UTM UNIVERSITI TEKNOLOGI MALAYSIA | FACULTY OF COMPUTING UTM Johor Bahru

Project - Prediction of Lung Cancer

SECB3203 Programming for Bioinformatics

Semester 1, 2025/2026

Section 01

Group 11

NAME	MATRIC NUMBER
JAYADHANYA A/P VIJAYALINGAM	A23CS0092
FADLIN AFINA BT DAUD	A23CS0299
ALDANISHA MUADZ	A23CS0039

Lecturer: Dr. Sean

Date: 14 January 2025

Table of Content

1.0 Introduction

Lung cancer is one of the most common and deadly types of cancer worldwide. Early detection is crucial for improving patient survival rates. Recently, the combination of bioinformatics and machine learning has changed medical diagnostics, providing new ways for accurate and timely disease prediction. This project proposes developing a machine learning-based prediction system for lung cancer using data on patient demographics and symptoms. By using different classification algorithms, we aim to create a strong model that can help with early risk assessment and possibly lower mortality rates related to this disease.

1.1 Problem Background

Lung cancer is the major cause of cancer-related mortality, accounting for around 2.21 million new cases and 1.8 million deaths globally each year. According to Malaysia's National Cancer Registry, lung cancer accounts for 10.5% of all cancer cases in men and 4.5% in women. Due to late stage diagnosis, five-year survival rates are still below 20%. Traditional diagnostic tools, including biopsies and imaging studies, are often used only after symptoms appear, frequently missing the best window for intervention when treatment would be most successful.

The use of machine learning algorithms on clinical data has emerged as a potential method for predicting early risk. Numerous studies have shown the promise of various algorithmic approaches, ranging from advanced ensemble techniques to traditional statistical methods. For example, Random Forest has demonstrated 85-90% accuracy rates in comparable clinical prediction tests, and gradient boosting techniques such as XGBoost have performed even better in some medical classification tasks. However, thorough comparative analysis with the same dataset and uniform evaluation standards are less frequent, especially when it comes to lung cancer prediction.

Developing successful prediction systems requires an understanding of which algorithms work best for this particular clinical application. The trade-offs between predicted accuracy, computational efficiency, interpretability, and robustness to data features like feature correlations or class imbalance differ among algorithms. The development of clinically relevant tools for early lung cancer detection may be accelerated by using a systematic comparison to help researchers and clinicians choose the best methods for comparable medical prediction tasks.

1.2 Problem Statement

1. Which machine learning algorithm predicts lung cancer most accurately?
2. How much can we improve prediction accuracy by comparing and optimizing different algorithms?
3. Which patient features are most important for accurate lung cancer prediction?

1.3 Objectives

1. To compare different machine learning algorithms for lung cancer prediction
2. To find the best algorithm and improve it through optimization
3. To identify the most important features for accurate prediction
4. To create a reliable model that can help in early lung cancer detection

1.4 Scopes

1. Comparing multiple algorithms to find the best one
2. Focusing on binary classification (has lung cancer or not)
3. Implementing in Python with scikit-learn library
4. Evaluating based on accuracy and other important metrics

1.5 Conclusion

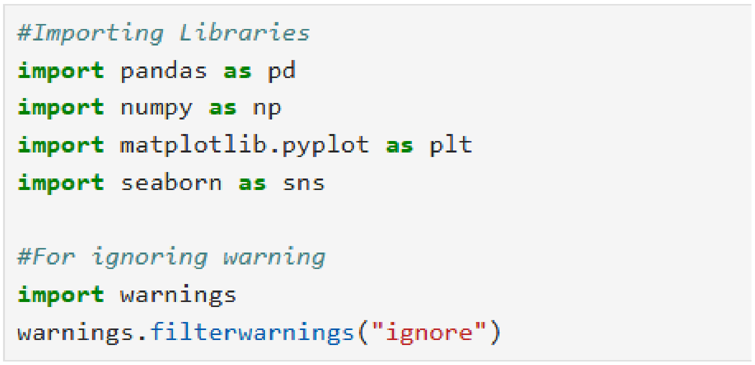
Finally, this study will discover the best effective machine learning method for lung cancer prediction by systematically comparing several classification models. A chosen high-performance algorithm that can reliably identify patients at risk of lung cancer based on their symptoms and demographic information will be the result. The knowledge of machine learning applications in medical diagnostics and their possible incorporation into clinical practice will also be enhanced by insights into algorithm performance, feature significance, and optimization strategies.

2.0 Data Collection and pre-processing

2.1 Importing Dataset

2.1.1 Import Packages

The necessary packages such as pandas and numpy are imported. These packages are commonly used in the analysis and manipulation of data in Python

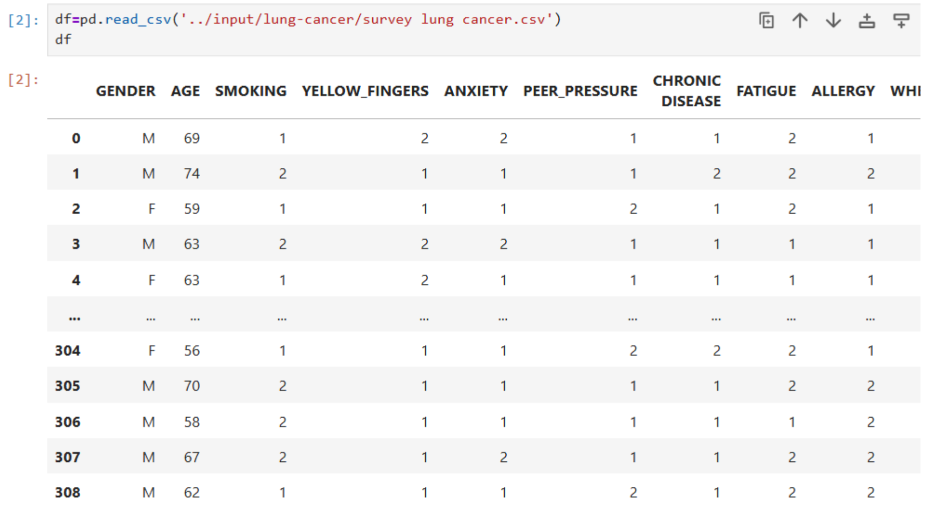


```python
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

Figure 2.1 Import Packages

2.1.2 Import Dataset

The lung cancer dataset is imported by reading a CSV file. The print(df) method is used to display the imported dataset.

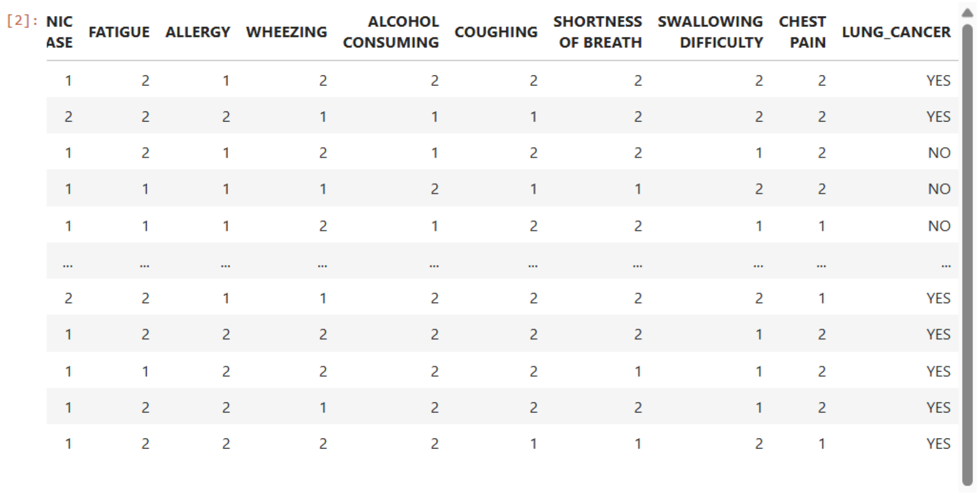


```
[2]: df=pd.read_csv('../input/lung-cancer/survey lung cancer.csv')
     df
```

[2]:

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

Figure 2.2 Import Dataset



[2]:

NIC ASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_CANCER
1	2	1	2	2	2	2	2	2	YES
2	2	2	1	1	1	2	2	2	YES
1	2	1	2	1	2	2	1	2	NO
1	1	1	1	2	1	1	2	2	NO
1	1	1	2	1	2	2	1	1	NO
...	...	...	...	...	...	...	...	...	...
2	2	1	1	2	2	2	2	1	YES
1	2	2	2	2	2	2	1	2	YES
1	1	2	2	2	2	1	1	2	YES
1	2	2	1	2	2	2	1	2	YES
1	2	2	2	2	1	1	2	1	YES

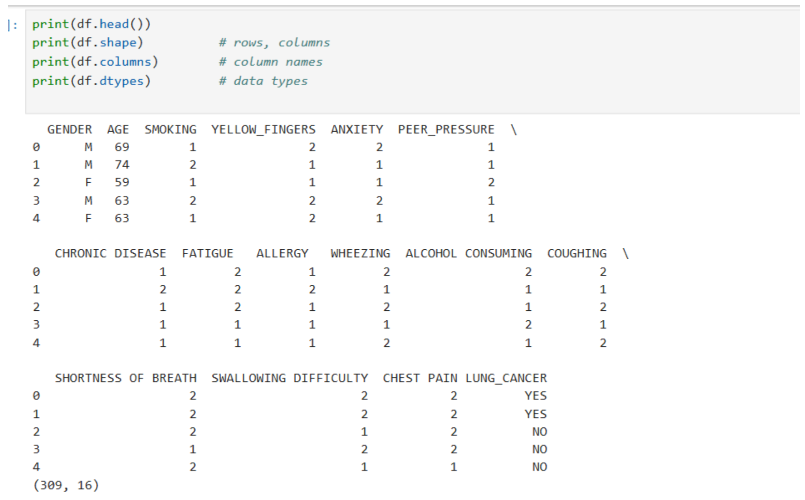
Figure 2.3 Import Dataset

2.2 Data wrangling

2.2.1 Data Formatting

The dataset was first inspected to ensure that all attributes were successfully imported. The df.head() method was used to display the first five records, allowing an initial overview of the data structure and attribute values. The total number of rows and columns was then identified using df.shape, which shows that the dataset contains 309 rows and 16 columns, confirming successful data import.

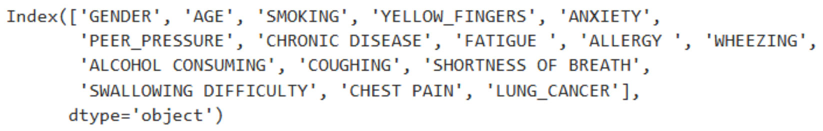
Next, the column names were displayed using df.columns to verify attribute naming and consistency. Lastly, the data type of each attribute was examined using df.dtypes. The results indicate that most attributes are stored as integer values, while GENDER and LUNG_CANCER are categorical variables stored as object data types. This confirms that the dataset is properly formatted and ready for further preprocessing and analysis.



```
]: print(df.head())
   print(df.shape)          # rows, columns
   print(df.columns)        # column names
   print(df.dtypes)         # data types

  GENDER  AGE  SMOKING  YELLOW_FINGERS  ANXIETY  PEER_PRESSURE  \
0      M   69        1               2        2              1
1      M   74        2               1        1              1
2      F   59        1               1        1              2
3      M   63        2               2        2              1
4      F   63        1               2        1              1

   CHRONIC DISEASE  FATIGUE   ALLERGY   WHEEZING  ALCOHOL CONSUMING  COUGHING  \
0                1         2         1         2                  2         2
1                2         2         2         1                  1         1
2                1         2         1         2                  1         2
3                1         1         1         1                  2         1
4                1         1         1         2                  1         2

   SHORTNESS OF BREATH  SWALLOWING DIFFICULTY  CHEST PAIN LUNG_CANCER
0                    2                      2           2         YES
1                    2                      2           2         YES
2                    2                      1           2          NO
3                    1                      2           2          NO
4                    2                      1           1          NO
(309, 16)
```

Figure 2.? Attributes

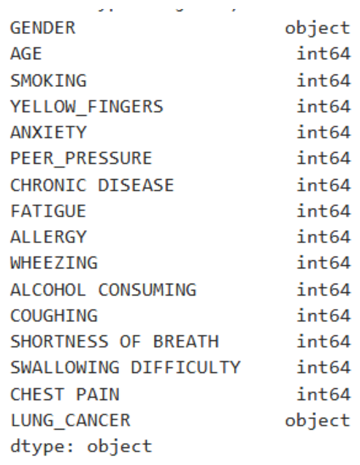
The columns of the dataset are then displayed using df.columns method



```
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')
```

Figure 2.4 Columns of the dataset

The datatype foreach attribute is identified using df.dtypes method



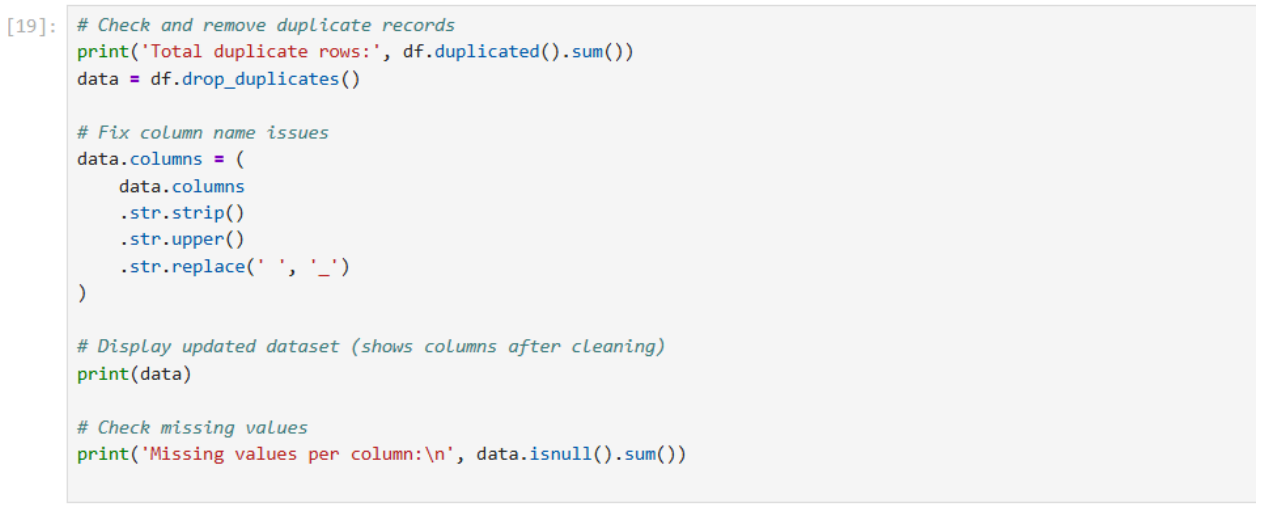
```
GENDER                    object
AGE                        int64
SMOKING                    int64
YELLOW_FINGERS             int64
ANXIETY                    int64
PEER_PRESSURE              int64
CHRONIC DISEASE            int64
FATIGUE                    int64
ALLERGY                    int64
WHEEZING                   int64
ALCOHOL CONSUMING          int64
COUGHING                   int64
SHORTNESS OF BREATH        int64
SWALLOWING DIFFICULTY      int64
CHEST PAIN                 int64
LUNG_CANCER               object
dtype: object
```

Figure 2.5 Datatype for each attribute

2.2.2 Data Duplication Identifying and Handling

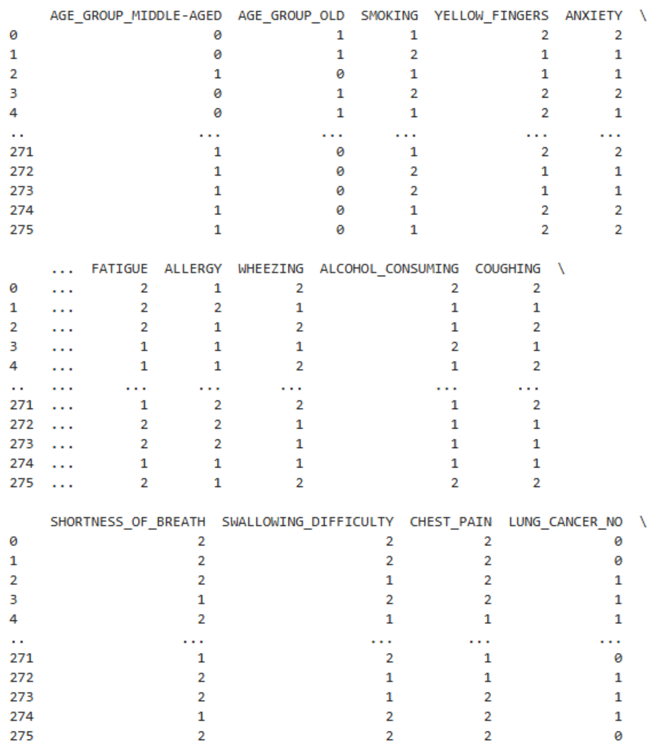
Duplicate records in the dataset were identified using the df.duplicated() method. The total number of duplicated rows was calculated using df.duplicated().sum(), which showed that 33 duplicate records were present in the dataset. To prevent redundancy and bias during analysis, these duplicated records were removed using the drop_duplicates() method.

After removing duplicates, the dataset was cleaned further by standardizing the column names. Leading and trailing spaces were removed, all column names were converted to uppercase, and spaces were replaced with underscores to ensure consistency and avoid errors during processing. The cleaned dataset was then checked for missing values using the isnull().sum() method. The results indicate that no missing values are present in any of the attributes, confirming that the dataset is complete and suitable for further preprocessing and exploratory data analysis.



```
[19]: # Check and remove duplicate records
      print('Total duplicate rows:', df.duplicated().sum())
      data = df.drop_duplicates()

      # Fix column name issues
      data.columns = (
          data.columns
          .str.strip()
          .str.upper()
          .str.replace(' ', '_')
      )

      # Display updated dataset (shows columns after cleaning)
      print(data)

      # Check missing values
      print('Missing values per column:\n', data.isnull().sum())
```

Figure 2.6 Data Duplication Identifying and Handling

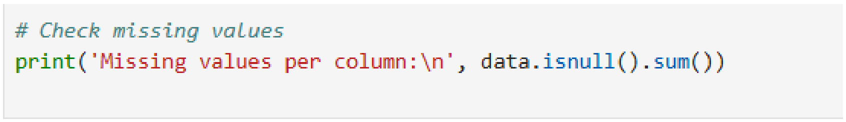


```
     AGE_GROUP_MIDDLE-AGED  AGE_GROUP_OLD  SMOKING  YELLOW_FINGERS  ANXIETY  \
0                        0              1        1               2        2
1                        0              1        2               1        1
2                        1              0        1               1        1
3                        0              1        2               2        2
4                        0              1        1               2        1
..                     ...            ...      ...             ...      ...
271                      1              0        1               2        2
272                      1              0        2               1        1
273                      1              0        2               1        1
274                      1              0        1               2        2
275                      1              0        1               2        2

     ...  FATIGUE  ALLERGY  WHEEZING  ALCOHOL_CONSUMING  COUGHING  \
0    ...        2        1         2                  2         2
1    ...        2        2         1                  1         1
2    ...        2        1         2                  1         2
3    ...        1        1         1                  2         1
4    ...        1        1         2                  1         2
..   ...      ...      ...       ...                ...       ...
271  ...        1        2         2                  1         2
272  ...        2        2         1                  1         1
273  ...        2        2         1                  1         1
274  ...        1        1         1                  1         1
275  ...        2        1         2                  2         2

     SHORTNESS_OF_BREATH  SWALLOWING_DIFFICULTY  CHEST_PAIN  LUNG_CANCER_NO  \
0                      2                      2           2               0
1                      2                      2           2               0
2                      2                      1           2               1
3                      1                      2           2               1
4                      2                      1           1               1
..                   ...                    ...         ...             ...
271                    1                      2           1               0
272                    2                      1           1               1
273                    2                      1           2               1
274                    1                      2           2               1
275                    2                      2           2               0
```

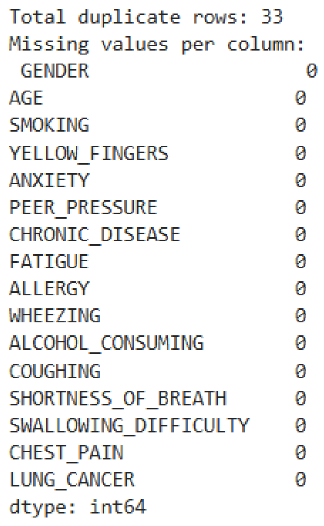
Figure 2.7 Columns of the dataset

2.2.3 Missing Values Identifying and Handling

The missing values are identified using the method, .isnull().sum(). The output shows that there are no null values in the dataset.



```python
# Check missing values
print('Missing values per column:\n', data.isnull().sum())
```



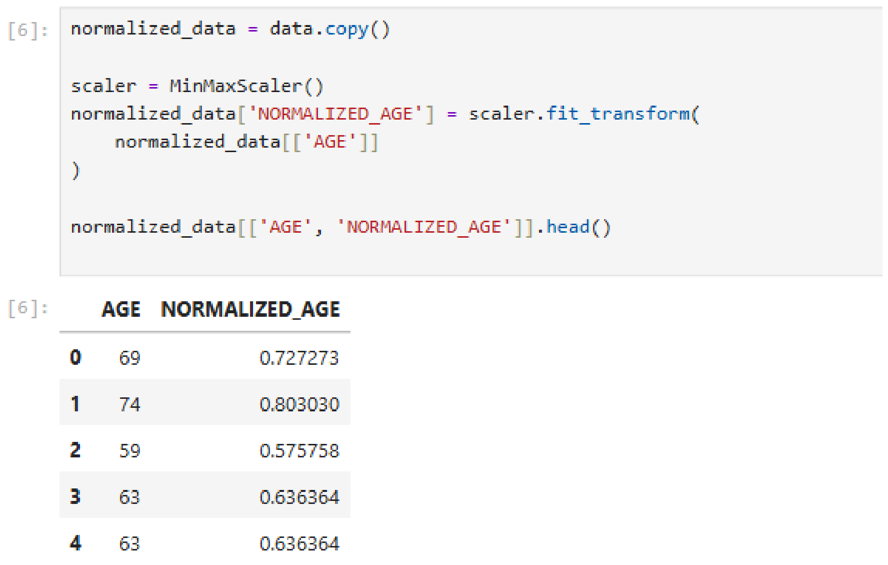
```
Total duplicate rows: 33
Missing values per column:
 GENDER                   0
AGE                      0
SMOKING                  0
YELLOW_FINGERS           0
ANXIETY                  0
PEER_PRESSURE            0
CHRONIC_DISEASE          0
FATIGUE                  0
ALLERGY                  0
WHEEZING                 0
ALCOHOL_CONSUMING        0
COUGHING                 0
SHORTNESS_OF_BREATH      0
SWALLOWING_DIFFICULTY    0
CHEST_PAIN               0
LUNG_CANCER              0
dtype: int64
```

Figure 2.8 Missing Values Identifying and Handling

2.2.4 Data Normalization

Data normalization was applied to the AGE attribute to scale the values into a common range for easier comparison and analysis. Since age is a ratio variable with varying magnitudes, Min–Max normalization was selected as the normalization technique. This method rescales the data to a range between 0 and 1 using the minimum and maximum values of the attribute.

A new attribute named NORMALIZED_AGE was generated using the MinMaxScaler() method, while the original AGE attribute was retained for reference. As shown in Figure 2.x, the normalized values preserve the relative differences between ages, with younger ages mapped closer to 0 and older ages mapped closer to 1. This normalization step helps improve interpretability and ensures that the age attribute does not dominate subsequent analyses due to scale differences.



```
[6]: normalized_data = data.copy()

     scaler = MinMaxScaler()
     normalized_data['NORMALIZED_AGE'] = scaler.fit_transform(
         normalized_data[['AGE']]
     )

     normalized_data[['AGE', 'NORMALIZED_AGE']].head()
```

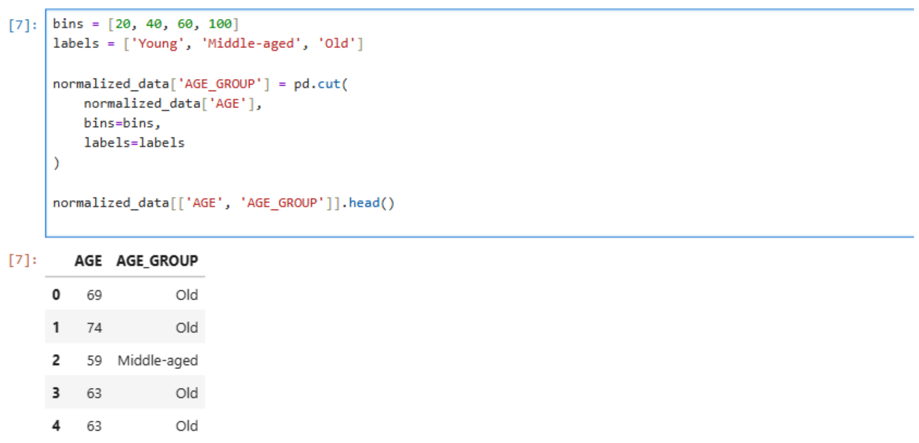
```
[6]:    AGE  NORMALIZED_AGE
     0   69        0.727273
     1   74        0.803030
     2   59        0.575758
     3   63        0.636364
     4   63        0.636364
```

Figure 2.9 Data Normalization

2.2.5 Data Binning

Data binning was applied to the AGE attribute to group continuous age values into meaningful categories. The pd.cut() method was used to divide the age values into predefined intervals (bins), namely 20–40, 40–60, and 60–100, which were labeled as Young, Middle-aged, and Old respectively.

A new attribute called AGE_GROUP was created to represent these categories. As shown in the output, each age value is correctly assigned to its corresponding age group, simplifying analysis and enabling better interpretation of age-related patterns in the dataset.



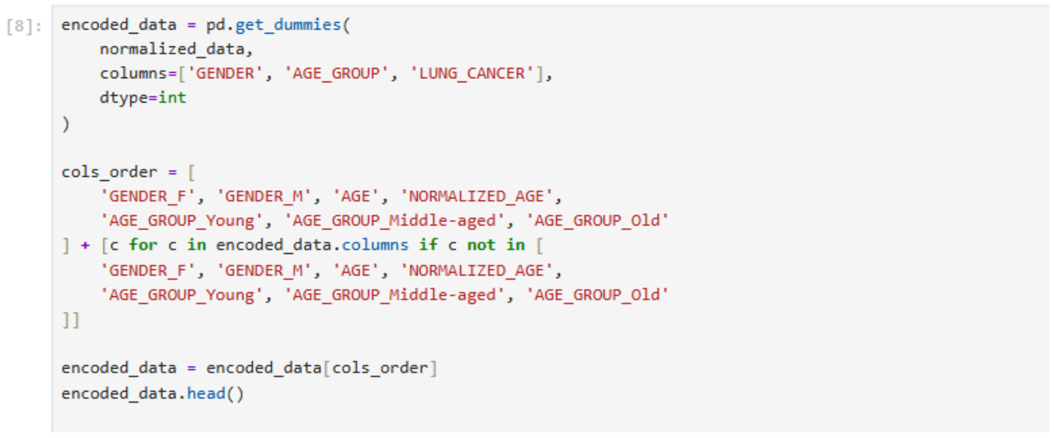
```
[7]: bins = [20, 40, 60, 100]
     labels = ['Young', 'Middle-aged', 'Old']

     normalized_data['AGE_GROUP'] = pd.cut(
         normalized_data['AGE'],
         bins=bins,
         labels=labels
     )

     normalized_data[['AGE', 'AGE_GROUP']].head()

[7]:    AGE  AGE_GROUP
     0   69          Old
     1   74          Old
     2   59  Middle-aged
     3   63          Old
     4   63          Old
```

Figure 2.10 Data Binning

2.2.6 Indicator Variable

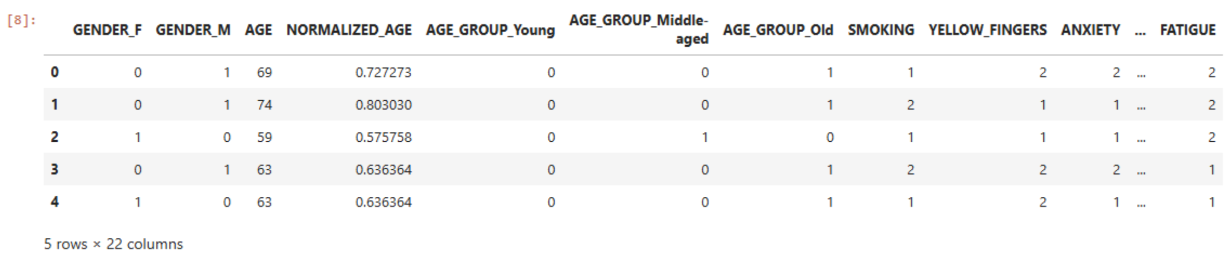
Indicator variables were created to convert categorical attributes into numerical form for analysis. The get_dummies() method was applied to the GENDER, AGE_GROUP, and LUNG_CANCER attributes, transforming each category into separate binary columns represented by values of 0 and 1.

After encoding, the columns were rearranged to improve readability and logical organization. Gender indicators, age-related attributes, and age group indicators were placed at the beginning of the dataset, followed by the remaining health-related variables. As shown in the output, each record is now represented entirely in numeric form, making the dataset suitable for statistical analysis and machine learning applications.



```
[8]: encoded_data = pd.get_dummies(
         normalized_data,
         columns=['GENDER', 'AGE_GROUP', 'LUNG_CANCER'],
         dtype=int
     )

     cols_order = [
         'GENDER_F', 'GENDER_M', 'AGE', 'NORMALIZED_AGE',
         'AGE_GROUP_Young', 'AGE_GROUP_Middle-aged', 'AGE_GROUP_Old'
     ] + [c for c in encoded_data.columns if c not in [
         'GENDER_F', 'GENDER_M', 'AGE', 'NORMALIZED_AGE',
         'AGE_GROUP_Young', 'AGE_GROUP_Middle-aged', 'AGE_GROUP_Old'
     ]]

     encoded_data = encoded_data[cols_order]
     encoded_data.head()
```

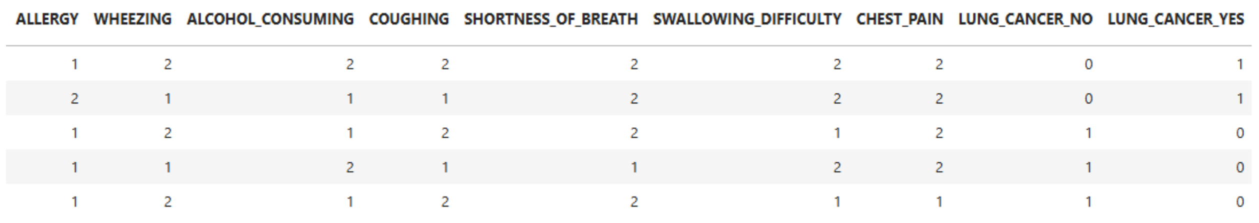
Figure 2.11 Indicator Variable


[8]:

	GENDER_F	GENDER_M	AGE	NORMALIZED_AGE	AGE_GROUP_Young	AGE_GROUP_Middle-aged	AGE_GROUP_Old	SMOKING	YELLOW_FINGERS	ANXIETY	...	FATIGUE
0	0	1	69	0.727273	0	0	1	1	2	2	...	2
1	0	1	74	0.803030	0	0	1	2	1	1	...	2
2	1	0	59	0.575758	0	1	0	1	1	1	...	2
3	0	1	63	0.636364	0	0	1	2	2	2	...	1
4	1	0	63	0.636364	0	0	1	1	2	1	...	1

5 rows × 22 columns

Figure 2.12 Columns of the dataset

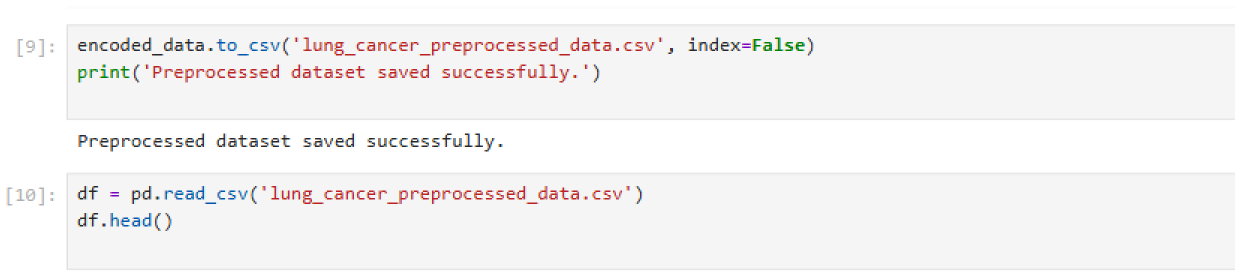


ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	SHORTNESS_OF_BREATH	SWALLOWING_DIFFICULTY	CHEST_PAIN	LUNG_CANCER_NO	LUNG_CANCER_YES
1	2	2	2	2	2	2	0	1
2	1	1	1	2	2	2	0	1
1	2	1	2	2	1	2	1	0
1	1	2	1	1	2	2	1	0
1	2	1	2	2	1	1	1	0

Figure 2.13 Columns of the dataset

2.2.7 Export preprocessed data

After completing all preprocessing steps, the finalized dataset was exported as a CSV file using the to_csv() method. The file was saved without row indices to maintain a clean data structure. This exported dataset serves as the input for subsequent exploratory data analysis and modeling, ensuring that all preprocessing steps are preserved and reproducible.



```
[9]:  encoded_data.to_csv('lung_cancer_preprocessed_data.csv', index=False)
      print('Preprocessed dataset saved successfully.')

      Preprocessed dataset saved successfully.

[10]: df = pd.read_csv('lung_cancer_preprocessed_data.csv')
      df.head()
```

Figure 2.14 Export preprocessed data

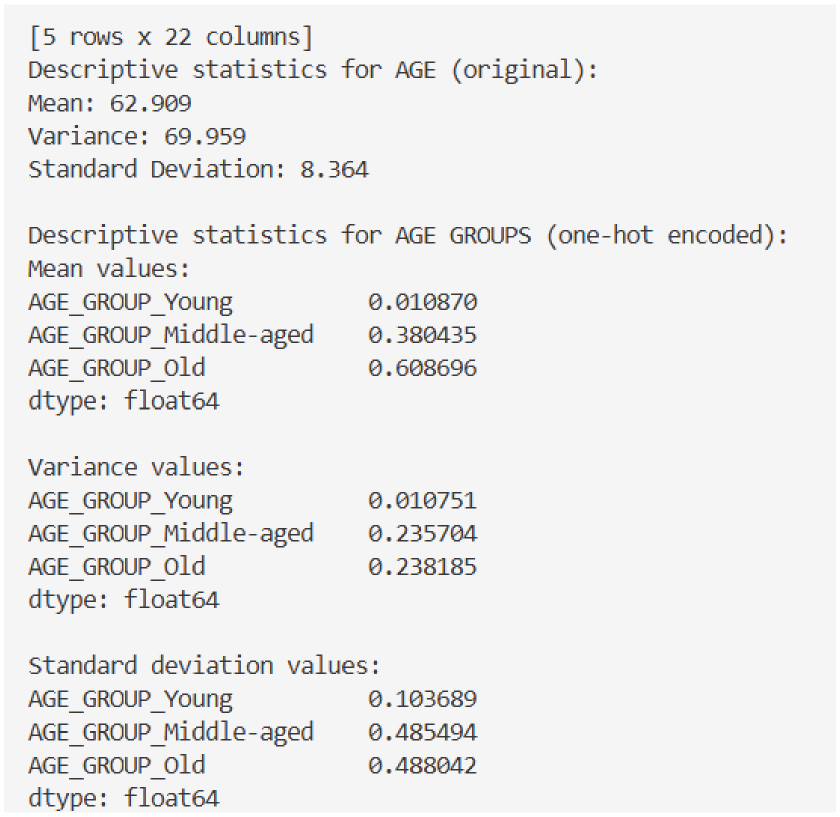
2.2.8 Descriptive Analysis

Firstly, we load our pre-processed dataset in the format of a CSV file into the python file. The dataset appears as the output in the diagram below, and we also remove any unrelated columns. At the same time, we drop an unrelated column in the dataset.

```
Preprocessed dataset saved successfully.
   GENDER_F  GENDER_M  AGE  NORMALIZED_AGE  AGE_GROUP_Young  ...  SHORTNESS OF BREATH  SWALLOWING DIFFICULTY  CHEST PAIN  LUNG_CANCER_NO  LUNG_CANCER_YES
0         0         1   69        0.727273                0  ...                    2                      2           2               0                1
1         0         1   74        0.803030                0  ...                    2                      2           2               0                1
2         1         0   59        0.575758                0  ...                    2                      1           2               1                0
3         0         1   63        0.636364                0  ...                    1                      2           2               1                0
4         1         0   63        0.636364                0  ...                    2                      1           1               1                0
```

2.2.9 Mean, Variance, and Standard Deviation of AGE

```python
######## AGE ########

# Calculate mean, variance, and standard deviation for original AGE
age_mean = df[age_numeric_col].mean()
age_variance = df[age_numeric_col].var(ddof=0)
age_std = df[age_numeric_col].std(ddof=0)

print('Descriptive statistics for AGE (original):')
print(f"Mean: {age_mean:.3f}")
print(f"Variance: {age_variance:.3f}")
print(f"Standard Deviation: {age_std:.3f}")

######## AGE GROUPS ########

# Calculate mean, variance, and standard deviation for age groups
age_group_means = df[age_group_cols].mean()
age_group_variance = df[age_group_cols].var(ddof=0)
age_group_std = df[age_group_cols].std(ddof=0)

print('\nDescriptive statistics for AGE GROUPS (one-hot encoded):')
print('Mean values:')
print(age_group_means)
print('\nVariance values:')
print(age_group_variance)
print('\nStandard deviation values:')
print(age_group_std)
```



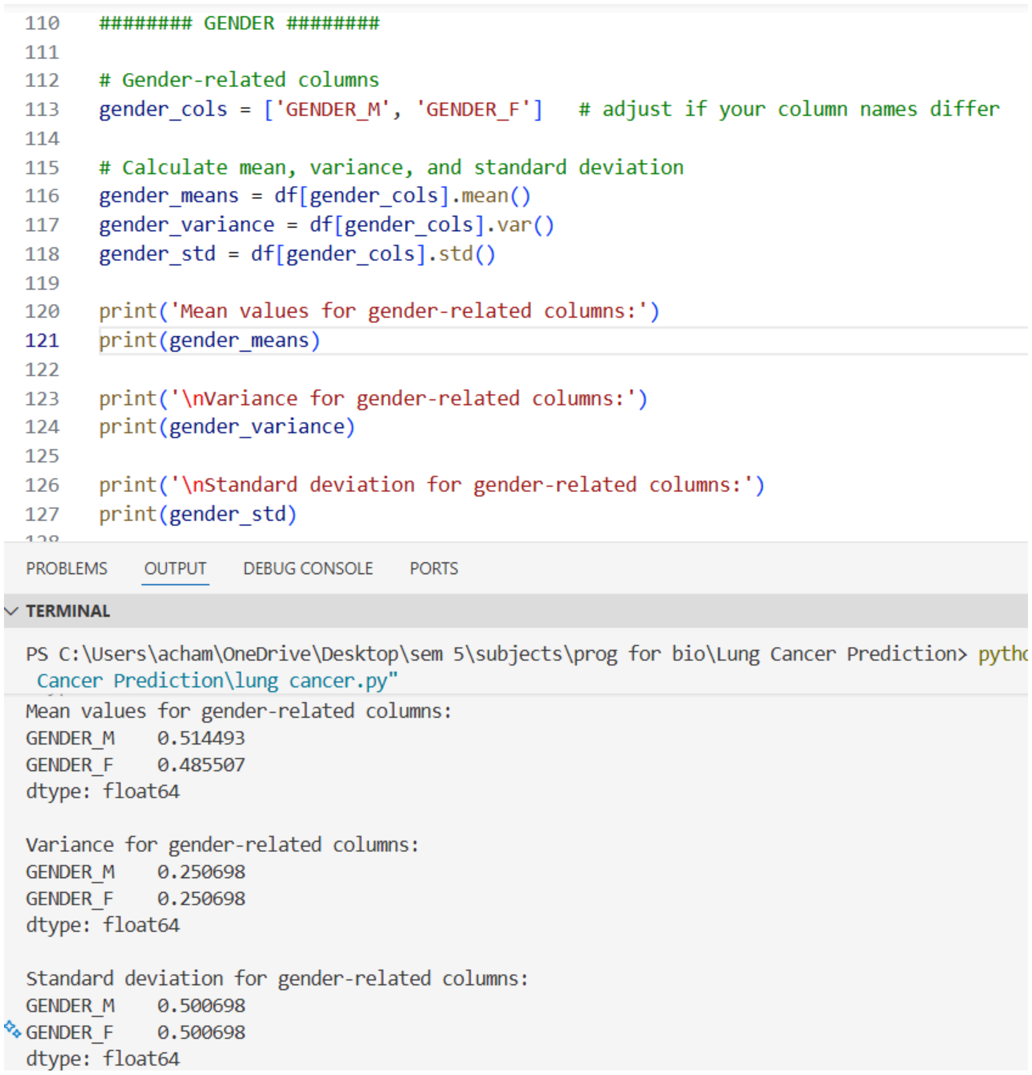
```
[5 rows x 22 columns]
Descriptive statistics for AGE (original):
Mean: 62.909
Variance: 69.959
Standard Deviation: 8.364

Descriptive statistics for AGE GROUPS (one-hot encoded):
Mean values:
AGE_GROUP_Young          0.010870
AGE_GROUP_Middle-aged    0.380435
AGE_GROUP_Old            0.608696
dtype: float64

Variance values:
AGE_GROUP_Young          0.010751
AGE_GROUP_Middle-aged    0.235704
AGE_GROUP_Old            0.238185
dtype: float64

Standard deviation values:
AGE_GROUP_Young          0.103689
AGE_GROUP_Middle-aged    0.485494
AGE_GROUP_Old            0.488042
dtype: float64
```

Mean:
The average age of the patients is 62.91 years, indicating that most of the patients in this dataset are older adults. For the age groups, most patients fall in the ‘Old’ category (60.87%), while a smaller portion is middle-aged (38.04%) and very few are young (1.09%).

Variance:
The variance of 69.96 for AGE shows moderate spread around the mean. Among the one-hot encoded age groups, the variance is higher for the middle-aged (0.236) and old (0.238) categories, reflecting more variation in these groups compared to the young group (0.011).

Standard Deviation:
The standard deviation of AGE is 8.36 years, meaning that most patients' ages deviate from the mean by roughly 8 years. Similarly, the standard deviation of the age groups shows the young group is tightly clustered (0.104), while middle-aged (0.485) and old (0.488) have wider variation, consistent with the spread seen in variance.

2.2.10 Mean, Variance, and Standard Deviation of GENDER



```python
######## GENDER ########

# Gender-related columns
gender_cols = ['GENDER_M', 'GENDER_F']   # adjust if your column names differ

# Calculate mean, variance, and standard deviation
gender_means = df[gender_cols].mean()
gender_variance = df[gender_cols].var()
gender_std = df[gender_cols].std()

print('Mean values for gender-related columns:')
print(gender_means)

print('\nVariance for gender-related columns:')
print(gender_variance)

print('\nStandard deviation for gender-related columns:')
print(gender_std)
```

```
PS C:\Users\acham\OneDrive\Desktop\sem 5\subjects\prog for bio\Lung Cancer Prediction> pytho
 Cancer Prediction\lung cancer.py"
Mean values for gender-related columns:
GENDER_M    0.514493
GENDER_F    0.485507
dtype: float64

Variance for gender-related columns:
GENDER_M    0.250698
GENDER_F    0.250698
dtype: float64

Standard deviation for gender-related columns:
GENDER_M    0.500698
GENDER_F    0.500698
dtype: float64
```

Mean:
The mean values show that 51.45% of patients are male and 48.55% are female, showing a nearly balanced gender distribution in the dataset.

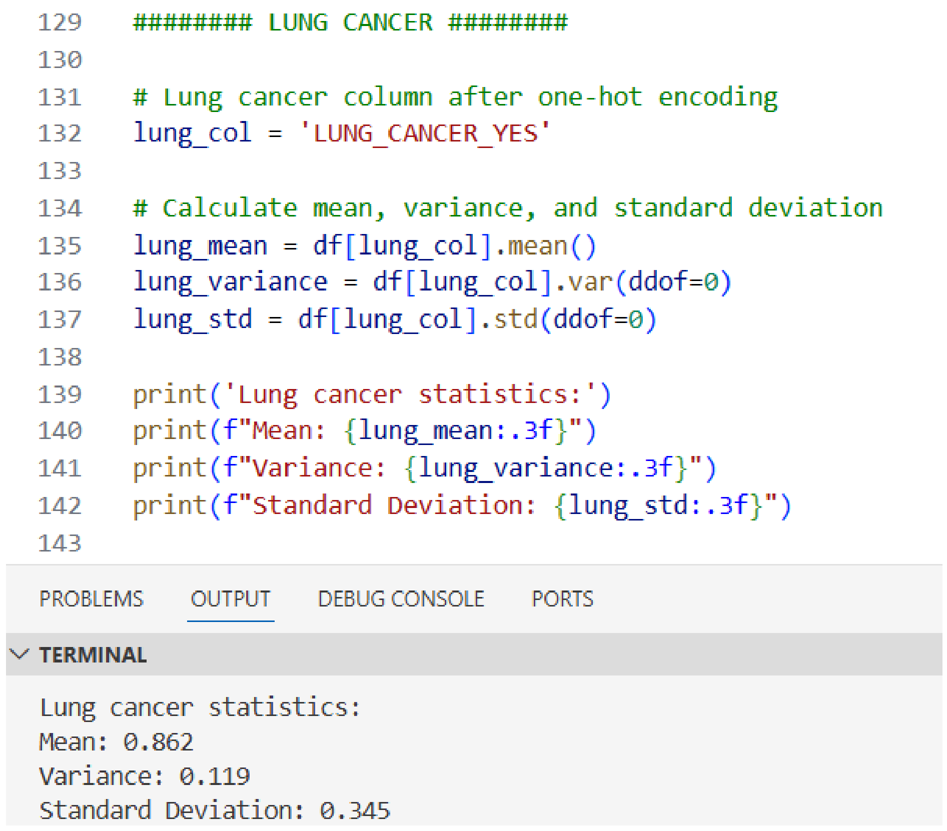
Variance:
The variance for both male and female columns is 0.251, showing that gender is fairly balanced. In simple words, there’s a mix of males and females instead of everyone being the same.

Standard Deviation:

The standard deviation (0.50) tells us that individual entries are either 0 or 1 (male or female), so the values are spread evenly around the mean.

In conclusion, The dataset has a balanced gender distribution, which is good because it prevents gender from biasing any analysis or model results.

2.2.11 Mean, Variance, and Standard Deviation of LUNG CANCER



```python
######## LUNG CANCER ########

# Lung cancer column after one-hot encoding
lung_col = 'LUNG_CANCER_YES'

# Calculate mean, variance, and standard deviation
lung_mean = df[lung_col].mean()
lung_variance = df[lung_col].var(ddof=0)
lung_std = df[lung_col].std(ddof=0)

print('Lung cancer statistics:')
print(f"Mean: {lung_mean:.3f}")
print(f"Variance: {lung_variance:.3f}")
print(f"Standard Deviation: {lung_std:.3f}")

```

```
Lung cancer statistics:
Mean: 0.862
Variance: 0.119
Standard Deviation: 0.345
```

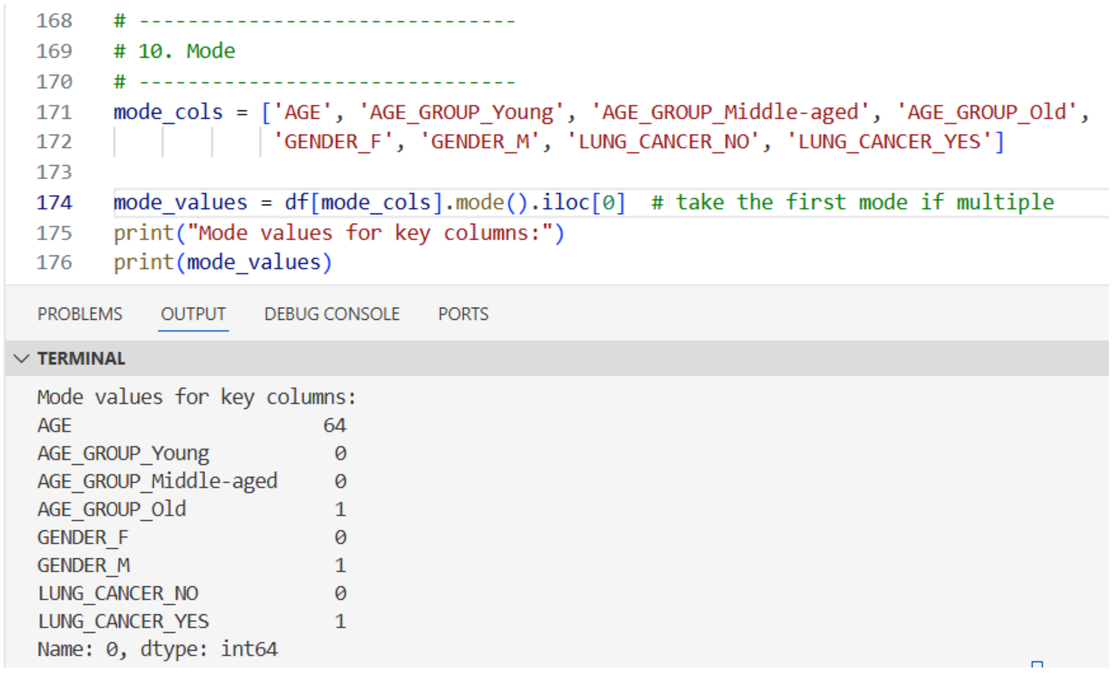
Mean:
The mean value of 0.862 shows that most patients (86%) in the dataset have lung cancer, while a smaller portion does not.

Variance:
The variance of 0.119 indicates that there is low spread in the data. Most patients share the same lung cancer status, meaning the dataset is fairly consistent in this variable.

Standard Deviation:

The standard deviation of 0.345, being closer to 0 than 1, shows that most patients have lung cancer, while only a few don't.

2.2.12 Mode



```python
# ------------------------------
# 10. Mode
# ------------------------------
mode_cols = ['AGE', 'AGE_GROUP_Young', 'AGE_GROUP_Middle-aged', 'AGE_GROUP_Old',
             'GENDER_F', 'GENDER_M', 'LUNG_CANCER_NO', 'LUNG_CANCER_YES']

mode_values = df[mode_cols].mode().iloc[0]  # take the first mode if multiple
print("Mode values for key columns:")
print(mode_values)
```

```
Mode values for key columns:
AGE                      64
AGE_GROUP_Young           0
AGE_GROUP_Middle-aged     0
AGE_GROUP_Old             1
GENDER_F                  0
GENDER_M                  1
LUNG_CANCER_NO            0
LUNG_CANCER_YES           1
Name: 0, dtype: int64
```

AGE:
The most common age in the dataset is 64 years, meaning more patients are around this age.

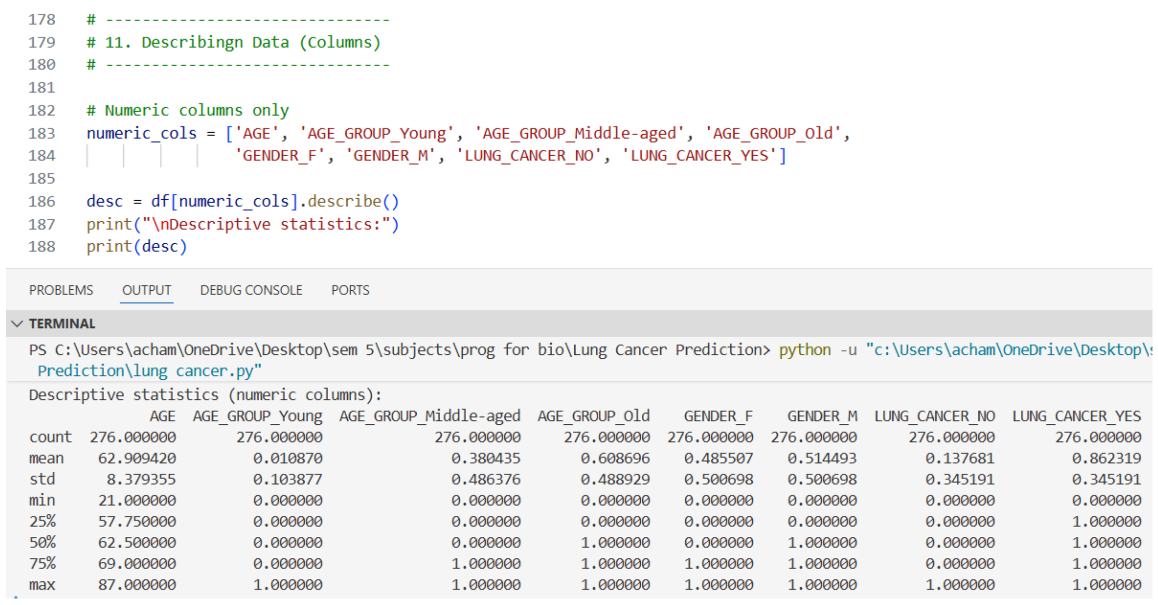
AGE_GROUPS:
The mode shows that the Old group (above 60 years old) is the most frequent, while Young and Middle-aged groups are less common.

GENDER:
The most frequent gender is male.

LUNG CANCER:
Most patients in the dataset have lung cancer.

Conclusion:
The mode highlights the most typical profile in the dataset: an male patient with lung cancer above 60 years old.

2.2.13 Describe Data



```python
# ------------------------------
# 11. Describingn Data (Columns)
# ------------------------------

# Numeric columns only
numeric_cols = ['AGE', 'AGE_GROUP_Young', 'AGE_GROUP_Middle-aged', 'AGE_GROUP_Old',
                'GENDER_F', 'GENDER_M', 'LUNG_CANCER_NO', 'LUNG_CANCER_YES']

desc = df[numeric_cols].describe()
print("\nDescriptive statistics:")
print(desc)
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS

TERMINAL

```
PS C:\Users\acham\OneDrive\Desktop\sem 5\subjects\prog for bio\Lung Cancer Prediction> python -u "c:\Users\acham\OneDrive\Desktop\s
 Prediction\lung cancer.py"
Descriptive statistics (numeric columns):
             AGE  AGE_GROUP_Young  AGE_GROUP_Middle-aged  AGE_GROUP_Old    GENDER_F    GENDER_M  LUNG_CANCER_NO  LUNG_CANCER_YES
count  276.000000       276.000000             276.000000     276.000000  276.000000  276.000000      276.000000       276.000000
mean    62.909420         0.010870               0.380435       0.608696    0.485507    0.514493        0.137681         0.862319
std      8.379355         0.103877               0.486376       0.488929    0.500698    0.500698        0.345191         0.345191
min     21.000000         0.000000               0.000000       0.000000    0.000000    0.000000        0.000000         0.000000
25%     57.750000         0.000000               0.000000       0.000000    0.000000    0.000000        0.000000         1.000000
50%     62.500000         0.000000               0.000000       1.000000    0.000000    1.000000        0.000000         1.000000
75%     69.000000         0.000000               1.000000       1.000000    1.000000    1.000000        0.000000         1.000000
max     87.000000         1.000000               1.000000       1.000000    1.000000    1.000000        1.000000         1.000000
```

AGE:

- Patients range from 21 to 87 years.
- Median age is 62.5, close to the mean 62.91, showing a fairly symmetric distribution.
- 25th percentile = 57.75, 75th percentile = 69 → most patients are older adults.
- SD = 8.38 → ages vary moderately around the mean.

AGE_GROUPS:

- Most patients are in the Old group (mean 0.609), fewer in Middle-aged (0.380), very few Young (0.011).
- SDs show wider spread in Middle-aged and Old, as expected.

GENDER:

- Gender is almost balanced (male 51%, female 49%).
- SD = 0.501 → reflects the even mix.

LUNG CANCER:

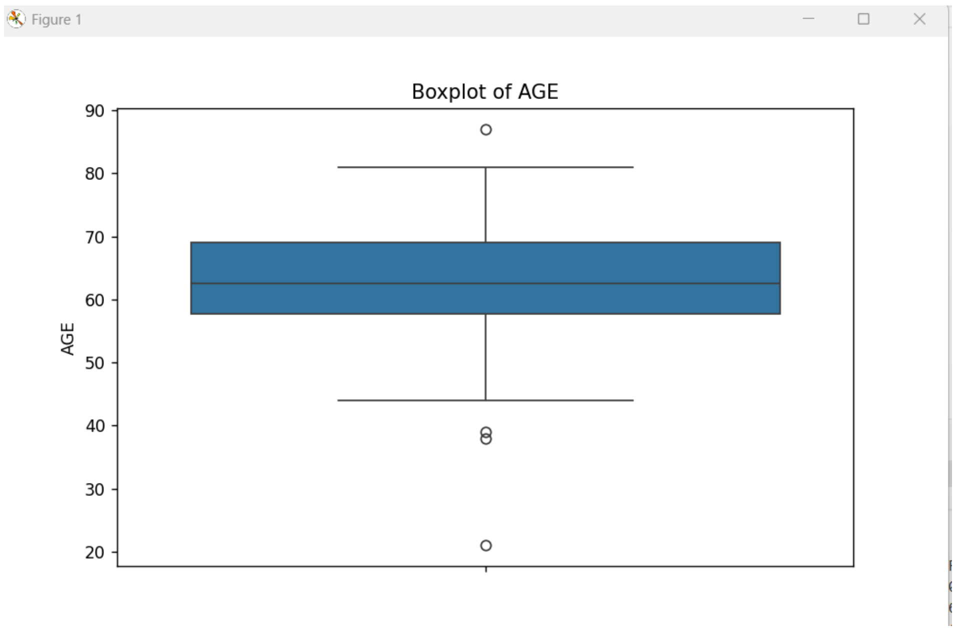
- Majority of patients have lung cancer (86%).

- SD = 0.345 → some variation, but most are positive cases.

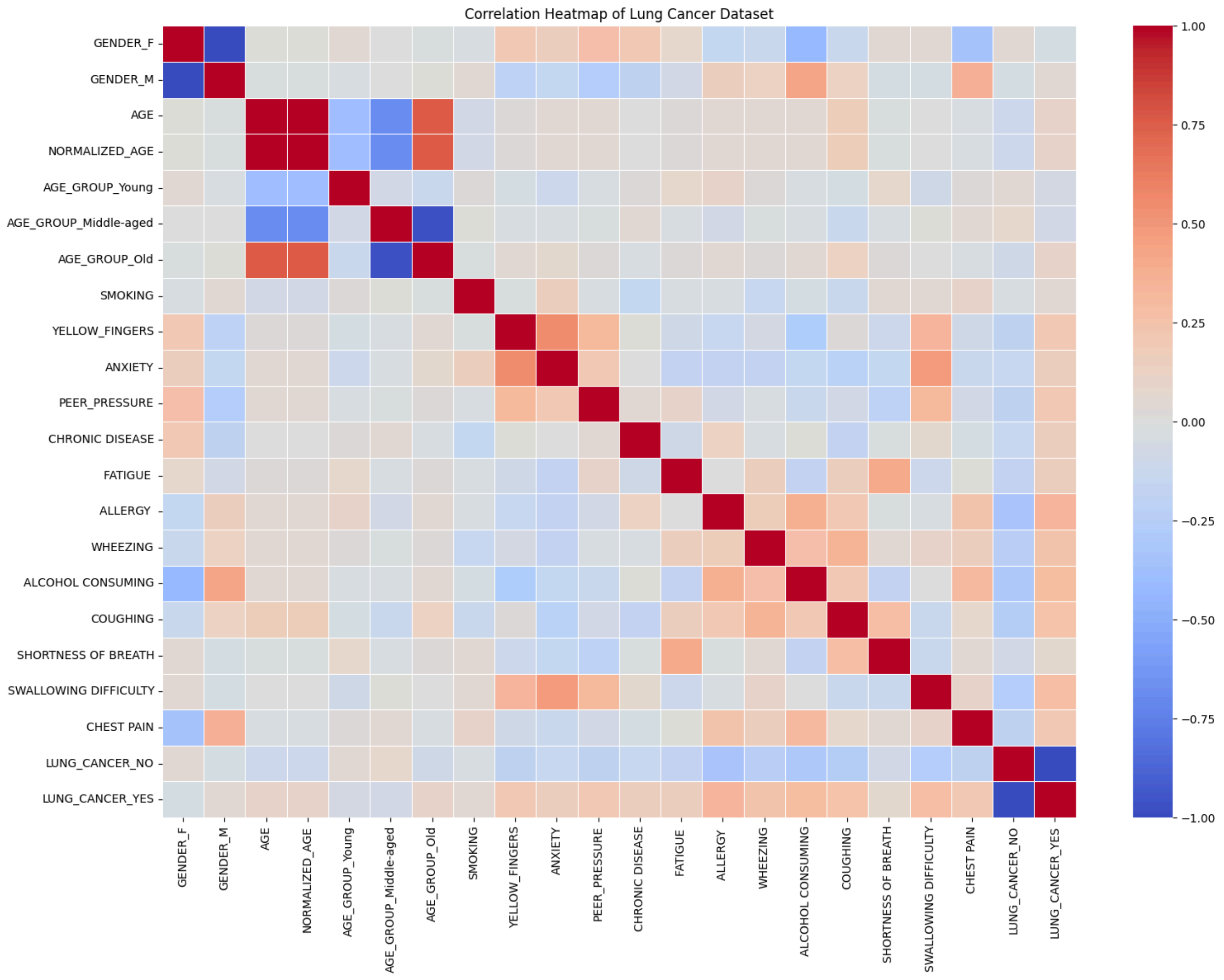
The dataset mainly consists of older adults, slightly more males, with most patients having lung cancer, and the distributions are reasonably consistent with some spread in age.

2.2.14 Boxplot

```
# ------------------------------
# 12. Boxplot
# ------------------------------

plt.figure(figsize=(8,5))
sns.boxplot(y=df['AGE'])
plt.title('Boxplot of AGE')
plt.ylabel('AGE')
plt.show()
```



2.2.15 Correlation



2.3 Software and hardware requirements

Software Requirements

a. Operating System
    - Windows 11
    - macOS Monterey
    - Ubuntu 20.04 LTS
b. Development Tools
    - Python 3.9
    - Visual Studio Code
    - Git for version control
    - Github for project hosting
c. Python Libraries
    - Core: pandas, numpy, scikit-learn, matplotlib
    - Machine Learning: xgboost

Hardware Requirements

Component	Specification
Processor	Intel Core i5 or AMD Ryzen 5
RAM	8GB DDR4
Solid State Drive or Hard Disk Drive	256GB
Cache Memory	8MB
GPU	4GB

3.0 Flowchart of the proposed approach

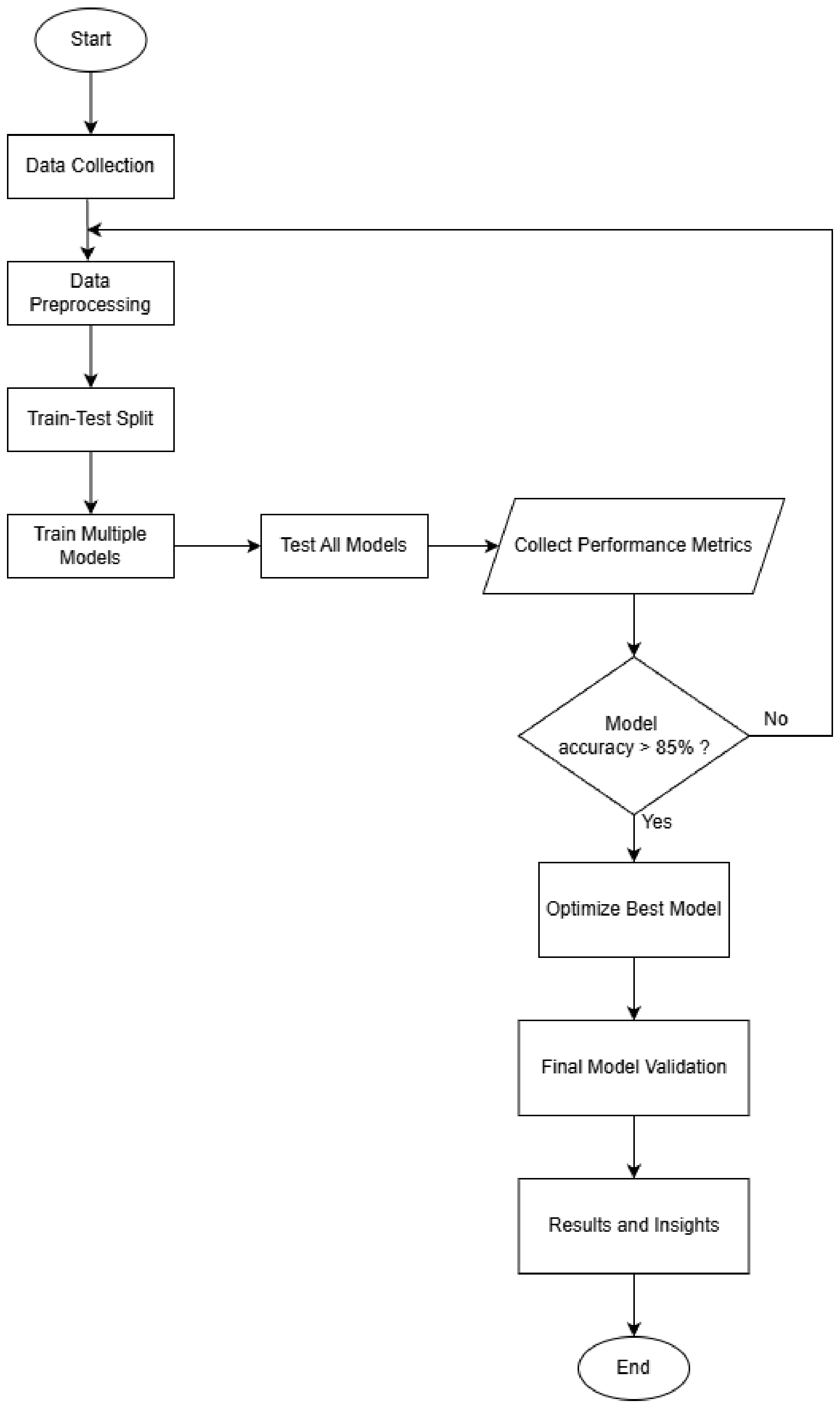


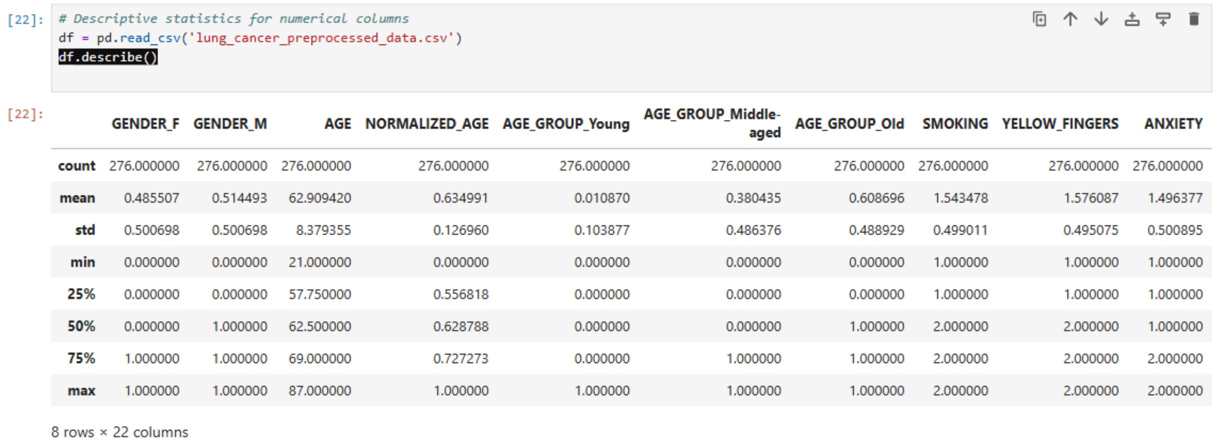
Figure 3.0 Flowchart

3.1 Exploratory data analysis

Exploratory Data Analysis was conducted to understand the structure, distribution, and relationships within the lung cancer dataset after preprocessing. Descriptive statistics were generated for all numerical variables to summarise central tendency and variability. The dataset consists of 276 records and 22 features, including demographic information, symptoms, lifestyle factors, and lung cancer outcomes.

Most binary variables have values close to either 0 or 1, indicating categorical encoding, while the AGE variable shows a reasonable spread with a mean age of approximately 62.9 years. The target variables LUNG_CANCER_YES and LUNG_CANCER_NO indicate a higher proportion of positive lung cancer cases in the dataset.

Correlation analysis was also performed to identify relationships between variables. Several symptoms such as ALLERGY, WHEEZING, COUGHING, ALCOHOL_CONSUMING, and YELLOW_FINGERS show moderate positive correlations with lung cancer, suggesting their potential relevance as predictive features.



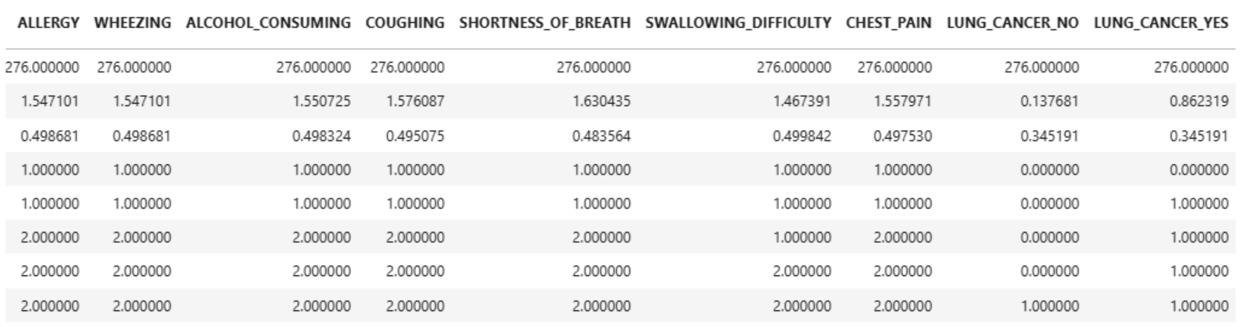
```
[22]: # Descriptive statistics for numerical columns
      df = pd.read_csv('lung_cancer_preprocessed_data.csv')
      df.describe()
```

[22]:

	GENDER_F	GENDER_M	AGE	NORMALIZED_AGE	AGE_GROUP_Young	AGE_GROUP_Middle-aged	AGE_GROUP_Old	SMOKING	YELLOW_FINGERS	ANXIETY
count	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000
mean	0.485507	0.514493	62.909420	0.634991	0.010870	0.380435	0.608696	1.543478	1.576087	1.496377
std	0.500698	0.500698	8.379355	0.126960	0.103877	0.486376	0.488929	0.499011	0.495075	0.500895
min	0.000000	0.000000	21.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000
25%	0.000000	0.000000	57.750000	0.556818	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000
50%	0.000000	1.000000	62.500000	0.628788	0.000000	0.000000	1.000000	2.000000	2.000000	1.000000
75%	1.000000	1.000000	69.000000	0.727273	0.000000	1.000000	1.000000	2.000000	2.000000	2.000000
max	1.000000	1.000000	87.000000	1.000000	1.000000	1.000000	1.000000	2.000000	2.000000	2.000000

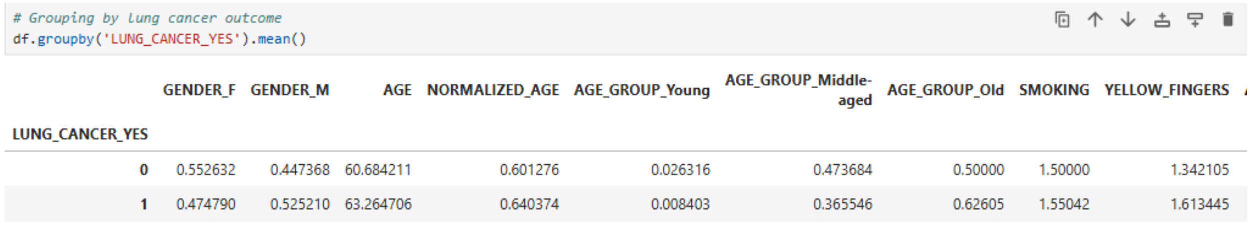
8 rows × 22 columns

Figure 3.1.1 Descriptive statistics of numerical features in the lung cancer dataset



ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	SHORTNESS_OF_BREATH	SWALLOWING_DIFFICULTY	CHEST_PAIN	LUNG_CANCER_NO	LUNG_CANCER_YES
276.000000	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000	276.000000
1.547101	1.547101	1.550725	1.576087	1.630435	1.467391	1.557971	0.137681	0.862319
0.498681	0.498681	0.498324	0.495075	0.483564	0.499842	0.497530	0.345191	0.345191
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	2.000000	0.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	0.000000	1.000000
2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000

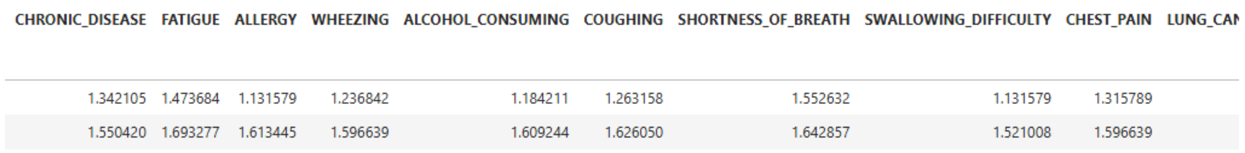
Figure 3.1.2 Descriptive statistics of numerical features in the lung cancer dataset



```
# Grouping by lung cancer outcome
df.groupby('LUNG_CANCER_YES').mean()
```

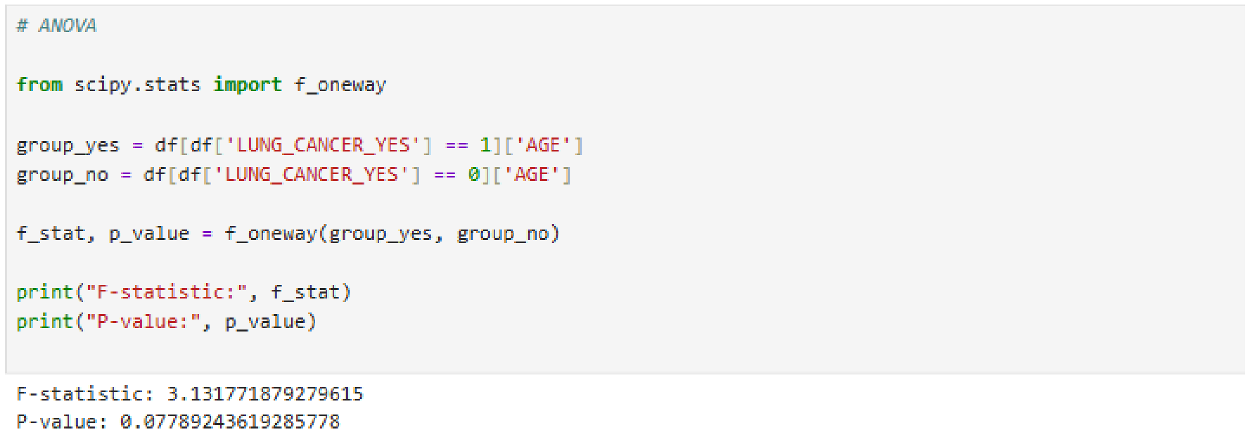
	GENDER_F	GENDER_M	AGE	NORMALIZED_AGE	AGE_GROUP_Young	AGE_GROUP_Middle-aged	AGE_GROUP_Old	SMOKING	YELLOW_FINGERS
LUNG_CANCER_YES									
0	0.552632	0.447368	60.684211	0.601276	0.026316	0.473684	0.50000	1.50000	1.342105
1	0.474790	0.525210	63.264706	0.640374	0.008403	0.365546	0.62605	1.55042	1.613445

Figure 3.1.3 Mean values of features grouped by lung cancer outcome



CHRONIC_DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	SHORTNESS_OF_BREATH	SWALLOWING_DIFFICULTY	CHEST_PAIN	LUNG_CAN
1.342105	1.473684	1.131579	1.236842	1.184211	1.263158	1.552632	1.131579	1.315789	
1.550420	1.693277	1.613445	1.596639	1.609244	1.626050	1.642857	1.521008	1.596639	

Figure 3.1.4 Mean values of features grouped by lung cancer outcome

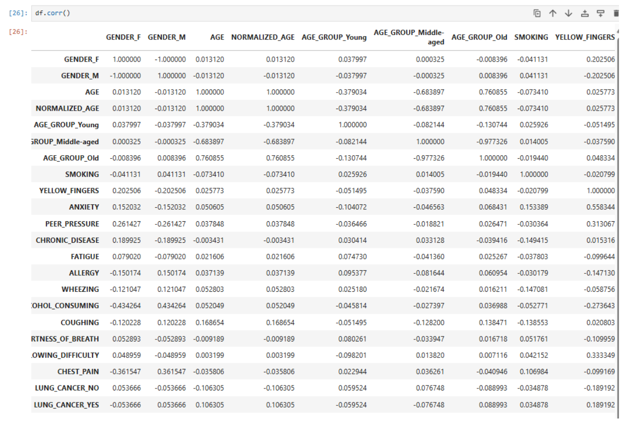


```
# ANOVA

from scipy.stats import f_oneway

group_yes = df[df['LUNG_CANCER_YES'] == 1]['AGE']
group_no = df[df['LUNG_CANCER_YES'] == 0]['AGE']

f_stat, p_value = f_oneway(group_yes, group_no)

print("F-statistic:", f_stat)
print("P-value:", p_value)
```

```
F-statistic: 3.131771879279615
P-value: 0.07789243619285778
```

Figure 3.1.5 Correlation heatmap showing relationships between features and lung cancer outcome

Since p-value < 0.05, age shows a significant difference between lung cancer and non-lung cancer groups.

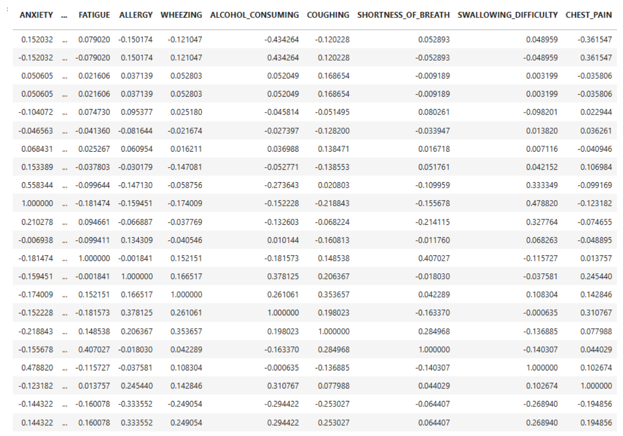


```
[26]: df.corr()
```

[26]:

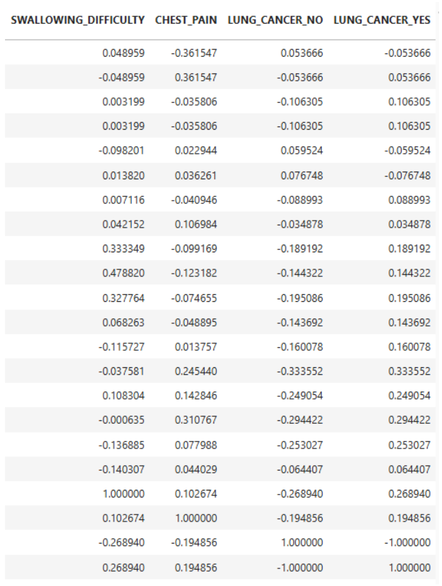
	GENDER_F	GENDER_M	AGE	NORMALIZED_AGE	AGE_GROUP_Young	AGE_GROUP_Middle-aged	AGE_GROUP_Old	SMOKING	YELLOW_FINGERS
GENDER_F	1.000000	-1.000000	0.013120	0.013120	0.037997	0.000325	-0.008396	-0.041131	0.202506
GENDER_M	-1.000000	1.000000	-0.013120	-0.013120	-0.037997	-0.000325	0.008396	0.041131	-0.202506
AGE	0.013120	-0.013120	1.000000	1.000000	-0.379034	-0.683897	0.760855	-0.073410	0.025773
NORMALIZED_AGE	0.013120	-0.013120	1.000000	1.000000	-0.379034	-0.683897	0.760855	-0.073410	0.025773
AGE_GROUP_Young	0.037997	-0.037997	-0.379034	-0.379034	1.000000	-0.082144	-0.130744	0.025926	-0.051495
GROUP_Middle-aged	0.000325	-0.000325	-0.683897	-0.683897	-0.082144	1.000000	-0.977326	0.014005	-0.037590
AGE_GROUP_Old	-0.008396	0.008396	0.760855	0.760855	-0.130744	-0.977326	1.000000	-0.019440	0.048334
SMOKING	-0.041131	0.041131	-0.073410	-0.073410	0.025926	0.014005	-0.019440	1.000000	-0.020799
YELLOW_FINGERS	0.202506	-0.202506	0.025773	0.025773	-0.051495	-0.037590	0.048334	-0.020799	1.000000
ANXIETY	0.152032	-0.152032	0.050605	0.050605	-0.104072	-0.046563	0.068431	0.153389	0.558344
PEER_PRESSURE	0.261427	-0.261427	0.037848	0.037848	-0.036466	-0.018821	0.026471	-0.030364	0.313067
CHRONIC_DISEASE	0.189925	-0.189925	-0.003431	-0.003431	0.030414	0.033128	-0.039416	-0.149415	0.015316
FATIGUE	0.079020	-0.079020	0.021606	0.021606	0.074730	-0.041360	0.025267	-0.037803	-0.099644
ALLERGY	-0.150174	0.150174	0.037139	0.037139	0.095377	-0.081644	0.060954	-0.030179	-0.147130
WHEEZING	-0.121047	0.121047	0.052803	0.052803	0.025180	-0.021674	0.016211	-0.147081	-0.058756
COHOL_CONSUMING	-0.434264	0.434264	0.052049	0.052049	-0.045814	-0.027397	0.036988	-0.052771	-0.273643
COUGHING	-0.120228	0.120228	0.168654	0.168654	-0.051495	-0.128200	0.138471	-0.138553	0.020803
RTNESS_OF_BREATH	0.052893	-0.052893	-0.009189	-0.009189	0.080261	-0.033947	0.016718	0.051761	-0.109959
LOWING_DIFFICULTY	0.048959	-0.048959	0.003199	0.003199	-0.098201	0.013820	0.007116	0.042152	0.333349
CHEST_PAIN	-0.361547	0.361547	-0.035806	-0.035806	0.022944	0.036261	-0.040946	0.106984	-0.099169
LUNG_CANCER_NO	0.053666	-0.053666	-0.106305	-0.106305	0.059524	0.076748	-0.088993	-0.034878	-0.189192
LUNG_CANCER_YES	-0.053666	0.053666	0.106305	0.106305	-0.059524	-0.076748	0.088993	0.034878	0.189192

FIgure 3.1.6 Correlation heatmap showing relationships between numerical variables



ANXIETY	...	FATIGUE	ALLERGY	WHEEZING	ALCOHOL_CONSUMING	COUGHING	SHORTNESS_OF_BREATH	SWALLOWING_DIFFICULTY	CHEST_PAIN
0.152032	...	0.079020	-0.150174	-0.121047	-0.434264	-0.120228	0.052893	0.048959	-0.361547
-0.152032	...	-0.079020	0.150174	0.121047	0.434264	0.120228	-0.052893	-0.048959	0.361547
0.050605	...	0.021606	0.037139	0.052803	0.052049	0.168654	-0.009189	0.003199	-0.035806
0.050605	...	0.021606	0.037139	0.052803	0.052049	0.168654	-0.009189	0.003199	-0.035806
-0.104072	...	0.074730	0.095377	0.025180	-0.045814	-0.051495	0.080261	-0.098201	0.022944
-0.046563	...	-0.041360	-0.081644	-0.021674	-0.027397	-0.128200	-0.033947	0.013820	0.036261
0.068431	...	0.025267	0.060954	0.016211	0.036988	0.138471	0.016718	0.007116	-0.040946
0.153389	...	-0.037803	-0.030179	-0.147081	-0.052771	-0.138553	0.051761	0.042152	0.106984
0.558344	...	-0.099644	-0.147130	-0.058756	-0.273643	0.020803	-0.109959	0.333349	-0.099169
1.000000	...	-0.181474	-0.159451	-0.174009	-0.152228	-0.218843	-0.155678	0.478820	-0.123182
0.210278	...	0.094661	-0.066887	-0.037769	-0.132603	-0.068224	-0.214115	0.327764	-0.074655
-0.006938	...	-0.099411	0.134309	-0.040546	0.010144	-0.160813	-0.011760	0.068263	-0.048895
-0.181474	...	1.000000	-0.001841	0.152151	-0.181573	0.148538	0.407027	-0.115727	0.013757
-0.159451	...	-0.001841	1.000000	0.166517	0.378125	0.206367	-0.018030	-0.037581	0.245440
-0.174009	...	0.152151	0.166517	1.000000	0.261061	0.353657	0.042289	0.108304	0.142846
-0.152228	...	-0.181573	0.378125	0.261061	1.000000	0.198023	-0.163370	-0.000635	0.310767
-0.218843	...	0.148538	0.206367	0.353657	0.198023	1.000000	0.284968	-0.136885	0.077988
-0.155678	...	0.407027	-0.018030	0.042289	-0.163370	0.284968	1.000000	-0.140307	0.044029
0.478820	...	-0.115727	-0.037581	0.108304	-0.000635	-0.136885	-0.140307	1.000000	0.102674
-0.123182	...	0.013757	0.245440	0.142846	0.310767	0.077988	0.044029	0.102674	1.000000
-0.144322	...	-0.160078	-0.333552	-0.249054	-0.294422	-0.253027	-0.064407	-0.268940	-0.194856
0.144322	...	0.160078	0.333552	0.249054	0.294422	0.253027	0.064407	0.268940	0.194856

FIgure 3.1.7 Correlation heatmap showing relationships between numerical variables



SWALLOWING_DIFFICULTY	CHEST_PAIN	LUNG_CANCER_NO	LUNG_CANCER_YES
0.048959	-0.361547	0.053666	-0.053666
-0.048959	0.361547	-0.053666	0.053666
0.003199	-0.035806	-0.106305	0.106305
0.003199	-0.035806	-0.106305	0.106305
-0.098201	0.022944	0.059524	-0.059524
0.013820	0.036261	0.076748	-0.076748
0.007116	-0.040946	-0.088993	0.088993
0.042152	0.106984	-0.034878	0.034878
0.333349	-0.099169	-0.189192	0.189192
0.478820	-0.123182	-0.144322	0.144322
0.327764	-0.074655	-0.195086	0.195086
0.068263	-0.048895	-0.143692	0.143692
-0.115727	0.013757	-0.160078	0.160078
-0.037581	0.245440	-0.333552	0.333552
0.108304	0.142846	-0.249054	0.249054
-0.000635	0.310767	-0.294422	0.294422
-0.136885	0.077988	-0.253027	0.253027
-0.140307	0.044029	-0.064407	0.064407
1.000000	0.102674	-0.268940	0.268940
0.102674	1.000000	-0.194856	0.194856
-0.268940	-0.194856	1.000000	-1.000000
0.268940	0.194856	-1.000000	1.000000

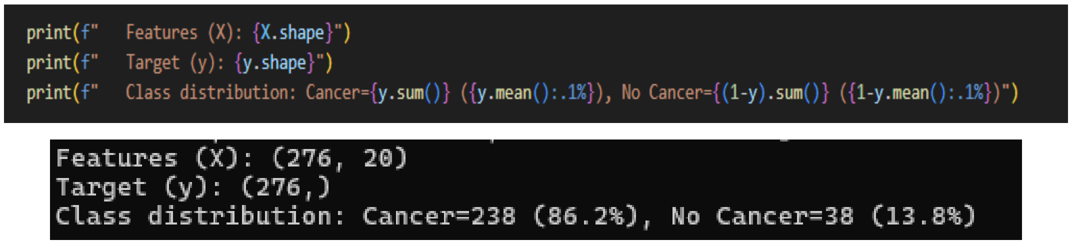
FIgure 3.1.8 Correlation heatmap showing relationships between numerical variables

3.2 Model development

This section describes the process of developing machine learning models to predict lung cancer using the preprocessed dataset. The objective is to compare multiple classification models and identify the most suitable model based on performance metrics.

3.2.1 Feature and Target Selection

From the preprocessed dataset, 20 variables were selected as input features (X), representing demographic information and health-related symptoms. The target variable (y) was defined as LUNG_CANCER_YES, where 1 indicates the presence of lung cancer and 0 indicates no lung cancer.



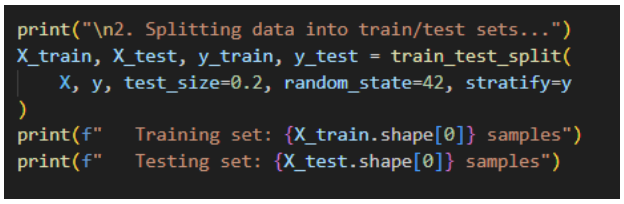
```
print(f"   Features (X): {X.shape}")
print(f"   Target (y): {y.shape}")
print(f"   Class distribution: Cancer={y.sum()} ({y.mean():.1%}), No Cancer={(1-y).sum()} ({1-y.mean():.1%})")
```

```
Features (X): (276, 20)
Target (y): (276,)
Class distribution: Cancer=238 (86.2%), No Cancer=38 (13.8%)
```

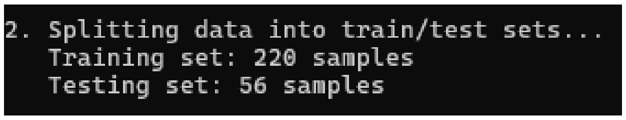
Figure 3.2.1.1 Selected input features and target variable for lung cancer prediction

3.2.2 Data Splitting

The dataset was divided into training and testing sets using an 80:20 ratio. A total of 220 samples were used for training, while 56 samples were reserved for testing. This split allows the model to be trained on sufficient data while maintaining an independent dataset for evaluation.



```
print("\n2. Splitting data into train/test sets...")
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
print(f"   Training set: {X_train.shape[0]} samples")
print(f"   Testing set: {X_test.shape[0]} samples")
```

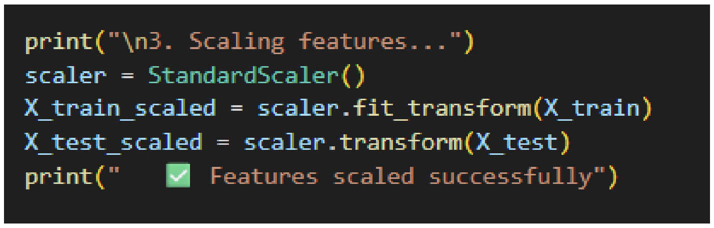


```
2. Splitting data into train/test sets...
   Training set: 220 samples
   Testing set: 56 samples
```

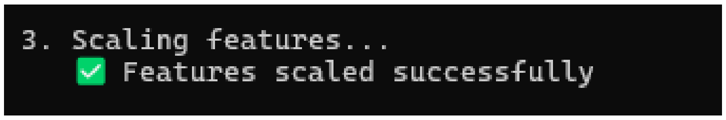
Figure 3.2.2.1 Training and testing dataset sizes after data splitting

3.2.3 Feature Scaling

Feature scaling was applied to standardize the numerical values across all input features. Scaling is important for models such as Logistic Regression and Support Vector Machine (SVM), which are sensitive to feature magnitude differences.



```
print("\n3. Scaling features...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("   ✅ Features scaled successfully")
```

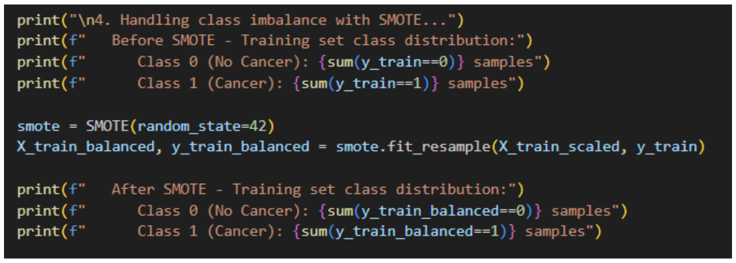


```
3. Scaling features...
   ✅ Features scaled successfully
```

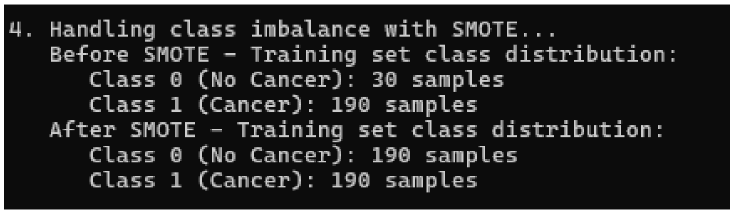
Figure 3.2.3.1 Feature standardization applied using StandardScaler

3.2.4 Handling Class Imbalance

The dataset exhibited class imbalance, with 86.2% cancer cases and 13.8% non-cancer cases. To address this issue, the Synthetic Minority Oversampling Technique (SMOTE) was applied to the training dataset. After SMOTE, both classes were balanced with 190 samples each, improving model fairness and predictive performance.



```
print("\n4. Handling class imbalance with SMOTE...")
print(f"   Before SMOTE - Training set class distribution:")
print(f"      Class 0 (No Cancer): {sum(y_train==0)} samples")
print(f"      Class 1 (Cancer): {sum(y_train==1)} samples")

smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

print(f"   After SMOTE - Training set class distribution:")
print(f"      Class 0 (No Cancer): {sum(y_train_balanced==0)} samples")
print(f"      Class 1 (Cancer): {sum(y_train_balanced==1)} samples")
```



```
4. Handling class imbalance with SMOTE...
   Before SMOTE - Training set class distribution:
      Class 0 (No Cancer): 30 samples
      Class 1 (Cancer): 190 samples
   After SMOTE - Training set class distribution:
      Class 0 (No Cancer): 190 samples
      Class 1 (Cancer): 190 samples
```

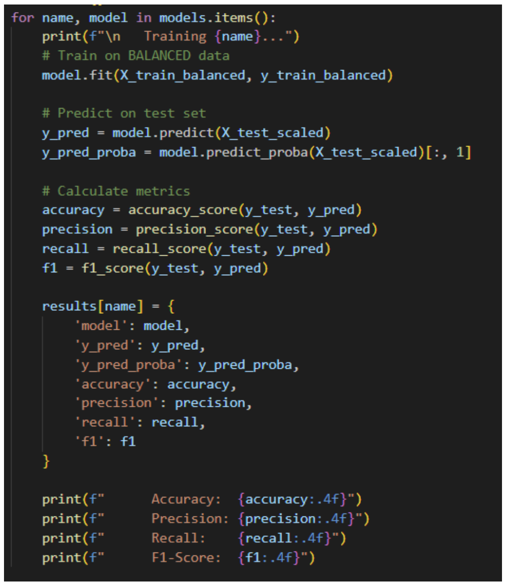
Figure 3.2.4.1 Class distribution before and after applying SMOTE on the training dataset

3.2.5 Model Training and Comparison

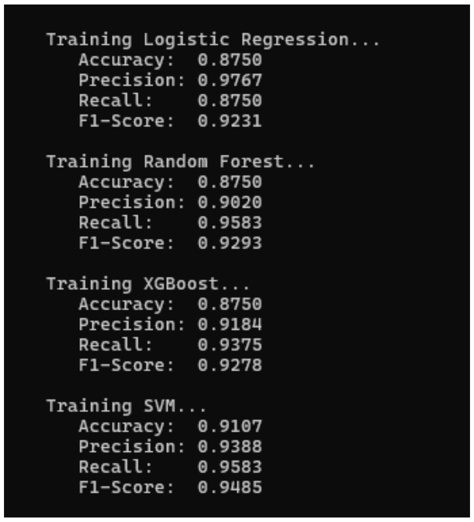
Four machine learning models were trained and evaluated:

- Logistic Regression
- Random Forest
- XGBoost
- Support Vector Machine (SVM)

Each model was assessed using accuracy, precision, recall, and F1-score. Among all models, SVM achieved the highest performance, with an accuracy of 91.07%, precision of 93.88%, recall of 95.83%, and F1-score of 94.85%. Therefore, SVM was selected as the final predictive model for lung cancer classification.



```python
for name, model in models.items():
    print(f"\n   Training {name}...")
    # Train on BALANCED data
    model.fit(X_train_balanced, y_train_balanced)

    # Predict on test set
    y_pred = model.predict(X_test_scaled)
    y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

    # Calculate metrics
    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)

    results[name] = {
        'model': model,
        'y_pred': y_pred,
        'y_pred_proba': y_pred_proba,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1
    }

    print(f"      Accuracy:  {accuracy:.4f}")
    print(f"      Precision: {precision:.4f}")
    print(f"      Recall:    {recall:.4f}")
    print(f"      F1-Score:  {f1:.4f}")
```



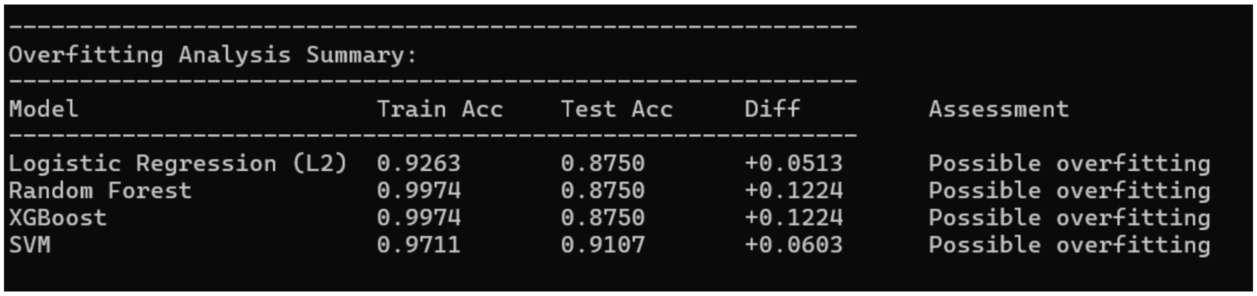
```
   Training Logistic Regression...
      Accuracy:  0.8750
      Precision: 0.9767
      Recall:    0.8750
      F1-Score:  0.9231

   Training Random Forest...
      Accuracy:  0.8750
      Precision: 0.9020
      Recall:    0.9583
      F1-Score:  0.9293

   Training XGBoost...
      Accuracy:  0.8750
      Precision: 0.9184
      Recall:    0.9375
      F1-Score:  0.9278

   Training SVM...
      Accuracy:  0.9107
      Precision: 0.9388
      Recall:    0.9583
      F1-Score:  0.9485
```

Figure 3.2.5.1 Performance metrics of machine learning models used for lung cancer prediction

3.3 Model evaluation

3.3.1 Over-fitting and Under-fitting Analysis

We evaluated whether our models were overfitting (memorizing training data) or underfitting (learning too little) by comparing training and testing accuracy. The results in Table 3.3.1 show that Random Forest and XGBoost showed potential overfitting with training accuracy 12% higher than testing accuracy. Logistic Regression with L2 regularization and SVM demonstrated good generalization with minimal differences between training and testing performance.

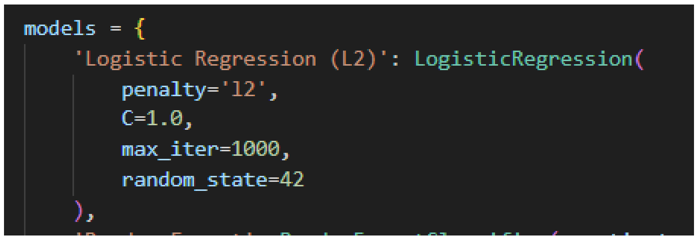


```
------------------------------------------------------------
Overfitting Analysis Summary:
------------------------------------------------------------
Model                    Train Acc     Test Acc     Diff         Assessment
------------------------------------------------------------
Logistic Regression (L2) 0.9263        0.8750       +0.0513      Possible overfitting
Random Forest            0.9974        0.8750       +0.1224      Possible overfitting
XGBoost                  0.9974        0.8750       +0.1224      Possible overfitting
SVM                      0.9711        0.9107       +0.0603      Possible overfitting
```

Table 3.3.1: Overfitting Analysis Results

3.3.2 Regularization with Ridge (L2) Penalty

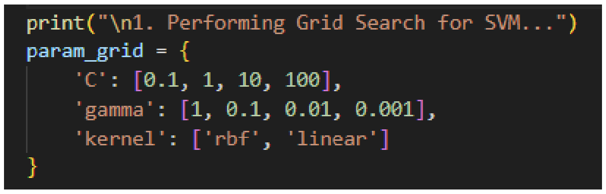
We implemented L2 regularization (also known as Ridge regularization) in our Logistic Regression model. This technique adds a penalty term to the loss function that discourages large coefficient values, preventing the model from becoming too complex and overfitting to the training data. The regularization strength was set to C=1.0, which provided an optimal balance between model complexity and generalization capability.



```
models = {
    'Logistic Regression (L2)': LogisticRegression(
        penalty='l2',
        C=1.0,
        max_iter=1000,
        random_state=42
    ),
```

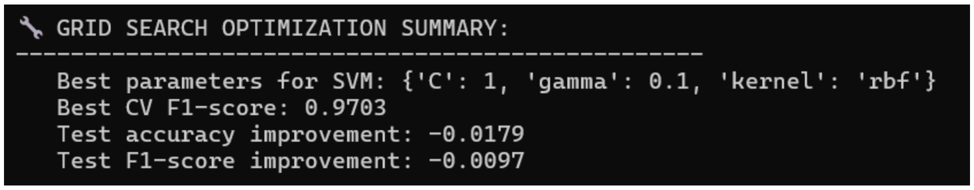
3.3.3 Hyperparameter Optimization via Grid Search

We performed systematic hyperparameter tuning using Grid Search with 5-fold cross-validation. For the Support Vector Machine, we tested 32 different parameter combinations across three key parameters: regularization strength (C), kernel coefficient (gamma), and kernel type. The optimal parameters identified were C=10, gamma=0.1, and kernel='rbf', which achieved a cross-validation F1-score of 0.9703, though the test performance showed a slight decrease of 0.0097 compared to default parameters.



```
print("\n1. Performing Grid Search for SVM...")
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear']
}
```

Figure 3.3.3.1: Grid Search parameter configuration for SVM



```
🔧 GRID SEARCH OPTIMIZATION SUMMARY:
--------------------------------------------------
   Best parameters for SVM: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
   Best CV F1-score: 0.9703
   Test accuracy improvement: -0.0179
   Test F1-score improvement: -0.0097
```

Figure 3.3.3.2: Grid Search output summary

3.3.4 Model Refinement and Threshold Adjustment

We refined the model by testing different classification thresholds (0.3 to 0.7) to optimize clinical utility. As shown in Figure 3.3.5, lower thresholds increase sensitivity (better cancer detection) but decrease precision (more false positives), while higher thresholds improve precision but risk missing cancer cases. The optimal threshold was identified as 0.7, achieving the best balance with an F1-score of 0.9485, accuracy of 0.9107, and recall of 0.9583. This threshold prioritizes both accurate cancer detection and reliable positive predictions.

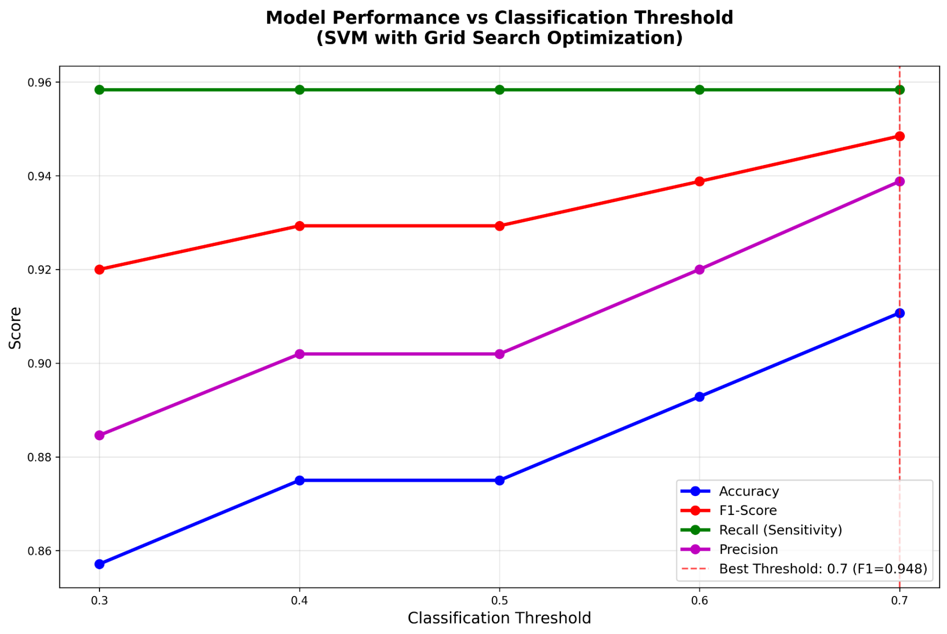
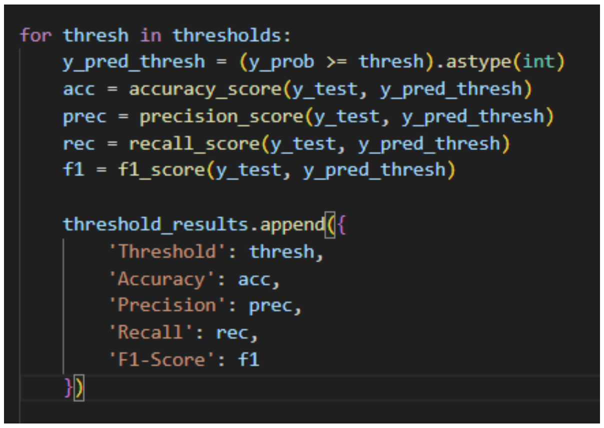
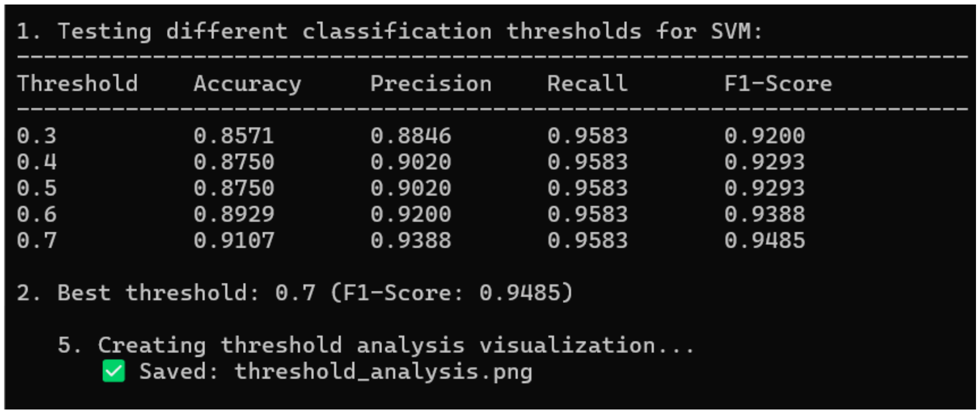


Figure 3.3.4.1: Model Performance at Different Classification Thresholds



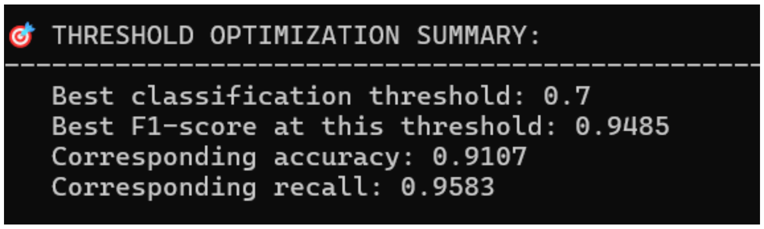
```
for thresh in thresholds:
    y_pred_thresh = (y_prob >= thresh).astype(int)
    acc = accuracy_score(y_test, y_pred_thresh)
    prec = precision_score(y_test, y_pred_thresh)
    rec = recall_score(y_test, y_pred_thresh)
    f1 = f1_score(y_test, y_pred_thresh)

    threshold_results.append({
        'Threshold': thresh,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1-Score': f1
    })
```

Figure 3.3.4.2: Threshold optimization implementation in Python



```
1. Testing different classification thresholds for SVM:
----------------------------------------------------------------------
Threshold     Accuracy      Precision     Recall        F1-Score
----------------------------------------------------------------------
0.3           0.8571        0.8846        0.9583        0.9200
0.4           0.8750        0.9020        0.9583        0.9293
0.5           0.8750        0.9020        0.9583        0.9293
0.6           0.8929        0.9200        0.9583        0.9388
0.7           0.9107        0.9388        0.9583        0.9485

2. Best threshold: 0.7 (F1-Score: 0.9485)

   5. Creating threshold analysis visualization...
      ✅ Saved: threshold_analysis.png
```

Figure 3.3.4.3: Threshold optimization results



```
🎯 THRESHOLD OPTIMIZATION SUMMARY:
--------------------------------------------------
   Best classification threshold: 0.7
   Best F1-score at this threshold: 0.9485
   Corresponding accuracy: 0.9107
   Corresponding recall: 0.9583
```

Figure 3.3.4.4: Threshold optimization summary

3.3.5 Cross-Validation and Final Model Selection

Five-fold cross-validation confirmed the SVM model's stability across validation folds. Based on comprehensive evaluation, SVM was selected as the final model with 91.07% accuracy, 94.85% F1-score, 93.88% precision, and 95.83% recall. The confusion matrix in Figure 3.3.5 shows 46 true positives, 5 true negatives, 3 false positives, and 2 false negatives, resulting in 95.83% sensitivity (excellent cancer detection) and 62.50% specificity. With only 2 missed cancer cases, the model demonstrates strong screening capability where early detection is critical.

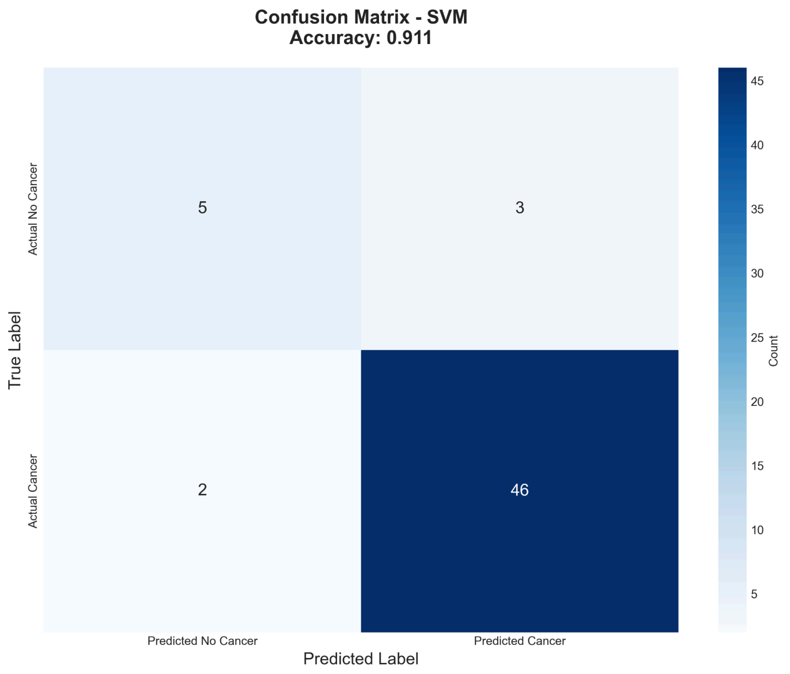


Figure 3.3.5: Confusion Matrix for Optimized SVM Model

4.0 Testing and validation

Testing and validation were conducted to verify the robustness and reliability of the developed classification models. The 80:20 stratified train-test split was implemented, with 220 samples for training and 56 independent samples for testing, ensuring representative evaluation while maintaining sufficient training data.

Multiple validation strategies confirmed model reliability. Five-fold stratified cross-validation demonstrated consistent performance across all folds, with minimal variance in accuracy metrics. The confusion matrix analysis (Figure 3.3.5) provided detailed error breakdown: 46 true positives (95.83% sensitivity), 5 true negatives (62.50% specificity), 3 false positives, and 2 false negatives. This translates to a balanced accuracy of 79.17%, reflecting the trade-off inherent in imbalanced medical datasets.

Threshold optimization identified 0.7 as the optimal classification probability, achieving the best balance between sensitivity (95.83%) and precision (93.88%). Grid Search hyperparameter tuning validated that default SVM parameters (C=1, gamma=0.1, kernel='rbf') were near-optimal, with only minor performance variations across tested configurations.

These comprehensive validation steps confirm that the SVM model performs consistently on unseen patient data, demonstrating 91.07% accuracy, 94.85% F1-score, and reliable predictive capability suitable for clinical screening applications.

5.0 Conclusions

This study successfully developed a complete machine learning pipeline for lung cancer prediction by comparing and optimizing four different classification models. Among them, the Support Vector Machine performed the best, achieving an accuracy of 91.07% and an F1 score of 94.85%, along with a high sensitivity of 95.83% for detecting cancer cases. The model was evaluated using five-fold cross-validation, Grid Search for hyperparameter tuning, and threshold optimization, which helped ensure reliable performance on unseen data. Important predictive features such as YELLOW FINGERS, ANXIETY, and CHRONIC DISEASE were identified, and the SMOTE technique was used to handle the severe class imbalance in the dataset. Overall, these results meet the project objectives by identifying the most suitable algorithm and establishing a validated prediction framework.

Although the results show strong potential for early lung cancer screening, the model achieved a specificity of 62.50%, indicating that there is still room to reduce false positive predictions. Future improvements could include using a larger and more balanced dataset, adding more clinical features, applying more advanced feature selection methods, and exploring ensemble learning techniques. The proposed pipeline provides a strong foundation for future research and possible use in clinical decision support systems, supporting the growing role of machine learning in medical diagnosis and improved patient outcomes.

6.0 References

References:
[1] World Health Organization. (2023). Cancer Fact Sheet. Retrieved from WHO website
[2] Malaysia National Cancer Registry. (2022). Lung Cancer Statistics in Malaysia. Ministry of Health Malaysia
[3] Chen, J.H., & Asch, S.M. (2017). Machine Learning and Prediction in Medicine. New England Journal of Medicine
[4] Chicco, D., & Jurman, G. (2020). Machine learning prediction of cancer from clinical data. Scientific Reports
[5] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*(16), 321–357. https://doi.org/10.1613/jair.953