Southampton Solent University

Department of Science and Engineering

Lung Cancer Data Analysis and Visualization

Author : Q102612026 Parteek Sharma

Course Title : **MSc Applied AI and Data Science**

Module Leader : **Jarutas Andritsch**

Date : **6 January 2025**

Table of Contents

Table of Figures

Index of Tables

1. Overview

The aim of this project is to analyze lung cancer data to identify key patterns and insights that could assist in early diagnosis, treatment planning, and patient management. The project focuses on using Python's Pandas and Matplotlib libraries for data analysis and visualization to derive meaningful trends from the dataset.

The dataset used in this project comprises a diverse set of features including patient demographics, clinical measurements, tumor characteristics, and comorbidities. These include both categorical features (e.g., gender, smoking history, and cancer stage) and numerical measurements (e.g., tumor size, blood pressure, and various blood test results). Most features are well-structured and appropriately typed, with integers representing counts or scores, floating-point values for continuous measurements, and strings for categorical labels. There is no missing data in the dataset used.

Table 1: Requirement Completion

Requirement	**Status**
A-Loading data using python's CSV module.	Completed
A1-Retrieve demographic information: age, gender, smoking history, and ethnicity based on the patient ID.	Completed
A2-Retrieve medical history details including family history of lung cancer, comorbidities of diabetes disease, comorbidities of kidney disease, and the haemoglobin level associated with a certain ethnicity.	Completed
A3-Retrieve treatment details including age, tumor size, tumor location, and tumor stage of patients who have survived more than 100 months on a certain treatment.	Completed
A4-Retrieve information from your chosen columns and apply a specific condition that relates to patient. Please select at least three columns and one condition that differs from previous requirements.	Completed
B-The system loads data from a CSV file into memory using the read_csv() function, utilising the file path or filename obtained from task A without prompting the user again.	Completed
B1-Identify the top 3 treatments for a certain ethnicity where patients have survived more than 100 months.	Completed
B2-Analyse the average white blood cell counts for certain treatments based on a certain ethnicity.	Completed
B3-Analyse the average number of smoking packs for patients in each treatment group, with a blood pressure (pulse) over 90 and a tumor size smaller than 15.0 mm, based by tumor location.	Completed
B4-Analyse the data to derive meaningful insights based on your unique selection, distinct from the previous requirements	Completed

C-The system enables users to visualise data using the matplotlib module. Using the dataframe from task B	Completed
C1-Create a chart to illustrate the proportion of cancer treatments among a certain ethnicity as specified by the user.	Completed
C2-Create a chart to show the trend of average smoking packs consumption across different cancer stages for each ethnicity within a single chart.	Completed
C3-Create a chart that visually compares the average of all blood pressure types across different treatment types within a single chart.	Completed
C4-Create a visualisation of your selection to showcase information related to patients, treatment, or conditions that can reveal trends, behaviours, or patterns, ensuring it is distinct from previous requirements.	Completed

Status options: Completed/ Partially Completed/ Not Attempted

2. Project Implementation

The project was implemented in Python, using libraries like Pandas for data manipulation and analysis and Matplotlib for data visualisation. NumPy was also used to generate some arrays and csv module was used to load the csv file.

2.1 Project structure

The software is organized into multiple modules, each focused on a specific task, making it easy to maintain. Here's a simplified outline of the structure:

Main File:

- Acts as the entry point and imports the menu module to start the program.
- Doesn't handle tasks directly; relies on the menu to guide the user.

Menu File:

- Provides a user interface with a main menu and submenus for Task A, Task B, and Task C.
- Imports task modules and directs the user to the chosen task.
- Allows navigation between tasks and the option to return to the main menu or exit.

Task Modules (A, B, and C):

- **Task A**: Handles file loading and basic analysis with csv.
- **Task B**: Manages data analysis using pandas.
- **Task C**: Handles data visualizations with matplotlib.
- These modules are imported into the menu and executed based on the user's choice.

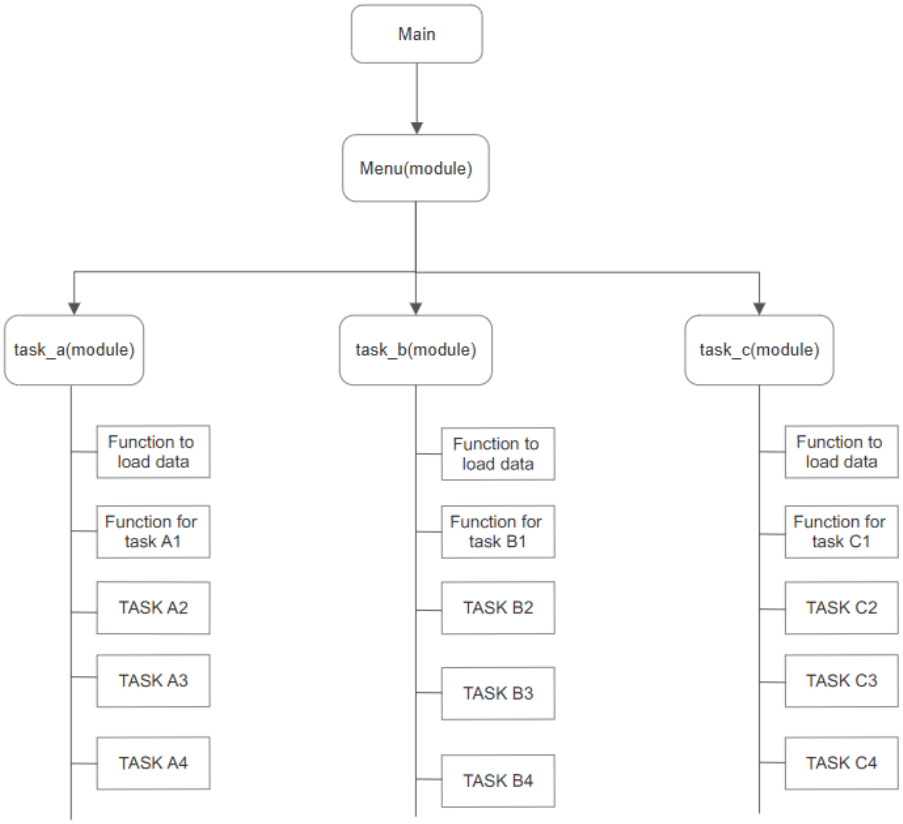

Figure 1 Sample of Project Structure

2.2 Modules/ Functions

There is a main file and total 4 modules. Each module contains different functions for different tasks.

Error Handling: Each function includes error handling for potential issues such as missing files, invalid input, or unexpected data formats. This ensures the program continues to run smoothly, providing meaningful error messages to the user when things go wrong.

2.2.1 Main file

This is the file that the user needs to open and run. It imports the main file, which serves as the core of the application and handles all user inputs and tasks. The if __name__ == "__main__" block is included to ensure that when the file is executed directly, the main() function is invoked.

This user only needs to run the main() function, and everything else, including menu navigation and task execution, is handled automatically, providing a seamless user experience.

2.2.2 Menu.py

This is the file which manages everything. First it imports task A,B and C modules/files. It contains 4 functions. main_menu, menu_task_a, menu_task_b and menu_task_c. The main menu accesses the menus for respective tasks as chosen by the user.

Main Menu:

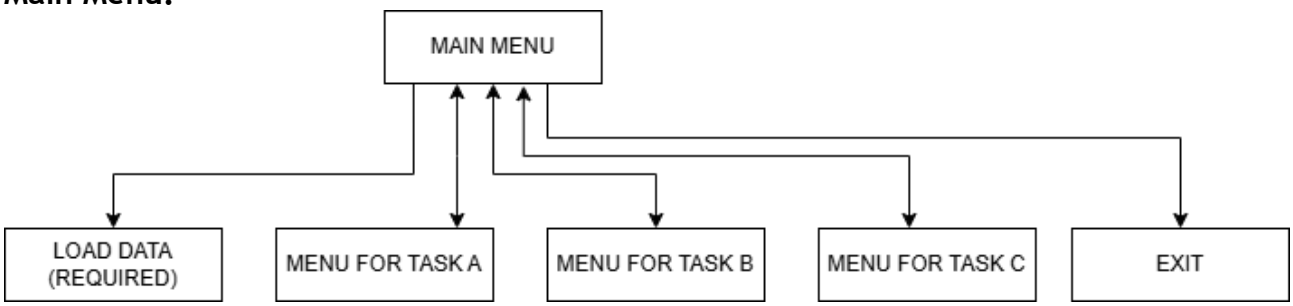

Figure 2: MENU STRUCTURE

The main_menu function initiates the program and provides the user with options: loading data, accessing Task A, Task B, Task C, or exiting. The data is initially set to None, ensuring that the user cannot access any task before loading the data. If the user selects a task before loading data, they are prompted to load it first.

```
----MAIN MENU----
1. Load Data (REQUIRED)
2. Retrieve general Information about the patients (TASK A: Basic python functions)
3. Load and Analyse the Data (TASK B: Using pandas)
4. Visualise the Data (TASK C: Using matplotlib)
5. Exit the program
Please choose one option:  2
Please load the data first using option 1

----MAIN MENU----
1. Load Data (REQUIRED)
```

Figure 3:Handling the situation if user tries to perform tasks without loading

The while True loop ensures that the menu is continuously displayed until the user decides to exit. User input is validated to ensure that only valid options are selected.

Task Menus:

Each task submenu provides the user with specific options related to the task chosen. The submenus are structured similarly, displaying options and handling user input in a loop. Before proceeding with any task, the program checks if the data is loaded, ensuring valid operations.

User has option to return to the main menu at any point.

Each task calls functions from the respective modules (task_a, task_b, task_c) to process the data and display results.

2.2.3 task_a.py

Contains 5 functions for each subtasks in TASK A.

load_data:

- Reads the CSV file specified by the user and stores its contents as a list of rows.
- Handles missing files or unexpected errors by showing appropriate error messages.

retrieve_patient_info:

- Retrieves important details (age, gender, smoking history, ethnicity) for a patient based on their ID.
- Converts rows into dictionaries to make accessing specific data easier.
- Returns a dictionary with the patient's details or a "No match found" message.

```
header =data_list[0] #first row of the data is heading/columns
rows = data_list[1:] #storing rest of the rows as rows/data

for row in rows:
    row_dictionary = dict(zip(header,row)) #converts the data into dictionary where column/header is the key and the row's data is value
    if row_dictionary.get("Patient_ID") == patient_id: # checks if it matches with the user input

        return{
            "Age":row_dictionary.get("Age"),
            "Gender":row_dictionary.get("Gender"),
            "Smoking_History":row_dictionary.get("Smoking_History"),
            "Ethnicity":row_dictionary.get("Ethnicity")} #returns the required info retrieved from the dictionary
```

Figure 4: Converting the data into dictionary and retrieving required values

retrieve_medical_history_by_ethnicity:

- Fetches and formats medical history for patients of a particular ethnicity.
- Converts inputs to lowercase to ensure matching is case-insensitive.
- Formats the output using the join function to make the data easier to read.

retrieve_treatment_details:

- Finds patients who survived more than 100 months for a specific treatment.
- Converts survival months to integers to allow comparisons.
- Handles cases where survival data is missing by using a default value.

Cancer_stage_for_adults_comorbidity:

- Analyzes cancer stages for patients under 45 with a specified comorbidity (diabetes or kidney disease).
- Validates user input and shows an error message if the input is invalid.
- Uses the join function to format the results neatly for display.

```
comorbidity= comorbidity.lower() # converts all the input into lowercase to avoid error
if comorbidity not in["diabetes", "kidney"]: #it is also in lowercase because all the input is converted into lowercase
    print(" Invalid Choice, Please Select one: Diabetes/Kidney")
    #checks if the entered info matches and prints the error message accordingly
    return None
comorbidity_column = "Comorbidity_Diabetes" if comorbidity == "diabetes" else "Comorbidity_Kidney_Disease"
# value of the comorbidity_column variable changes according to the user input
```

Figure 5: code to validate user input and handle accordingly

2.2.4 Task B file/module

Contains 5 functions for each sub tasks in TASK B.

top_3_treatments:

- Identifies the top three treatments for a specified ethnicity where patients survived more than 100 months.
- Filters data using the following conditions:
  - Ethnicity matches user input (case-insensitive and space-trimmed).
  - Survival months exceed 100.
  - **Key Method Used**: value_counts() sorts treatment frequencies automatically, making it easy to pick the top three.

average_wbc:

- Calculates the average WBC count for patients of a specific ethnicity and treatment.
- Handles case-insensitive comparisons for both Ethnicity and Treatment.
- Used .dropna() to drop missing WBC values before calculating the mean
- Prevents errors by skipping empty data and outputs a meaningful message if no records match.

average_smoking_packs:

- Computes the average smoking pack-years for different treatments under specific tumor conditions:
  - Tumor location matches user input.
  - Blood pressure is above 90.

- Tumor size is smaller than 15 mm.

- Groups the data by treatment to calculate averages for each treatment.
- Uses groupby and mean() for structured aggregation.

```
df_bp_tumor = df_main.loc[(df_main["Blood_Pressure_Pulse"]>90) & (df_main["Tumor_Size_mm"]<15) & (df_main["Tumor_Location"] == tumor_location)]
#makes a new series where only that data is present which meets our conditions
df_avg_smoking_packs = df_bp_tumor.groupby("Treatment")["Smoking_Pack_Years"].mean()
#groups the data based on treatments and then shows the average smoking packs for each treatment
```

Figure 6: usage of .loc[] and . groupby()

survival_rate_by_treatment:

- Determines the average survival months for patients receiving a specific treatment.
- Filters data to match the treatment input (case-insensitive).
- Drops missing survival values and calculates the mean.

2.2.4 Task C file/module

Contains 5 functions for each sub tasks in TASK C

load_data_task_c(file_path):

- Loads a CSV file into a Pandas DataFrame.
- Handles FileNotFoundError and general exceptions to provide clear error messages.
- Returns the DataFrame for use across other functions, ensuring data is loaded once.

treatment_proportion_by_ethnicity(df_main, ethnicity):

- Displays a pie chart showing the proportion of treatments for a specified ethnicity.
- Strips and converts user input for ethnicity to lowercase to handle case sensitivity.
- Uses .value_counts() to get treatment frequencies for the given ethnicity.
- Generates a pie chart with percentage values, ensuring visual clarity.
- The plt.legend(loc="best") dynamically selects the best position for the legend.

```
ethnicity = ethnicity.strip().lower()  # .strip() removes the extra spaces and .lower() makes the input lower to tackle case insensitivity
df_main["Ethnicity"] = df_main["Ethnicity"].str.lower()  #converts the values in ethnicity column into str and lowercase so that it matches

df_ethnicity = df_main.loc[df_main["Ethnicity"]==ethnicity]
#makes a new series where only that data is present which meets our conditions
df_treatments = df_ethnicity["Treatment"].value_counts() #retrieving the names of treatments present and how many times they are used
#now in this series, treatment names are the index and values are their count
```

Figure 7: Handling user input and filtering dataset accordingly

average_smoking_trend(df_main):

- Plots the average smoking trend across different cancer stages by ethnicity.
- Groups data by "Stage" and "Ethnicity" to compute average smoking pack years.
- Uses matplotlib for line plots, clearly distinguishing different ethnicities with a loop.

blood_pressure_comparison(df_main):

- Compares average blood pressure (systolic, diastolic, and pulse) across different treatments.
- Groups data by treatment type to compute mean values for multiple blood pressure parameters.
- Uses a bar chart with multiple bars for each treatment type, making it easy to compare different parameters.
- Applies a tight_layout() for neatness, avoiding overlap.

avg_platelets_by_stage(df_main):

- Compares the average platelet count across different cancer stages.
- Groups data by cancer stage to compute average platelet counts.
- Uses a bar chart with different colors for each stage, enhancing visual appeal.
- Rotation of x-tick labels ensures they remain readable even with many stages.
- Includes a legend and uses tight_layout() for proper chart formatting.

3. GitHub Repository Evidence

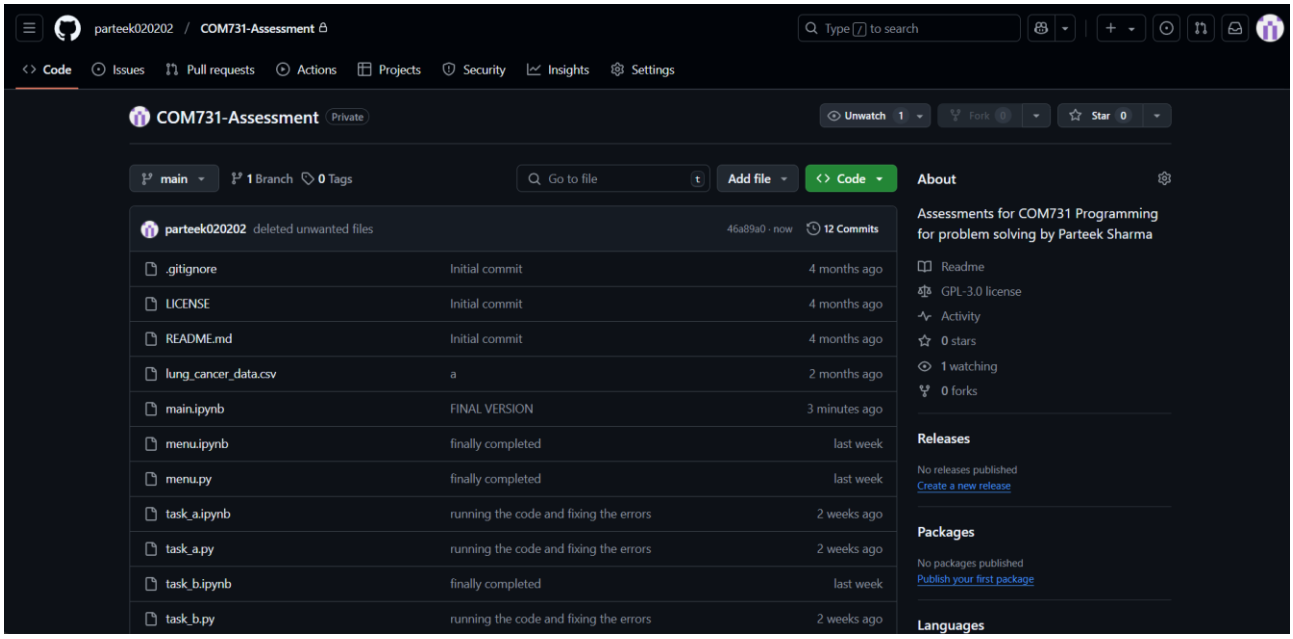

Figure 8:GitHub Repository Main

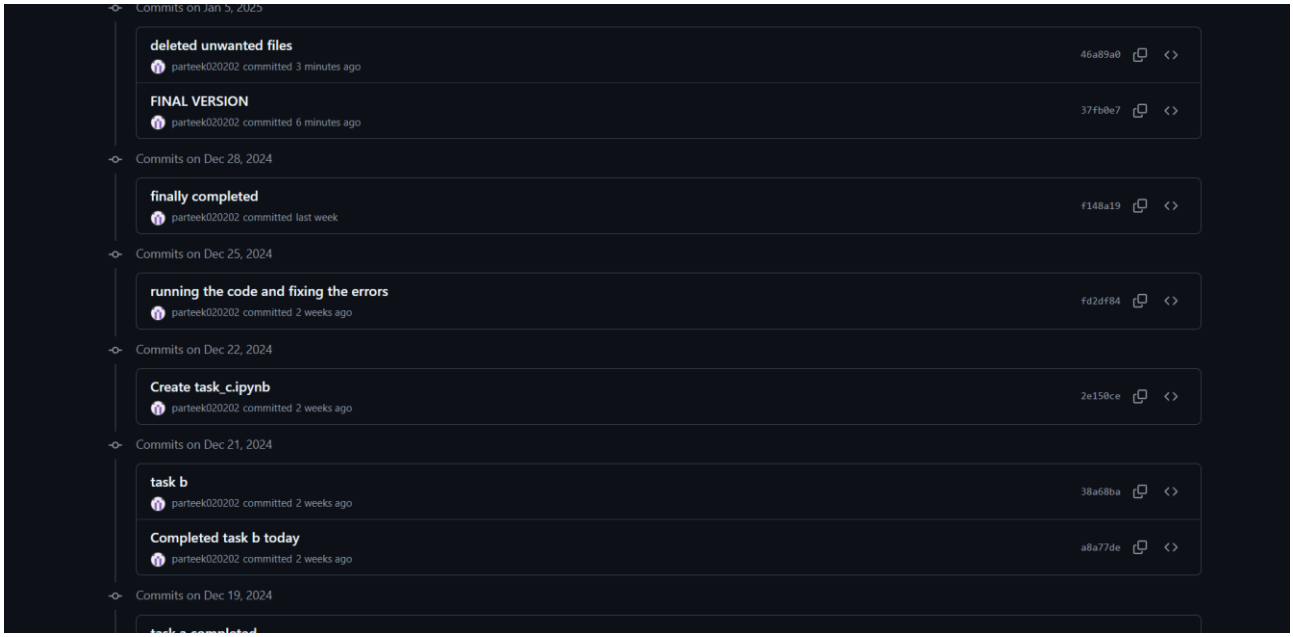

Figure 9:Sample screen shot of commit history

4. Declaration of Generative AI and AI-assisted technologies in the Programming and/or Writing Process

During the implementation of this work, I, [Parteek Sharma], utilised [ChatGPT] to

1. Problem:I wanted to convert the data list into dictionary in such a way that column is the key and the row data are values.
   - Prompt used: Tell me the easiest and most efficient method to convert the data list(converted from a csv file) into a dictionary in such a way that column is the key and the row data are values.
   - Solution: ChatGPT taught me about the “zip” function of python which then I used in my project
2. Problem: I wanted to handle the uppercase lowercase errors for user inputs.
   - Prompt used: “How to tackle the problem of user input in lowercase or upper case”
   - Solution: .lower() can be used and additional tip of using .strip() as well.
3. Problem: result output was very messy and unreadable
   - Promt: “How can I output a clearly formatted result if it is very large”
   - Solution: learned about .join() function.
4. Problem: wanted to ensure good error handling
   - Prompt: “While doing data analysis on a csv file using pandas, what are some good practices to avoid errors in data such as missing values or case sensitivity?”
   - Solution: learned about converting column into string using .str, using .dropna, using .isna etc
5. Problem: In task C, whenever any graph was getting plotted, there was a little “None” at the end.
   - Prompt: “Why is there a little None at the end of the graph plotted using function imported from another module what could be the possible reasons. Also the graph plotted are correct and there is not any other error”
   - Solution: Got to know that python has to return something or it will return just “none” so I returned a string e.g. “Pie chart”.
6. Problem: My bar chart was overlapping a little bit and I wanted to fix it.
   - Prompt: “How to fix the bar chart overlapping in matplotlib”.
   - Solution: learned about tight_layout() function.