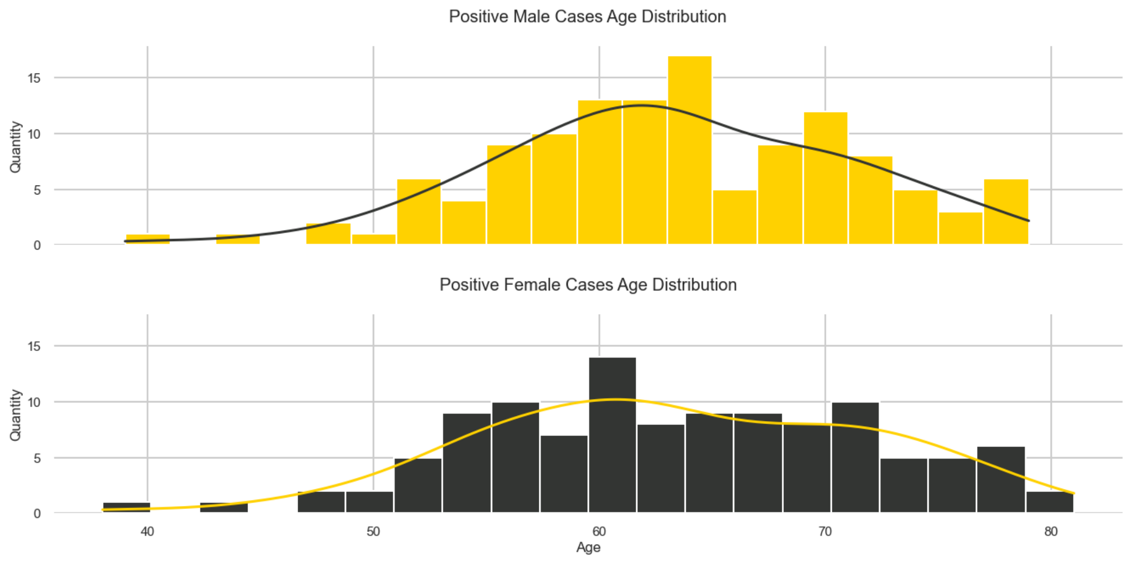
```
[11]: data_temp_pos = data_temp[data_temp["LUNG_CANCER"] == "YES"]
data_temp_pos.head().style.set_properties(**{"background-color":␣
 ↪"#2a9d8f","color":"white","border": "1.5px  solid black"})
```

```
[11]: <pandas.io.formats.style.Styler at 0x26f862f1790>
```

#

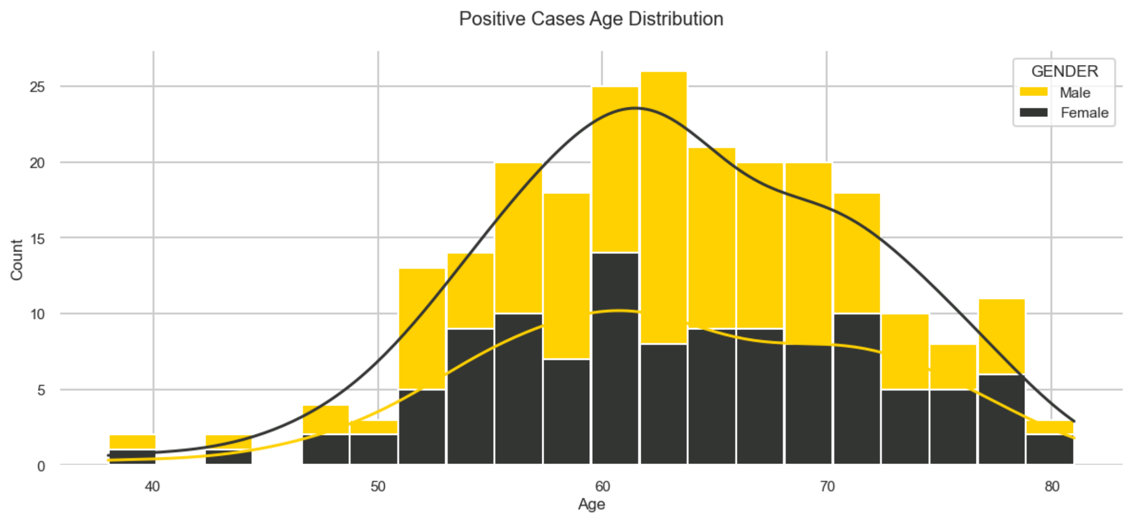
Positive Cases' Age Distribution

```
[12]: _, axs = plt.subplots(2,1,figsize=(20,10),sharex=True,sharey=True)
plt.tight_layout(pad=4.0)

sns.histplot(data_temp_pos[data_temp_pos["GENDER"]=="Male"]["AGE"],
color=palette[11],kde=True,ax=axs[0],bins=20,alpha=1,fill=True)
axs[0].lines[0].set_color(palette[12])
axs[0].set_title("\nPositive Male Cases Age Distribution\n",fontsize=20)
axs[0].set_xlabel("Age")
axs[0].set_ylabel("Quantity")

sns.histplot(data_temp_pos[data_temp_pos["GENDER"]=="Female"]["AGE"],
color=palette[12],kde=True,ax=axs[1],bins=20,alpha=1,fill=True)
axs[1].lines[0].set_color(palette[11])
axs[1].set_title("\nPositive Female Cases Age Distribution\n",fontsize=20)
axs[1].set_xlabel("Age")
axs[1].set_ylabel("Quantity")

sns.despine(left=True, bottom=True)
plt.show()
```



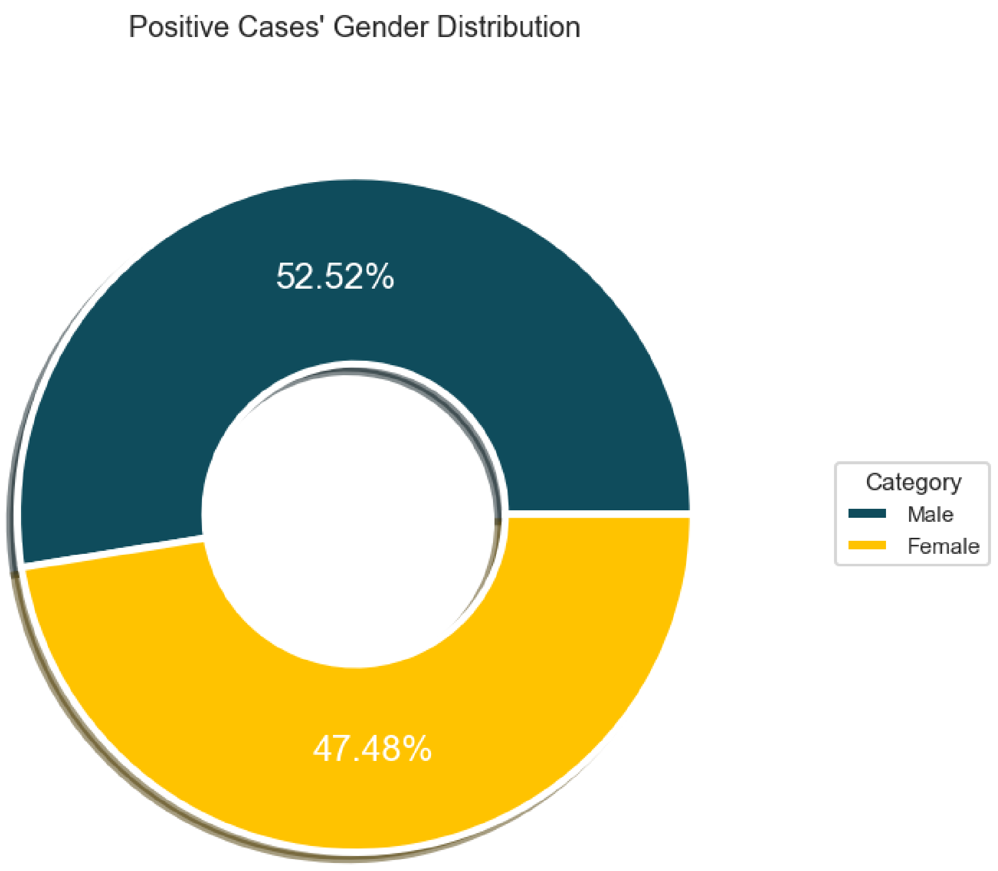
Let's stack them together in a same figure.

```
[13]: plt.subplots(figsize=(20, 8))
p = sns.
 ↪histplot(data=data_temp_pos,x="AGE",hue="GENDER",multiple="stack",palette=
palette[11:13],kde=True,shrink=.99,bins=20,alpha=1,fill=True)
p.axes.lines[0].set_color(palette[11])
p.axes.lines[1].set_color(palette[12])
p.axes.set_title("\nPositive Cases Age Distribution\n",fontsize=20)
plt.ylabel("Count")
plt.xlabel("Age")

sns.despine(left=True, bottom=True)
plt.show()
```



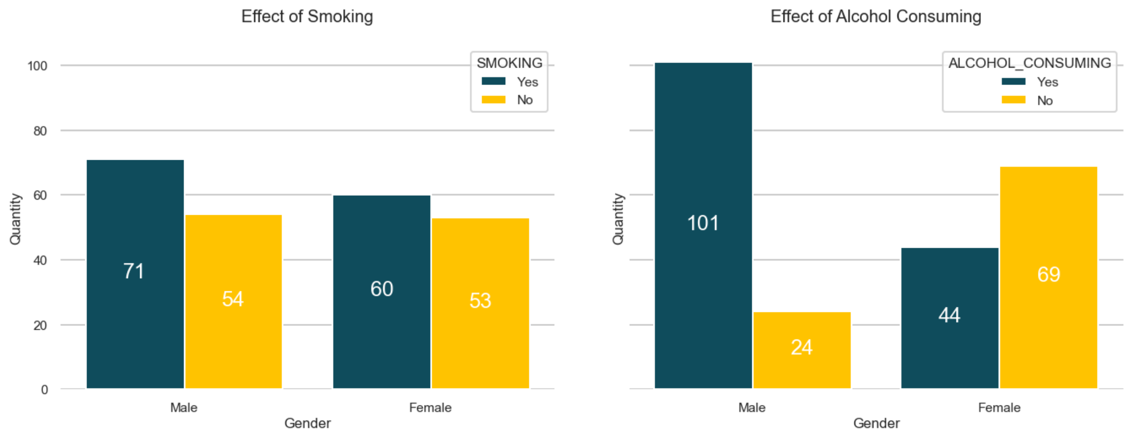
#

Positive Cases’ Gender Distribution



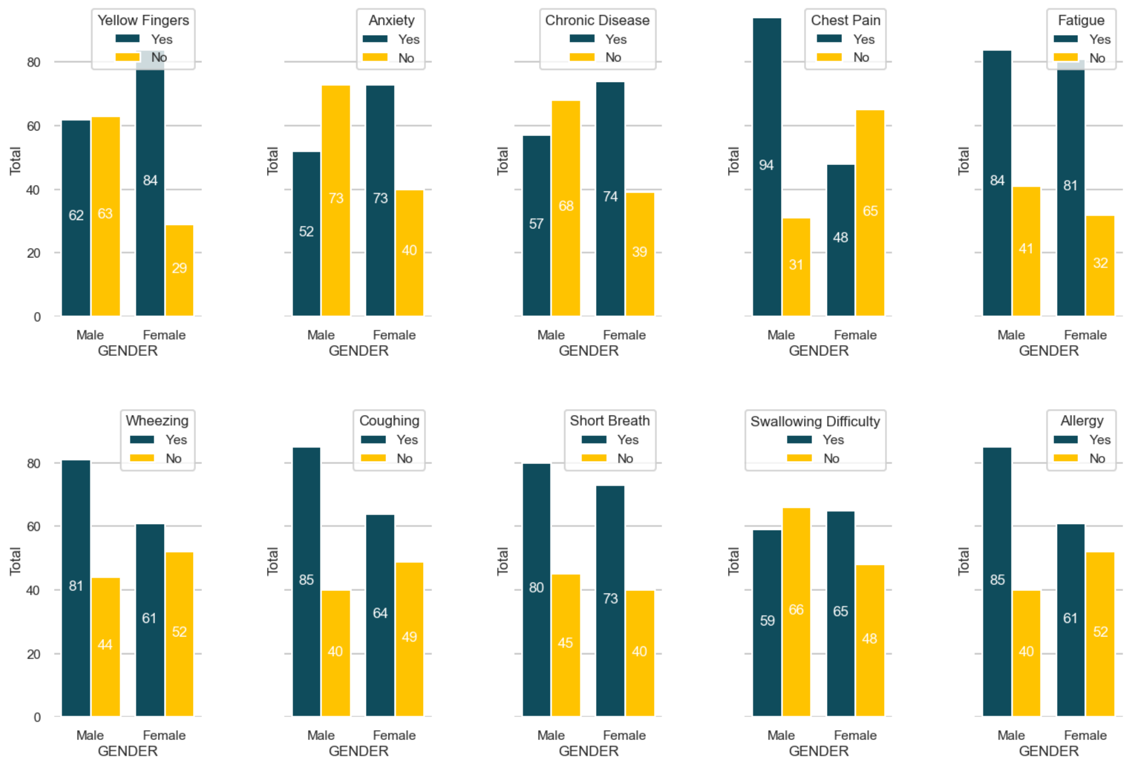
#

Gender-wise Positive Cases' Reasons



#

Gender-wise Positive Cases' Symptoms



#

Correlation Heatmap

Let's convert the target feature **"LUNG_CANCER"** from **Categorical** to **Numerical** data type by using **Label Encoder**. And converting the **"GENDER"** column from **Categorical** to **Numerical** data type by using **"One Hot Encoder"** for avoiding unexpected gender bias.

Preprocessing For Classification

[20]:
```
x = data.drop("LUNG_CANCER", axis = 1)
y = data["LUNG_CANCER"]

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

scaler = StandardScaler()
x = scaler.fit_transform(x)
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2,␣
 ↪random_state=42)

print(f"Shape of training data : {x_train.shape}, {y_train.shape}")
print(f"Shape of testing data : {x_test.shape}, {y_test.shape}")
```

```
Shape of training data : (247, 15), (247,)
Shape of testing data : (62, 15), (62,)
```

#

Logistic Regression Model

[21]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report


lr = LogisticRegression()
lr.fit(x_train, y_train)
lr_pred = lr.predict(x_test)
lr_conf = confusion_matrix(y_test, lr_pred)
lr_report = classification_report(y_test, lr_pred)
lr_acc = round(accuracy_score(y_test, lr_pred)*100, ndigits = 2)
print(f"Confusion Matrix : \n\n{lr_conf}")
print(f"\nClassification Report : \n\n{lr_report}")
print(f"\nThe Accuracy of Logistic Regression is {lr_acc} %")
```

```
Confusion Matrix :

[[ 1  1]
 [ 1 59]]

Classification Report :

              precision    recall  f1-score   support

           0       0.50      0.50      0.50         2
           1       0.98      0.98      0.98        60

    accuracy                           0.97        62
   macro avg       0.74      0.74      0.74        62
weighted avg       0.97      0.97      0.97        62


The Accuracy of Logistic Regression is 96.77 %
```

#

Gaussian Naive Bayes Model

[22]:
```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(x_train, y_train)
gnb_pred = gnb.predict(x_test)
gnb_conf = confusion_matrix(y_test, gnb_pred)
gnb_report = classification_report(y_test, gnb_pred)
gnb_acc = round(accuracy_score(y_test, gnb_pred)*100, ndigits = 2)
print(f"Confusion Matrix : \n\n{gnb_conf}")
print(f"\nClassification Report : \n\n{gnb_report}")
print(f"\nThe Accuracy of Gaussian Naive Bayes is {gnb_acc} %")
```

```
Confusion Matrix :

[[ 1  1]
 [ 3 57]]

Classification Report :

              precision    recall  f1-score   support

           0       0.25      0.50      0.33         2
           1       0.98      0.95      0.97        60

    accuracy                           0.94        62
   macro avg       0.62      0.72      0.65        62
weighted avg       0.96      0.94      0.95        62


The Accuracy of Gaussian Naive Bayes is 93.55 %
```

#

Bernoulli Naive Bayes Model

[23]:
```
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
bnb.fit(x_train, y_train)
bnb_pred = bnb.predict(x_test)
bnb_conf = confusion_matrix(y_test, bnb_pred)
bnb_report = classification_report(y_test, bnb_pred)
bnb_acc = round(accuracy_score(y_test, bnb_pred)*100, ndigits = 2)
print(f"Confusion Matrix : \n\n{bnb_conf}")
print(f"\nClassification Report : \n\n{bnb_report}")
print(f"\nThe Accuracy of Bernoulli Naive Bayes is {bnb_acc} %")
```

```
Confusion Matrix :
```

```
[[ 1  1]
 [ 2 58]]

Classification Report :

              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.98      0.97      0.97        60

    accuracy                           0.95        62
   macro avg       0.66      0.73      0.69        62
weighted avg       0.96      0.95      0.96        62


The Accuracy of Bernoulli Naive Bayes is 95.16 %
```

#

Support Vector Machine Model

```
[24]: from sklearn.svm import SVC

svm = SVC(C = 100, gamma = 0.002)
svm.fit(x_train, y_train)
svm_pred = svm.predict(x_test)
svm_conf = confusion_matrix(y_test, svm_pred)
svm_report = classification_report(y_test, svm_pred)
svm_acc = round(accuracy_score(y_test, svm_pred)*100, ndigits = 2)
print(f"Confusion Matrix : \n\n{svm_conf}")
print(f"\nClassification Report : \n\n{svm_report}")
print(f"\nThe Accuracy of Support Vector Machine is {svm_acc} %")
```

```
Confusion Matrix :

[[ 1  1]
 [ 1 59]]

Classification Report :

              precision    recall  f1-score   support

           0       0.50      0.50      0.50         2
           1       0.98      0.98      0.98        60

    accuracy                           0.97        62
   macro avg       0.74      0.74      0.74        62
weighted avg       0.97      0.97      0.97        62
```

```
The Accuracy of Support Vector Machine is 96.77 %
```

#

Random Forest Model

```
[25]: from sklearn.ensemble import RandomForestClassifier

rfg = RandomForestClassifier(n_estimators = 100, random_state = 42)
rfg.fit(x_train, y_train)
rfg_pred = rfg.predict(x_test)
rfg_conf = confusion_matrix(y_test, rfg_pred)
rfg_report = classification_report(y_test, rfg_pred)
rfg_acc = round(accuracy_score(y_test, rfg_pred)*100, ndigits = 2)
print(f"Confusion Matrix : \n\n{rfg_conf}")
print(f"\nClassification Report : \n\n{rfg_report}")
print(f"\nThe Accuracy of Random Forest Classifier is {rfg_acc} %")
```

```
Confusion Matrix :

[[ 1  1]
 [ 0 60]]

Classification Report :

              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.98      1.00      0.99        60

    accuracy                           0.98        62
   macro avg       0.99      0.75      0.83        62
weighted avg       0.98      0.98      0.98        62


The Accuracy of Random Forest Classifier is 98.39 %
```

#

K Nearest Neighbors Model

```
[26]: from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train, y_train)
knn_pred = knn.predict(x_test)
knn_conf = confusion_matrix(y_test, knn_pred)
knn_report = classification_report(y_test, knn_pred)
knn_acc = round(accuracy_score(y_test, knn_pred)*100, ndigits = 2)
```

```
print(f"Confusion Matrix : \n\n{knn_conf}")
print(f"\nClassification Report : \n\n{knn_report}")
print(f"\nThe Accuracy of K Nearest Neighbors Classifier is {knn_acc} %")
```

```
Confusion Matrix :

[[ 2  0]
 [ 6 54]]

Classification Report :

              precision    recall  f1-score   support

           0       0.25      1.00      0.40         2
           1       1.00      0.90      0.95        60

    accuracy                           0.90        62
   macro avg       0.62      0.95      0.67        62
weighted avg       0.98      0.90      0.93        62


The Accuracy of K Nearest Neighbors Classifier is 90.32 %
```

#

Extreme Gradient Boosting Model

[27]:
```
from xgboost import XGBClassifier

xgb = XGBClassifier(use_label_encoder = False)
xgb.fit(x_train, y_train)
xgb_pred = xgb.predict(x_test)
xgb_conf = confusion_matrix(y_test, xgb_pred)
xgb_report = classification_report(y_test, xgb_pred)
xgb_acc = round(accuracy_score(y_test, xgb_pred)*100, ndigits = 2)
print(f"Confusion Matrix : \n\n{xgb_conf}")
print(f"\nClassification Report : \n\n{xgb_report}")
print(f"\nThe Accuracy of Extreme Gradient Boosting Classifier is {xgb_acc} %")
```

```
Confusion Matrix :

[[ 1  1]
 [ 0 60]]

Classification Report :

              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.98      1.00      0.99        60

    accuracy                           0.98        62
   macro avg       0.99      0.75      0.83        62
weighted avg       0.98      0.98      0.98        62
```

```
The Accuracy of Extreme Gradient Boosting Classifier is 98.39 %
```

```
C:\Users\Acerr\anaconda3\lib\site-packages\xgboost\sklearn.py:1395: UserWarning:
`use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

#

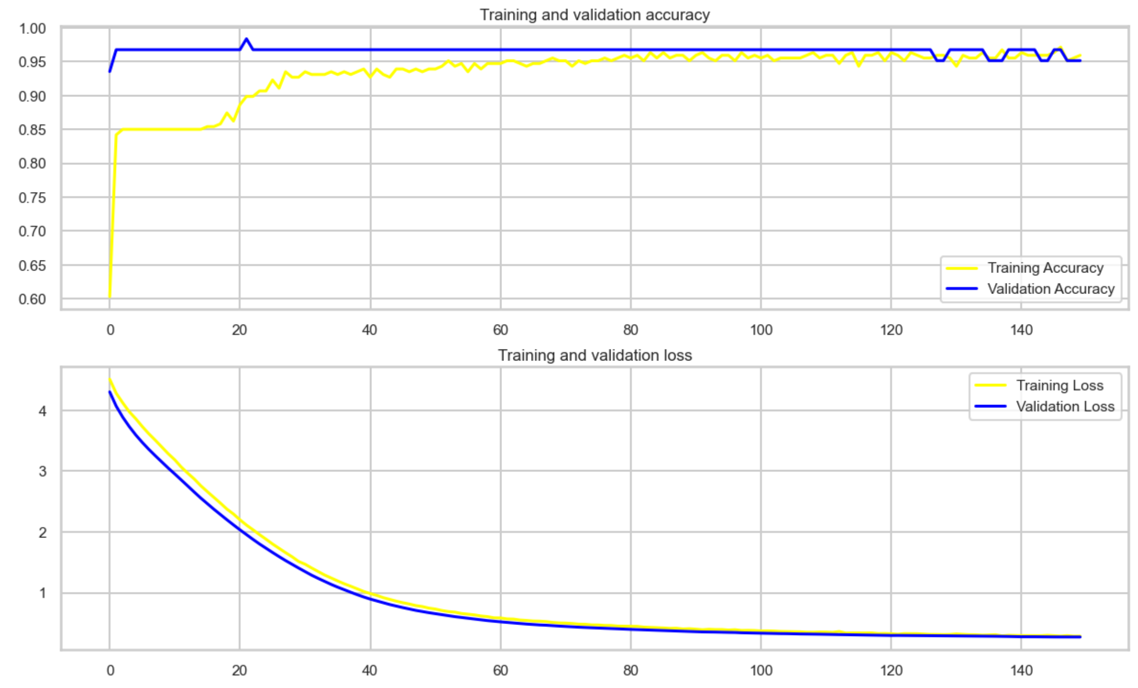
Neural Network Architecture

[28]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import Adam

regularization_parameter = 0.003

neural_model = Sequential([tf.keras.layers.Dense(units=32, input_dim=(x_train.
 ↪shape[-1]), activation="relu", kernel_regularizer = regularizers.
 ↪l1(regularization_parameter)),
                    tf.keras.layers.Dense(units=64, activation="relu",␣
 ↪kernel_regularizer = regularizers.l1(regularization_parameter)),
                    tf.keras.layers.Dense(units=128, activation="relu",␣
 ↪kernel_regularizer = regularizers.l1(regularization_parameter)),
                    tf.keras.layers.Dropout(0.3),
                    tf.keras.layers.Dense(units=16,activation="relu",␣
 ↪kernel_regularizer = regularizers.l1(regularization_parameter)),
                    tf.keras.layers.Dense(units=1, activation="sigmoid")
                    ])

print(neural_model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                512

 dense_1 (Dense)             (None, 64)                2112

 dense_2 (Dense)             (None, 128)               8320
 dropout (Dropout)           (None, 128)               0

 dense_3 (Dense)             (None, 16)                2064

 dense_4 (Dense)             (None, 1)                 17

=================================================================
Total params: 13,025
Trainable params: 13,025
Non-trainable params: 0
_________________________________________________________________
None
```

[29]:
```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get("accuracy") == 1.0):
            print("\nAccuracy is 100% so canceling training!")
            self.model.stop_training = True

callbacks = myCallback()


neural_model.compile(optimizer = Adam(learning_rate = 0.001),
                     loss = "binary_crossentropy",
                     metrics = ["accuracy"])

history = neural_model.fit(x_train, y_train,
                           epochs = 150,
                           verbose = 1,
                           batch_size = 64,
                           validation_data = (x_test, y_test),
                           callbacks = [callbacks])
```

[30]:
```
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

epochs = range(len(acc)) # number of epochs

plt.figure(figsize=(20, 12))
plt.subplot(2,1,1)
plt.plot(epochs, acc, "yellow", label= "Training Accuracy")
plt.plot(epochs, val_acc, "blue", label= "Validation Accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.subplot(2,1,2)
plt.plot(epochs, loss, "yellow", label= "Training Loss")
plt.plot(epochs, val_loss, "blue", label= "Validation Loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```



[31]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

[32]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

[33]:
```
df = pd.read_csv("E:/survey_lung_cancer2.csv")
df.head()
```

[33]: [34]:
```
df.tail()
```

[34]: [35]:
```
df.iloc[0]
```

[35]:
[36]:
```
df.shape
```

[36]:
```
(309, 16)
```

[ ]:
```

```

[37]:
```
df.columns
```

[37]:
```
Index(['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'],
      dtype='object')
```

[ ]:
```

```

[38]:
```
df.describe().T
```

[38]:
```
                       count       mean       std   min   25%   50%   75%  \
GENDER                 309.0   1.524272  0.500221   1.0   1.0   2.0   2.0
AGE                    309.0  62.673139  8.210301  21.0  57.0  62.0  69.0
SMOKING                309.0   1.563107  0.496806   1.0   1.0   2.0   2.0
YELLOW_FINGERS         309.0   1.569579  0.495938   1.0   1.0   2.0   2.0
ANXIETY                309.0   1.498382  0.500808   1.0   1.0   1.0   2.0
PEER_PRESSURE          309.0   1.501618  0.500808   1.0   1.0   2.0   2.0
CHRONIC DISEASE        309.0   1.504854  0.500787   1.0   1.0   2.0   2.0
FATIGUE                309.0   1.673139  0.469827   1.0   1.0   2.0   2.0
ALLERGY                309.0   1.556634  0.497588   1.0   1.0   2.0   2.0
WHEEZING               309.0   1.556634  0.497588   1.0   1.0   2.0   2.0
ALCOHOL CONSUMING      309.0   1.556634  0.497588   1.0   1.0   2.0   2.0
COUGHING               309.0   1.579288  0.494474   1.0   1.0   2.0   2.0
SHORTNESS OF BREATH    309.0   1.640777  0.480551   1.0   1.0   2.0   2.0
SWALLOWING DIFFICULTY  309.0   1.469256  0.499863   1.0   1.0   1.0   2.0
CHEST PAIN             309.0   1.556634  0.497588   1.0   1.0   2.0   2.0
LUNG_CANCER            309.0   1.873786  0.332629   1.0   2.0   2.0   2.0

                        max
GENDER                  2.0
AGE                    87.0
SMOKING                 2.0
YELLOW_FINGERS          2.0
ANXIETY                 2.0
PEER_PRESSURE           2.0
CHRONIC DISEASE         2.0
FATIGUE                 2.0
ALLERGY                 2.0
```

```
WHEEZING                 2.0
ALCOHOL CONSUMING        2.0
COUGHING                 2.0
SHORTNESS OF BREATH      2.0
SWALLOWING DIFFICULTY    2.0
CHEST PAIN               2.0
LUNG_CANCER              2.0
```

[39]:
```
df.info()
```

[40]:
```
print("missing values:")
df.isnull().sum()
```

```
missing values:
```

[41]:
```
# Print informative message about duplicated value statistics
print("Duplicated value statistics in our dataset: ")

df.duplicated().sum()
```

```
Duplicated value statistics in our dataset:
```

[41]:
```
33
```

[42]:
```
df.drop_duplicates(inplace=True)
```

[43]:
```
df.nunique()
```

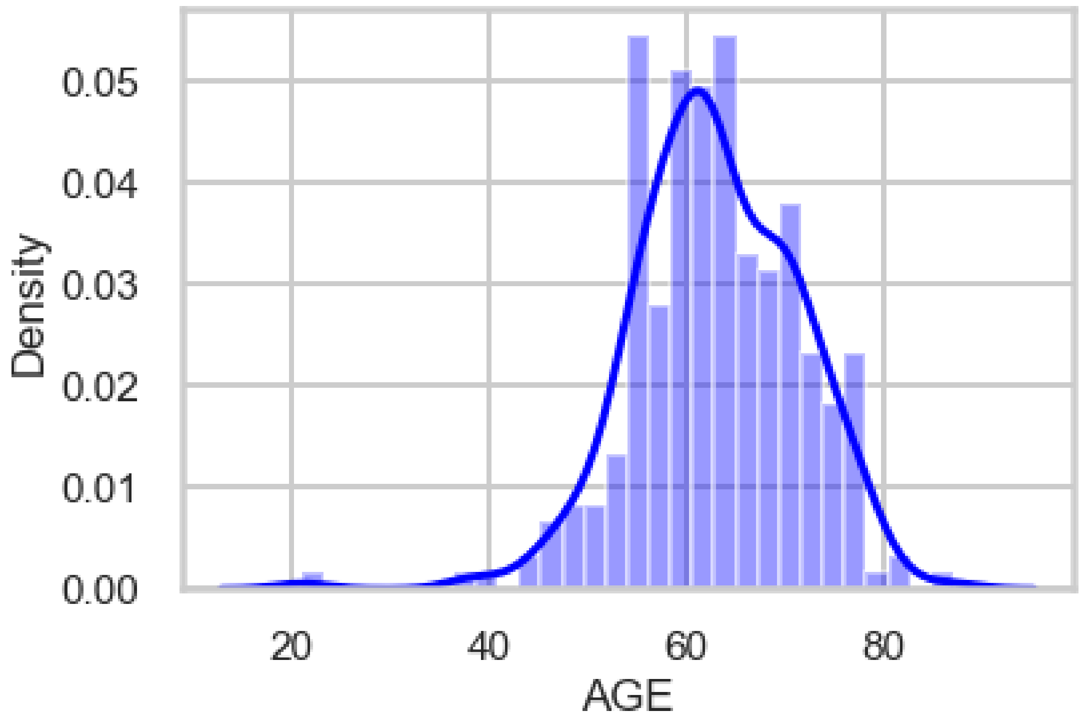
[44]:
```
df['GENDER'].value_counts()
```

[44]:
```
2    142
1    134
Name: GENDER, dtype: int64
```

[45]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from warnings import filterwarnings

# set the background style of the plot
sns.set_style('whitegrid')
sns.distplot(df['AGE'], kde = True, color ='blue', bins = 30)
```

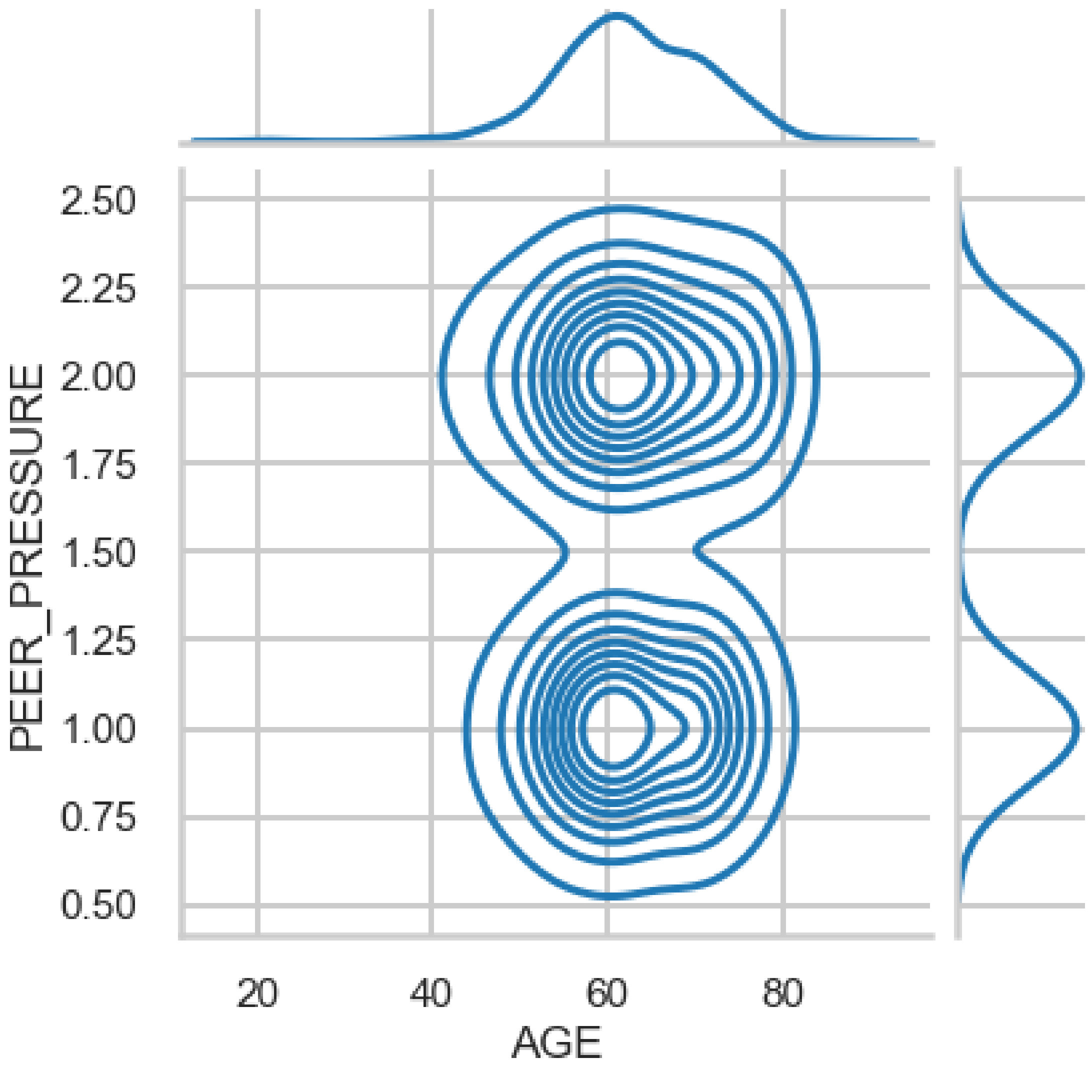
```
C:\Users\Acerr\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[45]:
```
<AxesSubplot:xlabel='AGE', ylabel='Density'>
```



[46]:
```
sns.jointplot(x ='AGE', y ='PEER_PRESSURE', data = df, kind ='kde')
```

[46]:
```
<seaborn.axisgrid.JointGrid at 0x26f969e5a60>
```



```
[47]: import plotly.express as px

      def plot_pie_charts(df):
          # Iterate over each column in the DataFrame
          for column in df.columns:
              # Count the frequency of each unique value in the column
              value_counts = df[column].value_counts()

              # Plot a pie chart for the distribution of values
              fig = px.pie(names=value_counts.index, values=value_counts.values,␣
       ↪title=f'Pie Chart for {column}')
              fig.show()
```

```
# Call the function with your DataFrame
plot_pie_charts(df)
```

[48]:
```
# Calculate the count of heart attacks (output) for each gender (sex)
male_lung_cancer_chances = df[(df['GENDER'] == 2) & (df['GENDER'] == 2)].shape[0]
female_lung_cancer_chances = df[(df['GENDER'] == 1) & (df['GENDER'] == 1)].
 ↪shape[0]
```

[49]:
```
# Calculate the count of no heart attacks (output) for each gender (sex)
male_no_lung_cancer_chances = df[(df['GENDER'] == 2) & (df['GENDER'] == 2)].
 ↪shape[0]
female_no_lung_cancer_chances = df[(df['GENDER'] == 1) & (df['GENDER'] == 1)].
 ↪shape[0]
```

[50]:
```
male_lung_cancer_chances
```

[50]:
```
142
```

[51]:
```
female_lung_cancer_chances
```

[51]:
```
134
```

[52]:
```
# Create data for the pie chart
labels = ['Male (lung cancer)', 'Female (lung cancer)', 'Male (No lung cancer)',␣
 ↪'Female (No lung cancer)']
sizes = [male_lung_cancer_chances, female_lung_cancer_chances,␣
 ↪male_no_lung_cancer_chances, female_no_lung_cancer_chances]
#sizes = ['1', '2']
```

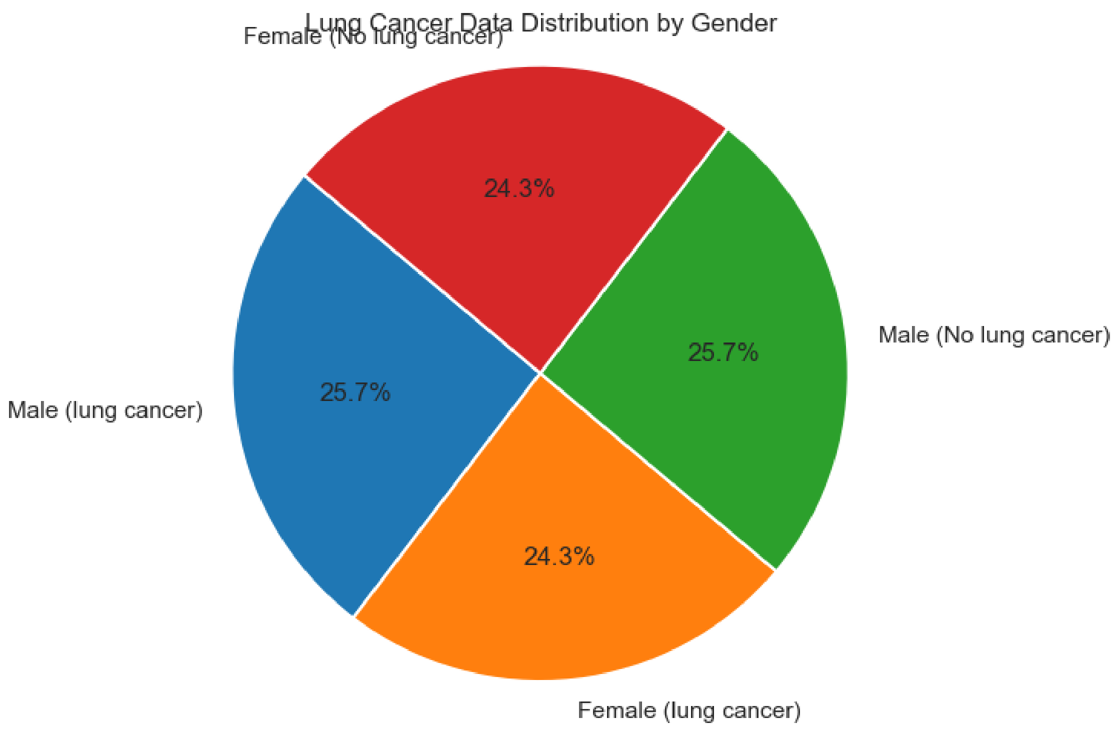
[53]:
```
labels
```

[53]:
```
['Male (lung cancer)',
 'Female (lung cancer)',
 'Male (No lung cancer)',
 'Female (No lung cancer)']
```

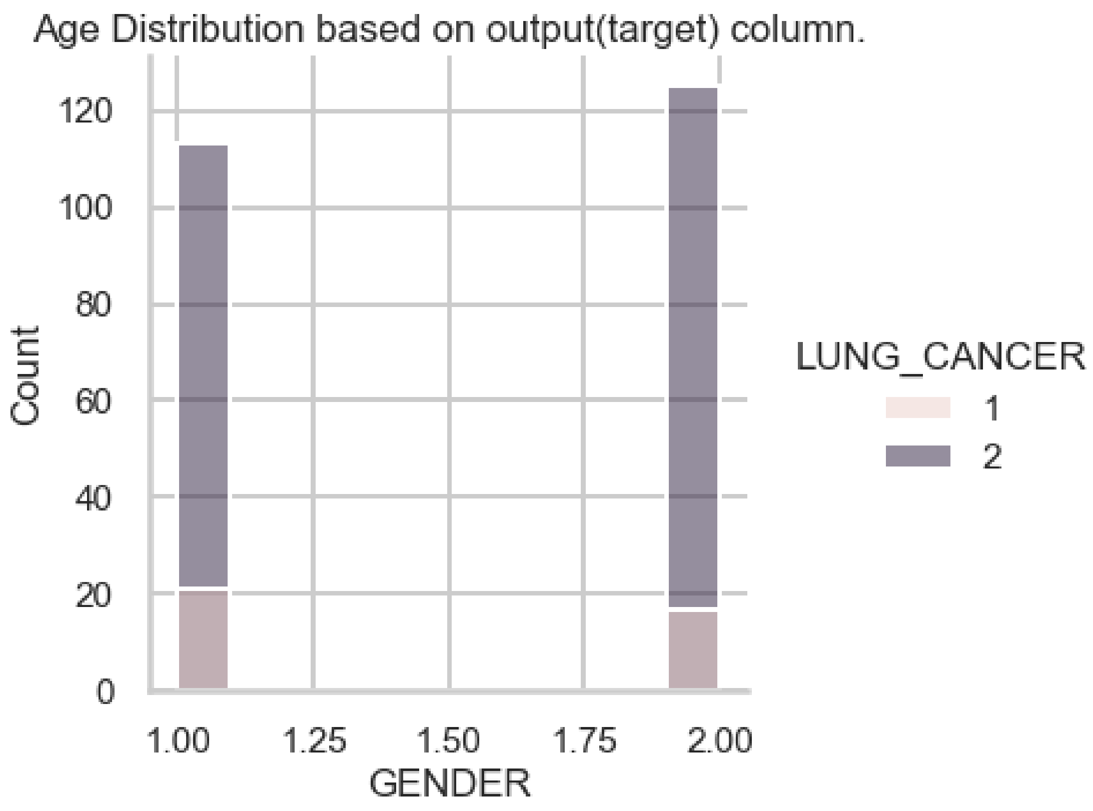
[54]:
```
sizes
```

[54]:
```
[142, 134, 142, 134]
```

[55]:
```
# Plot the pie chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Lung Cancer Data Distribution by Gender')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```



[56]:
```
sns.displot(df, x='GENDER', hue='LUNG_CANCER', color='red', bins='auto')
plt.title("Age Distribution based on output(target) column.")
plt.show()
```



```
[57]: df.head()
```

```
[57]:
[58]: # selecting the target feature
      X = df.drop('GENDER', axis=1)
      y = df['LUNG_CANCER']
```

```
[59]: import sklearn
      from sklearn.model_selection import train_test_split

      # spliting the data x and y
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
       ↪random_state=42)
```

```
[60]: from sklearn.preprocessing import MinMaxScaler, StandardScaler
      mms = MinMaxScaler()
      mms.fit_transform(X_train, y_train)
```

```
[60]: array([[0.44897959, 1.        , 1.        , ..., 1.        , 0.        ,
              1.        ],
             [0.        , 0.        , 1.        , ..., 0.        , 1.        ,
              1.        ],
             [0.59183673, 1.        , 1.        , ..., 1.        , 1.        ,
              1.        ],
             ...,
             [0.2244898 , 0.        , 0.        , ..., 0.        , 0.        ,
              1.        ],
             [0.79591837, 1.        , 1.        , ..., 1.        , 0.        ,
              1.        ],
             [0.69387755, 1.        , 1.        , ..., 1.        , 1.        ,
              1.        ]])
```

[61]:
```
ss = StandardScaler()
ss.fit_transform(X_train, y_train)
```

[61]:
```
array([[-0.40445505,  0.94686415,  0.85573563, ...,  1.08545743,
        -1.13651514,  0.36608827],
       [-3.14230461, -1.05611771,  0.85573563, ..., -0.92127059,
         0.87988269,  0.36608827],
       [ 0.4666789 ,  0.94686415,  0.85573563, ...,  1.08545743,
         0.87988269,  0.36608827],
       ...,
       [-1.77337983, -1.05611771, -1.16858521, ..., -0.92127059,
        -1.13651514,  0.36608827],
       [ 1.71115597,  0.94686415,  0.85573563, ...,  1.08545743,
        -1.13651514,  0.36608827],
       [ 1.08891744,  0.94686415,  0.85573563, ...,  1.08545743,
         0.87988269,  0.36608827]])
```

[62]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

#Boosting Algorithms
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import confusion_matrix,accuracy_score, precision_score,␣
 ↪recall_score, f1_score
```

[63]:
```
rfc = RandomForestClassifier()
scores = cross_val_score(rfc, X, y, cv=5)

# Train the model
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Confusion Matrix:
 [[12  0]
 [ 0 44]]
```

[64]:
```
knc = KNeighborsClassifier(n_neighbors=100)
knc.fit(X_train, y_train)

y_pred = knc.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.7857142857142857
Precision: 0.0
Recall: 0.0
F1-score: 0.0
Confusion Matrix:
 [[ 0 12]
 [ 0 44]]
```

```
C:\Users\Acerr\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning:
```

```
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
```

[65]:
```python
svm = SVC()
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.7857142857142857
Precision: 0.0
Recall: 0.0
F1-score: 0.0
Confusion Matrix:
 [[ 0 12]
 [ 0 44]]
```

```
C:\Users\Acerr\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1327: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
```

[66]:
```python
gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)

y_pred = gbc.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Confusion Matrix:
 [[12  0]
 [ 0 44]]
```

[67]:
```
adc = AdaBoostClassifier()
adc.fit(X_train, y_train)

y_pred = adc.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Confusion Matrix:
 [[12  0]
 [ 0 44]]
```

[68]:
```
# import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
# instantiate the DecisionTreeClassifier model with criterion gini index
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)
# fit the model
clf_gini.fit(X_train, y_train)
```

```
y_pred_gini = clf_gini.predict(X_test)
from sklearn.metrics import accuracy_score
print('Model accuracy score with criterion gini index: {0:0.4f}'.␣
 ↪format(accuracy_score(y_test, y_pred_gini)))
y_pred_train_gini = clf_gini.predict(X_train)
y_pred_train_gini
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train,␣
 ↪y_pred_train_gini)))
# print the scores on training and test set
print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
plt.figure(figsize=(12,8))
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Model accuracy score with criterion gini index: 1.0000
Training-set accuracy score: 1.0000
Training set score: 1.0000
Test set score: 1.0000
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Confusion Matrix:
 [[12  0]
 [ 0 44]]

<Figure size 864x576 with 0 Axes>
```

[69]:
```
# Define classifiers
classifiers = {
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(n_neighbors=100),
    "SVM": SVC(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "Decision Tree": DecisionTreeClassifier()
}
```

```
# Split the data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize variables to store the best algorithm and its performance
best_algorithm = None
best_accuracy = 0

# Loop through each classifier
for name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Print performance metrics
    print(f"{name} - Accuracy: {accuracy:.4f}")

    # Update best algorithm if the current one has higher accuracy
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_algorithm = name

print(f"\nBest Algorithm: {best_algorithm} with Accuracy: {best_accuracy:.4f}")
```

```
Random Forest - Accuracy: 1.0000
KNN - Accuracy: 0.7857
SVM - Accuracy: 0.7857
Gradient Boosting - Accuracy: 1.0000
Decision Tree - Accuracy: 1.0000

Best Algorithm: Random Forest with Accuracy: 1.0000
```

[ ]:

```
```

[70]:

```
import pandas as pd

#df = pd.read_excel('https://github.com/datagy/mediumdata/raw/master/
↪sample_pivot.xlsx', parse_dates=['Date'])
df = pd.read_csv('E:/survey_lung_cancer.csv')
df.head()
print(df.head())
#https://datagy.io/pandas-crosstab/
```

[71]:
```
pd.crosstab(df.GENDER, df.LUNG_CANCER)
#pd.crosstab(index = df.GENDER, columns = df.LUNG_CANCER)
```

[71]:
```
LUNG_CANCER  NO  YES
GENDER
F            22  125
M            17  145
```

[72]:
```
from scipy.stats import chi2_contingency

# Define the observed frequencies
observed = [[22, 125], [17, 145]]

# Perform the chi-square test
chi2_stat, p_val, dof, expected = chi2_contingency(observed)

print("Chi-square statistic:", chi2_stat)
print("P-value:", p_val)
print("Degrees of freedom:", dof)
print("Expected frequencies table:")
print(expected)
```

```
Chi-square statistic: 1.0215453687014526
P-value: 0.31215272362842433
Degrees of freedom: 1
Expected frequencies table:
[[ 18.55339806 128.44660194]
 [ 20.44660194 141.55339806]]
```

[73]:
```
R = pd.crosstab(df.GENDER, df.LUNG_CANCER)
# Define the observed frequencies
observed = R

# Perform the chi-square test
chi2_stat, p_val, dof, expected = chi2_contingency(observed)

print("Chi-square statistic:", chi2_stat)
print("P-value:", p_val)
print("Degrees of freedom:", dof)
print("Expected frequencies table:")
print(expected)
```

```
Chi-square statistic: 1.0215453687014526
P-value: 0.31215272362842433
Degrees of freedom: 1
Expected frequencies table:
[[ 18.55339806 128.44660194]
 [ 20.44660194 141.55339806]]
```

```
[74]: R1 = pd.crosstab(df.GENDER, df.SMOKING)
      # Define the observed frequencies
      observed = R1

      # Perform the chi-square test
      chi2_stat, p_val, dof, expected = chi2_contingency(observed)

      print("Chi-square statistic:", chi2_stat)
      print("P-value:", p_val)
      print("Degrees of freedom:", dof)
      print("Expected frequencies table:")
      print(expected)
```

```
Chi-square statistic: 0.2733828816044669
P-value: 0.6010714811865476
Degrees of freedom: 1
Expected frequencies table:
[[64.22330097 82.77669903]
 [70.77669903 91.22330097]]
```

Similarly, we have applied the Chi-square test for remaining Symptoms and the results obtained as follows:

For YELLOW FINGERS -

```
Chi-square statistic: 13.165694159157173
P-value: 0.0002851211460262936
Degrees of freedom: 1
Expected frequencies table:
[[63.27184466 83.72815534]
 [69.72815534 92.27184466]]
```

For ANXIETY -

```
Chi-square statistic: 6.55478150972719
P-value: 0.010460268269699964
Degrees of freedom: 1
Expected frequencies table:
[[73.73786408 73.26213592]
 [81.26213592 80.73786408]]
```

For PEER PRESSURE -

```
Chi-square statistic: 22.373541288460657
P-value: 2.244448794124298e-06
Degrees of freedom: 1
Expected frequencies table:
[[73.26213592 73.73786408]
 [80.73786408 81.26213592]]
```

For CHRONIC DISEASE -

```
Chi-square statistic: 12.129361514311332
P-value: 0.0004963411060069395
Degrees of freedom: 1
Expected frequencies table:
[[72.78640777 74.21359223]
 [80.21359223 81.78640777]]
```

For FATIGUE -

```
Chi-square statistic: 1.8155842235028048
P-value: 0.17783979904693004
Degrees of freedom: 1
Expected frequencies table:
[[ 48.04854369  98.95145631]
 [ 52.95145631 109.04854369]]
```

For ALLERGY -

```
Chi-square statistic: 6.743558227200801
P-value: 0.009408678086155932
Degrees of freedom: 1
Expected frequencies table:
[[65.17475728 81.82524272]
 [71.82524272 90.17475728]]
```

For WHEEZING -

```
Chi-square statistic: 5.60524518116532
P-value: 0.017906789057283688
Degrees of freedom: 1
Expected frequencies table:
[[65.17475728 81.82524272]
 [71.82524272 90.17475728]]
```

For ALCOHOL CONSUMING -

```
Chi-square statistic: 61.94714687139498
P-value: 3.5280060834605535e-15
Degrees of freedom: 1
Expected frequencies table:
[[65.17475728 81.82524272]
 [71.82524272 90.17475728]]
```

For COUGHING -

```
Chi-square statistic: 4.963429198495046
P-value: 0.025888819041797448
Degrees of freedom: 1
Expected frequencies table:
[[61.84466019 85.15533981]
 [68.15533981 93.84466019]]
```

For SHORTNESS OF BREATH -

```
Chi-square statistic: 1.045123754449151
P-value: 0.3066328090486877
Degrees of freedom: 1
Expected frequencies table:
[[ 52.80582524  94.19417476]
 [ 58.19417476 103.80582524]]
```

For SWALLOWING DIFFICULTY -

```
Chi-square statistic: 1.5871491005990321
P-value: 0.20773392112696337
Degrees of freedom: 1
Expected frequencies table:
[[78.01941748 68.98058252]
 [85.98058252 76.01941748]]
```

For CHEST PAIN -

```
Chi-square statistic: 39.257454092795385
P-value: 3.7144759539674507e-10
Degrees of freedom: 1
Expected frequencies table:
[[65.17475728 81.82524272]
 [71.82524272 90.17475728]]
```

[87]:

```
import numpy as np
from statsmodels.stats.proportion import proportions_ztest
count = 142
nobs = 309
value =.05
stat, pval = proportions_ztest(count, nobs, value)
print('{0:0.3f}'.format(pval))

if pval < value:
    print("Reject the null hyptohesis: There is a signifciant difference between␣
 ↪the proportions")
else:
    print("Fail to reject the null hyptohesis: There is no signifciant␣
 ↪difference between the proportions")
```

```
0.000
Reject the null hyptohesis: There is a signifciant difference between the
proportions
```

[88]:

```
import numpy as np
from statsmodels.stats.proportion import proportions_ztest
count = np.array([142, 134])
nobs = np.array([309, 309])
stat, pval = proportions_ztest(count, nobs)
print('{0:0.3f}'.format(pval))
```

```
if pval < value:
    print("Reject the null hyptohesis: There is a signifciant difference between␣
 ↪the proportions")
else:
    print("Fail to reject the null hyptohesis: There is no signifciant␣
 ↪difference between the proportions")
```

```
0.517
Fail to reject the null hyptohesis: There is no signifciant difference between
the proportions
```

5 RESULT

In this study, following are the important results obtained from the analysis of the lung cancer dataset:

1. Initially there were 309 observations. Among that, 33 observations are found to be duplicate. Finally, there are 276 observations having 142 male and 134 female patients.

2. The chance of a lung cancer in males was 52.52 %.

3. The chance of a lung cancer in females was 47.48 %.

4. Those who are addicted with smoking they can easily affect with lung cancer.

5. It is observed that, alcohol consumption majorly affect on male patients whereas in the female patients it is least affected on them.

6. Among all classifier, Random forest classifier is the best with accuracy 1.

6 CONCLUSION

This study aims to address to critical issue of lung cancer disease by utilizing python. Through the analysis and extraction of insights from selected data sets, we have gained better understanding of the prevalence and risk factors associated with lung cancer. By identifying patterns, trends and relationships within the data, we have been able to identify most preferable classifier for the effectiveness of lung cancer prediction system. The finding from our study has significant implications for public health. By raising awareness about the risk factors and preventive measures, we can minimize or eliminate the probability of mis-classification of lung cancer, ultimately improving lung health and well-being in the population. Moreover, we have applied Chi-square test to test the genderwise significance with various parameters. We observed that, Yellow fingers, anxiety, peer pressure, allergy, wheezing, alcohol consumption, coughing, chest pain symptoms are significant. Thereafter, proportion test is applied to test the significance regarding lung cancer male and female patients and results reveals that it is insignificant.

Finally, we conclude that, based on significant symptoms if we can do the treatments at the earlier stage, then it will help to reduce the severity of the disease.

7 REFERENCES

1. H.Frank, E.,Hall, M.A., Pal, C.J.(2016) data mining fourth edition: Practical Machine Learning tools and techniques. SanFrancisco: Morgan Kaufmann Publishers Inc.

2. V.K.Kapoor and S.C.Gupta fundamental and Statistics.Maimon,O.,Rokach,L.(Eds) .(2005). Data Mining and Knowledge Discovery handbook.

3. Data set Link:"https://www.kaggle.com/code/hasibalmuzdadid/lung-cancer-analysis-accuracy-96-4/input".