Prediction Of Ovarian Cancer Prognosis

Using Deep Neural Network (DNN) Algorithm

Progress 5

Aisyah Binti Mohd Nadzri
&
Thuvaaritha A/P Sivarajah

Programming for Bioinformatics Project

October 2023

TABLE OF CONTENTS

TITLE	PAGE
TABLE OF CONTENTS	2
LIST OF TABLES	6
LIST OF FIGURES	7
CHAPTER 1 INTRODUCTION	
1.0 Overview	14
1.1 Introduction	14 - 15
1.2 Problem Background	15 - 16
1.3 Problem Statement	16 - 17
1.4 Research Aim and Objectives	17
1.5 Research Scope	
1.5.0 Domain of the data	18 - 19
1.5.1 Techniques to be used	19 - 20
1.5.2 Methodology	20
1.5.3 Limitation of the research	21
1.6 Chapter Summary	22
CHAPTER 2 RESEARCH METHODOLOGY	
2.1 Overview	23
2.2 Proposed Research Framework	
2.1.0 Data Collection and Preprocessing	23 - 25
2.1.1 Flowchart of the Proposed Framework	26
2.3 Performance Measurement	27
2.4 Hardware and Software Requirement	28

2.5 Methodology 28 - 30

2.6 Chapter Summary 31 - 32

CHAPTER 3 IMPORTING DATASET AND DATA WRANGLING

3.1 Overview 33

3.2 Importing Dataset

3.1.0 Understanding the data 33

3.1.1 Importing and exporting data in Python 34

3.1.2 Getting started analyzing data in Python 35 - 37

3.1.3 Python packages for Data Science 38 - 40

3.3 Data Wrangling

3.2.0 Identifying and handling missing values 41

3.2.1 Data formatting 42 - 48

3.2.2 Data normalization 49 - 50

3.2.3 Binning 51 - 53

3.2.4 Indicator variables 54

3.4 Chapter Summary 55

CHAPTER 4 EXPLORATORY DATA ANALYSIS

4.0 Overview 56

4.1 Descriptive Statistics 56 - 58

4.2 Basic of Grouping 58 - 59

4.3 ANOVA 60-61

4.4 Correlation 62

4.5 Chapter Summary 63

CHAPTER 5 MODEL DEVELOPMENT

5.0 Overview 64

5.1 Simple and Multiple Linear Regression 65 - 70

5.2 Model Evaluation Using Visualization 71 - 75

5.3 Polynomial Regression and Pipelines 76-80

5.4 R-Squared and MSE For In-Sample Evaluation 81 - 82

5.5 Prediction and Decision Making 83

5.6 Chapter Summary 84

CHAPTER 6 MODEL EVALUATION (SKLEARN)

6.0 Overview 64

6.1 Model Evaluation 65 - 70

6.2 Over-fitting, Under-fitting and Model Selection 71 - 75

6.3 Ridge Regression 76-80

6.4 Grid Search 81 - 82

6.5 Model Refinement 83

6.6 Chapter Summary 84

LIST OF TABLES

TABLE NO.	TITLE	PAGE
Table 1.5	Table 1.5.1 The criteria of dataset	19

LIST OF FIGURES

FIGURE	PAGE
Figure 2.1.0.1 Case Study	19
Figure 2.1.0.2 Datasets	20
Figure 2.1.0.3 Ovariantotal.csv dataset preview	21
Figure 2.1.1.1 Proposed Flowchart	22
Figure 3.1.1.1 Importing dataset into Google Colab using Python	30
Figure 3.1.2.1 Rows and columns of the dataset	31
Figure 3.1.2.2 Data type of each column	32
Figure 3.1.2.3 Data type of each column	33
Figure 3.2.0.1 Missing Values Identification	37
Figure 3.2.0.2 Missing Values Results	37
Figure 3.2.1.1 Data Formatting : Change original column names to specific names	39
Figure 3.2.1.2 Data Formatting : Change original column names to specific names	40
Figure 3.2.1.3 Data Formatting : Change original column names to specific names	40
Figure 3.2.1.4 Data Formatting : Change original column names to specific names output	41
Figure 3.2.1.5 Display updated column names	41
Figure 3.2.1.6 Data Formatting : Move ‘Age’ column to the first column of the dataset	42
Figure 3.2.1.7 Data Formatting : Move ‘Age’ column to the first column of the dataset output	42
Figure 3.2.1.8 Data Formatting : Move ‘Age’ column to the first column of the dataset	43
Figure 3.2.1.9 Data Formatting : Move ‘Age’ column to the first column of the dataset	44
Figure 3.2.1.10 Data Formatting : Move ‘Age’ column to the first column of the dataset	44
Figure 3.2.1.11 Data Formatting : Move ‘Age’ column to the first column of the dataset output	45
Figure 3.2.2.1 Data Normalization : Centering	46
Figure 3.2.2.2 Data Normalization : Centering	47

LIST OF FIGURES

FIGURE	PAGE
Figure 3.2.2.3 Data Normalization : Centering output	47
Figure 3.2.2.4 Data Normalization :After Centering output Visualization	47
Figure 3.2.2.5 Data Normalization :After Centering output Visualization	47
Figure 3.2.3.1 Data Binning : ‘AGE’ column	48
Figure 3.2.3.2 Data Binning : Result after Data Binning for ‘AGE’ column	48
Figure 3.2.3.3 Data Binning : Converting the ‘HGB’ unit from g/L to g/dL	49
Figure 3.2.3.4 Data Binning : Result after Converting the ‘HGB’ unit from g/L to g/dL	49
Figure 3.2.3.5 Data Binning : ‘HGB’ & ‘PLT’ Columns	50
Figure 3.2.3.6 Data Binning : Result after Data Binning for ‘HGB’ & ‘PLT’ Columns	50
Figure 3.2.4.1 Indicator Variables : Using 0 and 1	51
Figure 4.1.1 Descriptive Statistics : Summary of Main Characteristics	56
Figure 4.1.2 Descriptive Statistics : Summary of Main Characteristics Results	57
Figure 4.1.3 Descriptive Statistics : Summary of Main Characteristics Results	57
Figure 4.1.4 Descriptive Statistics : Summary of Main Characteristics Results	58
Figure 4.1.5 Descriptive Statistics : Results after performing ‘df.describe()’	58
Figure 4.2.1 Basics of Grouping : Method	59
Figure 4.2.2 Basics of Grouping : Method and Results	59
Figure 4.3.1 ANOVA : ANOVA on the Platelet Count variable across different age groups	60
Figure 4.3.2 ANOVA : Results	61
Figure 4.4.1 Correlation : Correlation between ‘PLT’ and ‘AGE’	62
Figure 4.4.2 Correlation : Correlation Results	62
Figure 5.1.1 Simple and Multiple Linear Regression : Using ‘CA125’ as dependent variable	65
Figure 5.1.2 Simple and Multiple Linear Regression : Results	66

LIST OF FIGURES

FIGURE	PAGE
Figure 5.1.3 Simple and Multiple Linear Regression : Visual Representation for Simple Linear Regression	68
Figure 5.1.4 Simple and Multiple Linear Regression : Visual Representation Results	69
Figure 5.1.5 Simple and Multiple Linear Regression : Visual Representation for Multiple Linear Regression	70
Figure 5.1.6 Simple and Multiple Linear Regression : Visual Representation Results	70
Figure 5.2.1 Model Evaluation Using Visualization : No specific nature of target variable	72
Figure 5.2.2 Model Evaluation Using Visualization : No specific nature of target variable	73
Figure 5.2.3 Model Evaluation Using Visualization : No specific nature of target variable	73
Figure 5.2.4 Model Evaluation Using Visualization : Results	74
Figure 5.2.5 Model Evaluation Using Visualization : Binary Target Variable	74
Figure 5.2.6 Model Evaluation Using Visualization : Binary Target Variable	75
Figure 5.2.7 Model Evaluation Using Visualization : Binary Target Variable	75
Figure 5.2.8 Model Evaluation Using Visualization : Results	76
Figure 5.3.1 Polynomial Regression and Pipelines : Polynomial Regression Without Pipelines	77
Figure 5.3.2 Polynomial Regression and Pipelines : Results of Polynomial Regression Without Pipelines	78
Figure 5.3.3 Polynomial Regression and Pipelines : Polynomial Regression With Pipelines	79
Figure 5.3.4 Polynomial Regression and Pipelines :Results of Polynomial Regression With Pipelines	80
Figure 5.4.1 R-Squared and MSE for In-Sample Training	82
Figure 5.4.2 R-Squared and MSE for In-Sample Training : Results	82
Figure 5.5.1 Prediction and Decision Making	83
Figure 5.5.2 Prediction and Decision Making : Results	83
Figure 6.1.1 Model Evaluation	90
Figure 6.1.2 Model Evaluation : Results	91
Figure 6.2.1 Over-fitting, Under-fitting and Model Selection	92

LIST OF FIGURES

FIGURE	PAGE
Figure 6.2.2 Over-fitting, Under-fitting and Model Selection : Results	93
Figure 6.3.1 Ridge Regression	94
Figure 6.3.2 Ridge Regression : Results	95
Figure 6.4.1 Grid Search	96
Figure 6.4.2 Grid Search : Results	97
Figure 6.5.1 Model Refinement	98
Figure 6.5.2 Model Refinement : Results	99

CHAPTER 1

INTRODUCTION

1.0 Overview

This chapter discusses the introduction to case study, problem background, problem statement, research aim and objectives. Research scope is stated clearly in this chapter which include the domain of the data, techniques to be used, methodology and limitations of the research.

1.1 Introduction to Case Study

Ovarian cancer, a formidable adversary in the realm of women's health, is known for its silent invasion and late-stage diagnosis, often making it challenging to treat effectively. This cancer is called forgotten cancer and is sometimes misdiagnosed (Tan JH et al,. 2020). It has been a constant concern for both patients and medical professionals. Despite advances in therapy, ovarian cancer remains the most deadly of the gynecological cancers where less than 30% of women with advanced stage disease survive long term (Badgwell D & Bast RC Jr., 2007). Improving the prognosis of ovarian cancer patients has become an urgent and essential mission in the field of oncology,

Despite efforts, accuracy in predicting ovarian cancer remains inadequate. Various studies have explored the efficiency of biomarkers, yielding diverse results. To address this, our study proposes a comprehensive approach. Firstly, we emphasize the importance of early detection of ovarian cancer, with a focus on identifying the disease in its initial stages using biomarkers. Secondly, we employ a dual methodology, incorporating both statistical and machine learning

methods to identify significant and associative biomarkers. Notably, our study introduces a novel dimension by utilizing a comprehensive dataset that includes blood samples, general chemistry tests, and ovarian cancer markers. Thirdly, we apply advanced machine learning models to this dataset, aiming for robust and statistically sound analyses. Previous studies encountered challenges, such as sensitivity to feature set size and overfitting.

Through this integrated approach, our study seeks to enhance the accuracy of clinical testing, providing a more dependable tool for diagnosing ovarian cancer. The amalgamation of statistical and machine learning methodologies introduces a nuanced perspective, offering potential improvements over previous studies and addressing the existing gaps in accuracy. In this study, a deep learning approach is employed to demonstrate its capability and effectiveness in the classification of ovarian cancer prognosis. Accurate classifications have the potential to guide treatment strategies, enhance survival rates, and ultimately contribute to better outcomes for ovarian cancer patients.

1.2 Problem Background

Cancer is a disease in which some of the body's cells grow uncontrollably and spread to other parts of the body. The fundamental abnormality resulting in the development of cancer is the continual unregulated proliferation of cancer cells. Likewise, ovarian cancer is a type of cancer that begins in the ovaries, considering that this cancer is considered as one of the most deadly causes of death among women. Ovarian cancer is a highly malignant gynecological tumor that has a poor prognosis, reducing the chances of successful treatments and survival as this disease is often diagnosed at an advanced stage. Forasmuch as, due to inadequate screening instruments and difficulties in timely detection, 80% of patients with ovarian cancer are

diagnosed at an advanced stage and 50% - 70% patients will experience recurrence within two years of finishing therapy with a poor 5-year survival rate of 30%. Currently, the main treatments for ovarian cancer are platinum-based chemotherapies and surgery. The 5 - year survival rate has been slowly improving despite recent advancements in therapy. Considering the limitations of ovarian cancer treatments, reliable innovative prognostic models are desperately needed to increase the viability of targeted therapies since the limits of ovarian cancer treatments need the development of new therapeutic targets in order to enhance the clinical outcome of ovarian cancer.

1.3 Problem Statement

The classification of ovarian cancer prognosis presents a pressing challenge in the field of oncology due to the disease's elusive nature, late-stage diagnoses, and limited accuracy in predicting patient outcomes. Ovarian cancer, often referred to as the "forgotten cancer," is characterized by subtle progression, leading to difficulties in effective intervention. Current prognostic methods, despite therapeutic advancements, exhibit shortcomings in providing accurate classifications, resulting in a significant gap in understanding treatment effectiveness and long-term survivorship.

The urgency of this problem is underscored by the persistently high fatality rates, particularly in advanced-stage ovarian cancer, where less than 30% of women achieve long-term survival. Achieving a precise classification of ovarian cancer prognosis is crucial for guiding personalized treatment strategies, enhancing survival rates, and improving the overall quality of life for survivors. The limitations of existing prognostic models highlight the need for innovative approaches, such as leveraging advanced classification techniques, to harness the wealth of available biomedical and clinical data and transform the accuracy and reliability of ovarian cancer prognostication. Addressing this problem is imperative for advancing patient care,

improving clinical decision-making, and ultimately contributing to better outcomes for individuals affected by ovarian cancer.

1.4 Research Aim and Objectives

The aim of this research is to implement and develop classification models that can produce high classification accuracy for diagnosing ovarian cancer prognosis using Deep Neural Network (DNN) Algorithm. In accordance with the accomplishment of this project, these are the specific aims and objectives that we wish to achieve.

1. To identify the algorithm that can be used for classification of ovarian cancer prognosis.
2. To implement a prediction model by applying a deep learning algorithm for classification of ovarian cancer.
3. To evaluate the accuracy of the proposed classification model performance.

1.5 Scopes

1.5.0 Domain of the data

In this research, the dataset criteria used are the same as those used in earlier studies by J Pers Med (2022) which has been extracted from NCBI (National Library of Medicine) PubMed Central article site. The dataset consists of three subgroups of pathology diagnosis resulting in this dataset has 49 features. Those three subgroups are blood routine tests, general chemistry and tumor markers. The study was conducted from July 2011 to July 2018. The dataset consists of data of 349 patients which include 171 ovarian cancer patients and 178 benign ovarian tumor patients. The criteria of the dataset are shown in Table 3.1.

Table 1.5 The criteria of dataset

Criteria	Value
Sample Type	Blood routine tests, general chemistry and tumor markers
Disease state	Severe to death
Tissue	Ovary
Ovarian cancer type	Ovarian cancer: 171 patients Benign ovarian tumor patients: 178 patients
Institute	Third Affiliated Hospital of Soochow University

The downloaded dataset is retrieved from NCBI. However the sources are all embedded in github. All of the datasets retrieved are in .csv extension.The file then opens using Microsoft Excel to view in excel workbook. The pathology information of all samples provided in the dataset would be used as the benchmark to classify the samples. The

dataset will undergo the pre-processing and exploratory analysis to be cleaned before being used in the research.

1.5.1 Techniques to be used

1. Deep Neural Networks (DNN)

   The primary technique that would be used is DNN as it is used for feature extraction. It plays a central role in our research for classification of ovarian cancer prognosis.

2. Machine Learning for Predictive Modeling

   While DNN is highlighted, this project is likely to include machine learning techniques for building predictive models. These includes various algorithms such as Logistic Regression (LR), Support Vector Machines (SVM), Decision Tree (DT), Gradient Boosting Machine (GBM), Light Gradient Boosting Machine (LGBM) and Random Forest (RF)

3. Evaluation of Classification Model

   This study uses several evaluation metrics, namely accuracy, precision, recall, F-score based on the True Positives (*TP*), False Positives (*FP*), True Negatives (*TN*) and True Positives (*TP*).

4. Data Retrieval and Preprocessing

In order to clean, format and prepare the dataset for analysis, we will first handle the missing data followed by data formatting, data normalization and data binning.

1.5.2 Methodology

Notwithstanding what has just been said, in order to analyze the dataset we have chosen, we should break it down into several methods of analyzation in which as below.

- Approach: Put Deep Neural Networks (DNN) to use as DNN models have the potential to provide new insights in the study of cognitive processes in which it imitates human decision making as it is high in capacity and data -driven design.
- Testing/experimental tools/measurement: Principal Component Analysis (PCA) for data preprocessing. PCA can be applied to reduce the dimensionality of metadata while retaining most of the relevant information. PCA can be a valuable method for data preprocessing.
- Test data: Utilize Scikit-Learn Python for testing predictive models in Python. The processes involved are data splitting, model training, model evaluation, etcetera.

1.5.3 Limitation of the research

1. Late Diagnosis

   Ovarian cancer is often diagnosed at an advanced stage due to the lack of specific early symptoms and effective screening methods. This late diagnosis significantly reduces the chances of successful treatment.

2. Heterogeneity

   Ovarian cancer is not a single disease but a group of distinct subtypes each with different characteristics and responses to treatment. This complicates research efforts to find a one-size-fits-all solution.

3. Limited biomarkers

   The lack of highly specific and sensitive biomarkers for early detection and monitoring of ovarian cancer is a significant challenge as the existing ones lack the accuracy needed for screening.

4. Data Source

   The data source mentioned is from a specific time frame (August 2008 to March 2012), and ovarian cancer research and treatments may have evolved since then. Using older data might not reflect the current state of the disease

1.5 Chapter Summary

To sum up, an essential component of enhancing patient outcomes and customizing treatment regimens is predicting the prognosis of ovarian cancer. In this work, researchers used Deep Neural Networks (DNN) to create a predictive model based on a dataset that included 54 samples from patients with ovarian cancer which it retrieved from Gene Expression Omnibus (GEO). DNNs are a subclass of deep learning methods that are especially useful for managing high-dimensional, complicated datasets, such as those used in medical research.

In addition to that, to guarantee its quality, we will be starting by preprocessing and cleaning the dataset. We then used deep neural networks (DNNs) to mine the data for complex patterns and associations that would help us develop a prediction model for the prognosis of ovarian cancer. The model was able to evaluate and forecast the probability of distinct prognostic outcomes by feeding the DNNs with a variety of clinical and molecular data. This allowed the model to offer important insights into the course of the disease and support the development of individualized treatment plans.

Last but foremost, by utilizing deep learning techniques, the findings of this study present a viable option for improving the care of patients with ovarian cancer. This study should assist in advancing the continuing endeavor to enhance early detection and treatment approaches for ovarian cancer by utilizing the predictive powers of DNNs, which will eventually improve patient outcomes and raise survival rates.

CHAPTER 2

RESEARCH METHODOLOGY

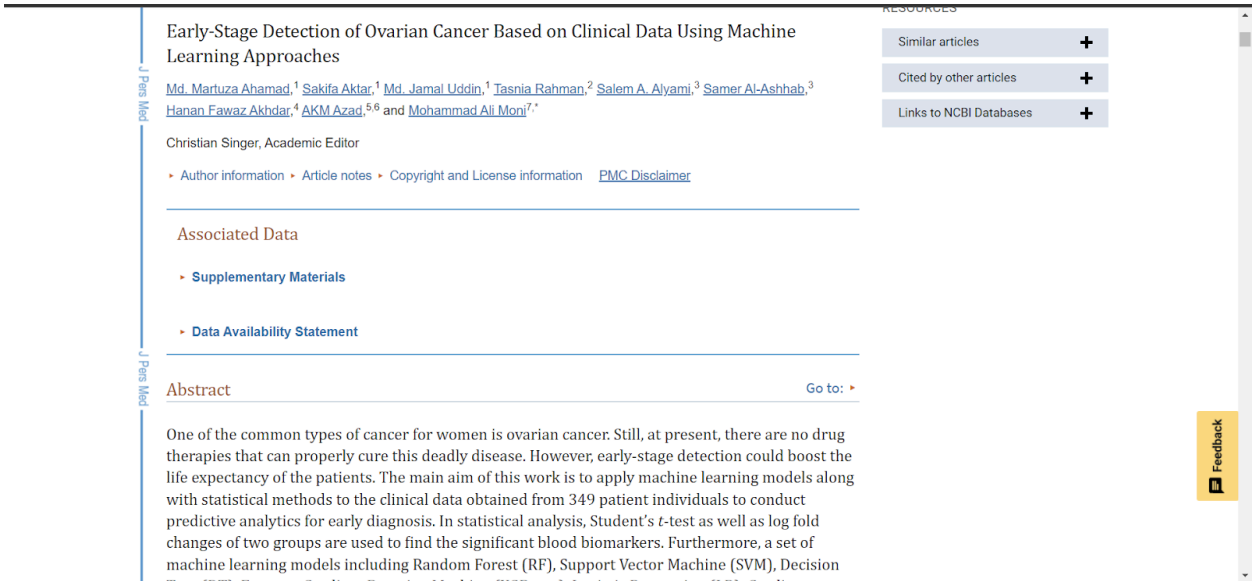
2.0 Overview

This chapter discusses the proposed framework,flowchart of proposed framework, performance measurement, hardware and software requirement.

2.1 Proposed Framework

2.1.0 Data Collection and Preprocessing

The Ovarian Cancer dataset was obtained from National Library of Medicine (NCBI) through the link https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9394434/#B29-jpm-12-01211



Early-Stage Detection of Ovarian Cancer Based on Clinical Data Using Machine Learning Approaches

Md. Martuza Ahamad,[1] Sakifa Aktar,[1] Md. Jamal Uddin,[1] Tasnia Rahman,[2] Salem A. Alyami,[3] Samer Al-Ashhab,[3] Hanan Fawaz Akhdar,[4] AKM Azad,[5,6] and Mohammad Ali Moni[7,*]

Christian Singer, Academic Editor

Author information · Article notes · Copyright and License information · PMC Disclaimer

Associated Data

Supplementary Materials

Data Availability Statement

Abstract

Go to:

One of the common types of cancer for women is ovarian cancer. Still, at present, there are no drug therapies that can properly cure this deadly disease. However, early-stage detection could boost the life expectancy of the patients. The main aim of this work is to apply machine learning models along with statistical methods to the clinical data obtained from 349 patient individuals to conduct predictive analytics for early diagnosis. In statistical analysis, Student's *t*-test as well as log fold changes of two groups are used to find the significant blood biomarkers. Furthermore, a set of machine learning models including Random Forest (RF), Support Vector Machine (SVM), Decision

Similar articles

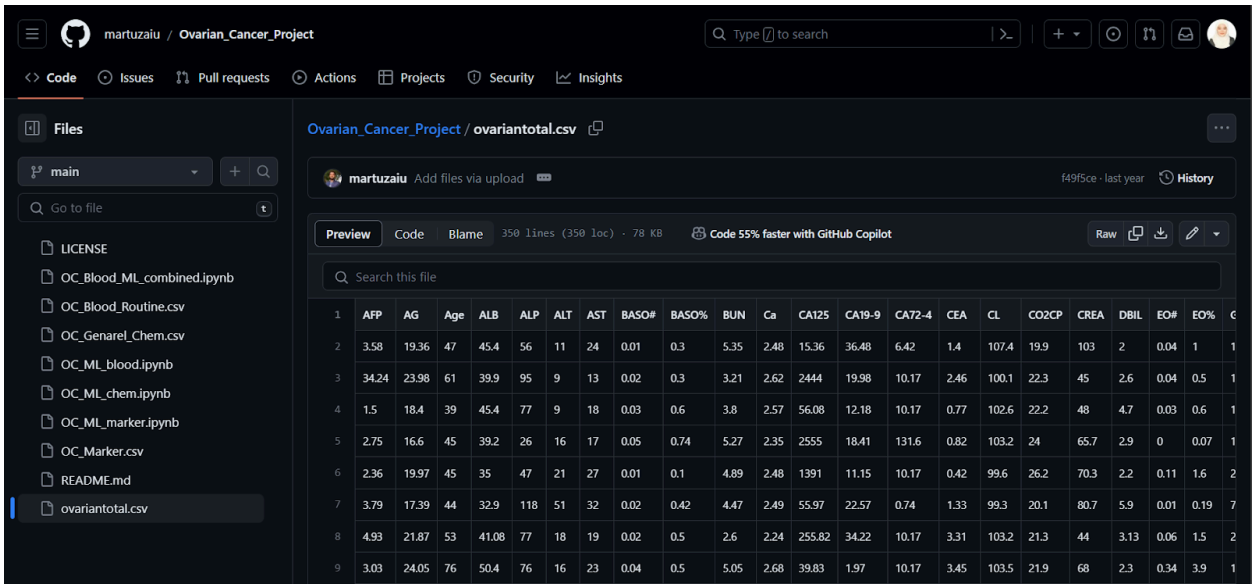
Cited by other articles

Links to NCBI Databases

Feedback

Figure 2.1.0.1 Case Study

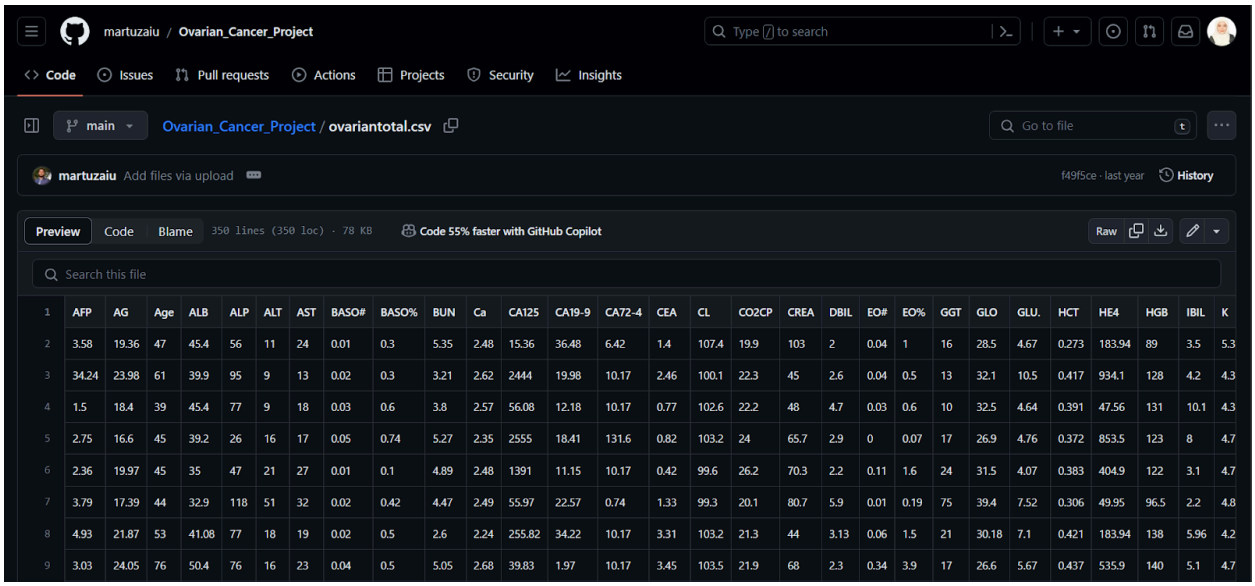
We can view the data availability from the github link provided by the authors. It displays here the raw datasets of combined dataset, blood test samples dataset, general chemistry test dataset and cancer marker dataset. The authors likewise provided the jupyter notebook version of each datasets.



1	AFP	AG	Age	ALB	ALP	ALT	AST	BASO#	BASO%	BUN	Ca	CA125	CA19-9	CA72-4	CEA	CL	CO2CP	CREA	DBIL	EO#	EO%	G
2	3.58	19.36	47	45.4	56	11	24	0.01	0.3	5.35	2.48	15.36	36.48	6.42	1.4	107.4	19.9	103	2	0.04	1	1
3	34.24	23.98	61	39.9	95	9	13	0.02	0.3	3.21	2.62	2444	19.98	10.17	2.46	100.1	22.3	45	2.6	0.04	0.5	1
4	1.5	18.4	39	45.4	77	9	18	0.03	0.6	3.8	2.57	56.08	12.18	10.17	0.77	102.6	22.2	48	4.7	0.03	0.6	1
5	2.75	16.6	45	39.2	26	16	17	0.05	0.74	5.27	2.35	2555	18.41	131.6	0.82	103.2	24	65.7	2.9	0	0.07	1
6	2.36	19.97	45	35	47	21	27	0.01	0.1	4.89	2.48	1391	11.15	10.17	0.42	99.6	26.2	70.3	2.2	0.11	1.6	2
7	3.79	17.39	44	32.9	118	51	32	0.02	0.42	4.47	2.49	55.97	22.57	0.74	1.33	99.3	20.1	80.7	5.9	0.01	0.19	7
8	4.93	21.87	53	41.08	77	18	19	0.02	0.5	2.6	2.24	255.82	34.22	10.17	3.31	103.2	21.3	44	3.13	0.06	1.5	2
9	3.03	24.05	76	50.4	76	16	23	0.04	0.5	5.05	2.68	39.83	1.97	10.17	3.45	103.5	21.9	68	2.3	0.34	3.9	1

Figure 2.1.0.2 Datasets

Here we get a quick look of how the dataset looks like.



1	AFP	AG	Age	ALB	ALP	ALT	AST	BASO#	BASO%	BUN	Ca	CA125	CA19-9	CA72-4	CEA	CL	CO2CP	CREA	DBIL	EO#	EO%	GGT	GLO	GLU.	HCT	HE4	HGB	IBIL	K
2	3.58	19.36	47	45.4	56	11	24	0.01	0.3	5.35	2.48	15.36	36.48	6.42	1.4	107.4	19.9	103	2	0.04	1	16	28.5	4.67	0.273	183.94	89	3.5	5.3
3	34.24	23.98	61	39.9	95	9	13	0.02	0.3	3.21	2.62	2444	19.98	10.17	2.46	100.1	22.3	45	2.6	0.04	0.5	13	32.1	10.5	0.417	934.1	128	4.2	4.3
4	1.5	18.4	39	45.4	77	9	18	0.03	0.6	3.8	2.57	56.08	12.18	10.17	0.77	102.6	22.2	48	4.7	0.03	0.6	10	32.5	4.64	0.391	47.56	131	10.1	4.3
5	2.75	16.6	45	39.2	26	16	17	0.05	0.74	5.27	2.35	2555	18.41	131.6	0.82	103.2	24	65.7	2.9	0	0.07	17	26.9	4.76	0.372	853.5	123	8	4.7
6	2.36	19.97	45	35	47	21	27	0.01	0.1	4.89	2.48	1391	11.15	10.17	0.42	99.6	26.2	70.3	2.2	0.11	1.6	24	31.5	4.07	0.383	404.9	122	3.1	4.7
7	3.79	17.39	44	32.9	118	51	32	0.02	0.42	4.47	2.49	55.97	22.57	0.74	1.33	99.3	20.1	80.7	5.9	0.01	0.19	75	39.4	7.52	0.306	49.95	96.5	2.2	4.8
8	4.93	21.87	53	41.08	77	18	19	0.02	0.5	2.6	2.24	255.82	34.22	10.17	3.31	103.2	21.3	44	3.13	0.06	1.5	21	30.18	7.1	0.421	183.94	138	5.96	4.2
9	3.03	24.05	76	50.4	76	16	23	0.04	0.5	5.05	2.68	39.83	1.97	10.17	3.45	103.5	21.9	68	2.3	0.34	3.9	17	26.6	5.67	0.437	535.9	140	5.1	4.7

Figure 2.1.0.3 Ovariantotal.csv dataset preview

The dataset is not yet explored in depth, however it will still be needing to undergo data preprocessing.

2.1.1 Flowchart of the Proposed Framework

To preview the proposed solution of ours, it can be viewed through the following flowchart.

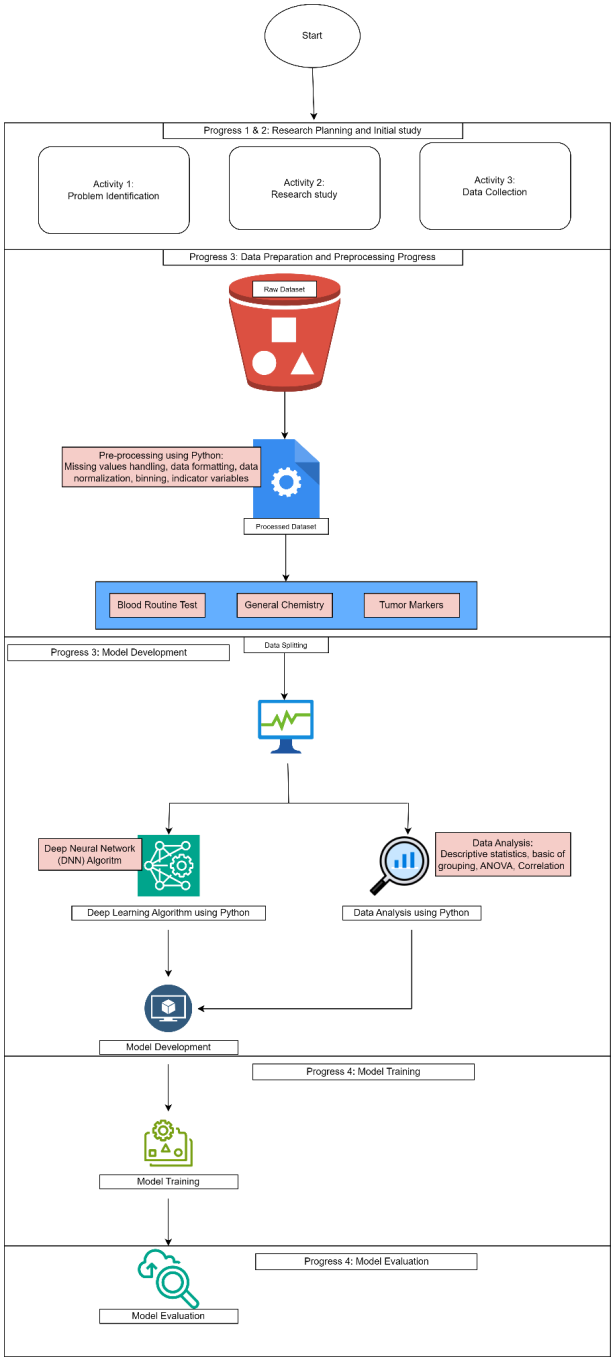


Figure 2.1.1.1 Proposed Flowchart

2.2 Performance Measurement

1. Accuracy

   Accuracy represents the correctness of a model, and it can be expressed as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision

   Precision states the percentage of appropriately identified samples (positive) inside all identified samples (positive), which can be stated as follows:

$$Precision = \frac{TP}{TP + FP}$$

3. Recall

   Recall expresses the capacity of the classifier to properly classify samples within a given class, which is as follows:

$$Recall = \frac{TP}{TP + FN}$$

4. F1 Score

   F1-score is used for the case when there is class imbalance in data by harmonizing Precision and Recall, which is as follows:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

2.3 Hardware and Software Requirements

In this research, several basic requirements have been identified in order to reduce the processing time especially in the process of designing and developing Deep Neural Network (DNN) for cancer prediction. Hardware requirements for this research is at least a device with Intel Core i5 processor, Windows 11 operating system and 4GB RAM memory. Alternate to that, the software requirements are like Visual Studio Code, Python programming, Tensorflow, GitHub and Google Docs.

2.4 Methodology

Phase 1 : Research Planning and Initial Study

1. Literature Review

- Review existing literature on ovarian cancer, predictive models and deep learning applications in healthcare.
- Gain a comprehensive understanding of the disease, its prognosis and related research.

2. Data Collection

- Retrieve the ovarian cancer dataset from the National Library of Medicine (NCBI) through the link provided above.
- Verify the completeness and quality of the dataset in the next step.

Phase 2 : Data Preparation and Preprocessing

3. Data Preprocessing

- This is to handle missing data, outliers as well as any data quality issues.

4. Data Formatting

- This is crucial for data analysis and data science as it ensures consistency, integrity, and compatibility of datasets.

5. Data Normalization

- Enhances the stability and efficiency of models, leading to improved performance and more reliable insights from the data.
- Consider centering and scaling for data normalization.

6. Data Binning

- A process of transforming continuous numerical variables into discrete categorical 'bins', for grouped analysis

Phase 3 : Development

7. Prediction Model Development

- Implement a Deep Neural Network (DNN) model for classification ovarian cancer prognosis

Phase 4 : Testing and Evaluation Phase

8. Classification Result

- Test the final model using the test dataset to assess its performance

9. Evaluation of model's performance / Performance Measurement

- Evaluate the model's performance using various metrics, as stated above. Those metrics include accuracy, precision, recall and F1 score.

10. Report and Documentation

- Prepare a comprehensive research report that documents the entire process, including methodology, results and conclusions.
- Cite all references and provide a clear overview of the work done.

2.5 Chapter Summary

In conclusion, the summary to this chapter embodies a multifaceted approach to address the critical issue of ovarian cancer prognosis. The project's success hinges on a comprehensive set of methods and techniques that encompass the entire research process. Beginning with a thorough literature review, the project gains a holistic understanding of ovarian cancer, predictive models, and the applications of deep learning in healthcare. Data collection, achieved through the retrieval of the ovarian cancer dataset from the NCBI database, is an essential initial step. The dataset, which features blood test samples, cancer markers and general chemistry tests will be transformed into a usable format for analysis, including the handling of missing data and data quality issues. To prepare the dataset for analysis, methods such as Principal Component Analysis (PCA) for dimensionality reduction and data normalization, which ensures gene expression levels' comparability, will be applied. Data analysis will involve the identification of differentially expressed genes (DEGs) and exploring additional data types such as mean platelet volume, red blood count and their relationships with gene expression. The project's core component is the development of a deep neural network (DNN) model for predicting ovarian cancer prognosis. The DNN model, being data-driven and capable of high-capacity decision-making, plays a pivotal role in the research. Model training and testing involve various techniques, including Scikit-Learn Python for predictive model testing and model evaluation. Performance measurement is a critical aspect, and the project employs a range of metrics such as accuracy, precision, sensitivity, specificity, F1 score, and cross-validation to assess the model's performance comprehensively. Mean Absolute Error (MAE) is also applied to evaluate the model's predictions against actual values. The Receiver Operating Characteristic (ROC) curve visually represents the trade-off between sensitivity and specificity, allowing for the selection of

the optimal operating point based on priorities. The entire research process is documented meticulously, with a focus on methodology, results, and conclusions, and is supported by proper citations and references. This project's extensive and well-structured approach is poised to contribute significantly to the advancement of ovarian cancer research and the enhancement of patient outcomes by improving prognosis accuracy and treatment strategies.

CHAPTER 3

IMPORTING DATASET AND DATA WRANGLING

3.0 Overview

This chapter discusses the way of importing dataset in Python and understanding the dataset, analyzing dataset, python packages used and data wrangling.

3.1 Importing Dataset

3.1.0 Understanding the Data

In this research, there is a dataset being used in which Ovarian Cancer Combined dataset. This dataset is the combination sub groups result of blood samples dataset, general chemistry tests dataset and cancer marker dataset. This dataset is obtained from the NCBI database and collected from the Third Affiliated Hospital of Soochow University. The dataset consists of data from 171 ovarian cancer patients and 178 benign ovarian tumor patients which makes it has about 349 entries and 49 features.

3.1.1 Importing and Exporting Data in Python

Here the raw dataset is imported to Google Colab using Python Programming.

```
[ ] import pandas as pd
    import numpy as np
```

```
# 3.1: Importing Dataset

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Display the first few rows of the DataFrame
df.head()
```

	AFP	AG	Age	ALB	ALP	ALT	AST	BASO#	BASO%	BUN	...	PCT	PDW	PHOS	PLT	RBC	RDW	TBIL	TP	UA	TYPE
0	3.58	19.36	47	45.4	56	11	24	0.01	0.30	5.35	...	0.09	13.4	1.46	74	2.64	13.7	5.5	73.9	396.4	0
1	34.24	23.98	61	39.9	95	9	13	0.02	0.30	3.21	...	0.30	11.2	1.09	304	4.89	12.7	6.8	72.0	119.2	0
2	1.50	18.40	39	45.4	77	9	18	0.03	0.60	3.80	...	0.13	15.2	0.97	112	4.62	12.0	14.8	77.9	209.2	0
3	2.75	16.60	45	39.2	26	16	17	0.05	0.74	5.27	...	0.25	17.4	1.25	339	4.01	14.6	10.9	66.1	215.6	0
4	2.36	19.97	45	35.0	47	21	27	0.01	0.10	4.89	...	0.28	11.9	0.94	272	4.40	13.4	5.3	66.5	206.0	0

5 rows × 50 columns

Figure 3.1.1.1 Importing dataset into Google Colab using Python

3.1.2 Getting Started Analyzing Data in Python

The data analyzation is started with getting know the number of rows and columns the dataset has.

```
# 3.1.2 Getting Started Analyzing Data in Python

# a) To display the number of rows and columns in the DataFrame
print("Number of rows:", df.shape[0])
print("Number of columns:", df.shape[1])
```

```
Number of rows: 349
Number of columns: 50
```

Figure 3.1.2.1 Rows and columns of the dataset

The analysis is then proceeded with getting to know the data types of each columns.

```
# 3.1.2 Getting Started Analyzing Data in Python

# b) To display the data types of each column in the DataFrame
print(df.dtypes)
```

```
AFP          float64
AG           float64
Age            int64
ALB          float64
ALP            int64
ALT            int64
AST            int64
BASO#        float64
BASO%        float64
BUN          float64
Ca           float64
CA125        float64
CA19-9       float64
CA72-4       float64
CEA          float64
CL           float64
CO2CP        float64
CREA         float64
DBIL         float64
EO#          float64
EO%          float64
GGT            int64
GLO          float64
GLU.         float64
HCT          float64
HE4          float64
```

Figure 3.1.2.2 Data type of each column

```
HGB          float64
IBIL         float64
K            float64
LYM#         float64
LYM%         float64
MCH          float64
MCV          float64
Menopause      int64
Mg           float64
MONO#        float64
MONO%        float64
MPV          float64
Na           float64
NEU          float64
PCT          float64
PDW          float64
PHOS         float64
PLT            int64
RBC          float64
RDW          float64
TBIL         float64
TP           float64
UA           float64
TYPE           int64
dtype: object
```

Figure 3.1.2.3 Data type of each column

3.3.3 Python Packages for Data Science

1. Tensorflow

Features:

- Better computational graph visualizations
- Reduces error by 50 to 60 percent in neural machine learning
- Parallel computing to execute complex models
- Seamless library management backed by Google
- Quicker updates and frequent new releases to provide you with the latest features

Applications:

- Speech and image recognition
- Text-based applications
- Time-series analysis
- Video detection

2. Numpy

Features:

- Provides fast, precompiled functions for numerical routines
- Array-oriented computing for better efficiency
- Supports an object-oriented approach
- Compact and faster computations with vectorization

Applications:

- Extensively used in data analysis
- Creates powerful N-dimensional array
- Forms the base of other libraries, such as SciPy and scikit-learn
- Replacement of MATLAB when used with SciPy and matplotlib

3. Pandas

Features:

- Eloquent syntax and rich functionalities that gives you the freedom to deal with missing data
- Enables you to create your own function and run it across a series of data
- High-level abstraction
- Contains high-level data structures and manipulation tools

Applications:

- General data wrangling and data cleaning
- ETL (extract, transform, load) jobs for data transformation and data storage, as it has excellent support for loading CSV files into its data frame format
- Used in a variety of academic and commercial areas, including statistics, finance and neuroscience
- Time-series-specific functionality, such as date range generation, moving window, linear regression and date shifting.

4. Scikit-learn

Applications:

- clustering
- classification
- regression
- model selection
- dimensionality reduction

5. Matplotlib

Features:

- Usable as a MATLAB replacement, with the advantage of being free and open source
- Supports dozens of backends and output types, which means you can use it regardless of which operating system you're using or which output format you wish to use
- Pandas itself can be used as wrappers around MATLAB API to drive MATLAB like a cleaner
- Low memory consumption and better runtime behavior

Applications:

- Correlation analysis of variables
- Visualize 95 percent confidence intervals of the models
- Outlier detection using a scatter plot etc.
- Visualize the distribution of data to gain instant insights

3.2 Data Wrangling (Pandas Numpy)

3.2.0 Identifying and Handling Missing Values

Identifying Missing Values using pandas.

```python
# 3.2.0 Identifying and handling missing values

# Identify missing values in the DataFrame
missing_values = df.isnull().sum()

# Display the count of missing values for each column
print("Missing values in the dataset:")
print(missing_values)

# Check if there are any missing values
if missing_values.sum() == 0:
    print("\nThere are no missing values in the dataset.")
else:
    print("\nThere are missing values in the dataset.")
```

Figure 3.2.0.1 Missing Values Identification

```
Missing values in the dataset:
AFP          0
AG           0
Age          0
ALB          0
ALP          0
ALT          0
AST          0
BASO#        0
BASO%        0
BUN          0
Ca           0
CA125        0
CA19-9       0
CA72-4       0
CEA          0
CL           0
CO2CP        0
CREA         0
DBIL         0
EO#          0
EO%          0
GGT          0
GLO          0
GLU.         0
HCT          0
HE4          0
HGB          0
```

```
HGB          0
IBIL         0
K            0
LYM#         0
LYM%         0
MCH          0
MCV          0
Menopause    0
Mg           0
MONO#        0
MONO%        0
MPV          0
Na           0
NEU          0
PCT          0
PDW          0
PHOS         0
PLT          0
RBC          0
RDW          0
TBIL         0
TP           0
UA           0
TYPE         0
dtype: int64

There are no missing values in the dataset.
```

Figure 3.2.0.2 Missing Values Results

This prints the count of missing values in each column of our dataset. However, there does not seem to be any missing values in our dataset because we chose a dataset where

the missing values have been identified and handled. The original dataset had 7% of missing values but the original dataset was not provided, therefore we proceeded with the clean dataset.

3.2.1 Data Formatting

Data formatting is crucial for data analysis and data science as it ensures consistency, integrity, and compatibility of datasets. Well-formatted data facilitates efficient analysis, visualization, and processing by maintaining standardized structures, clear column names, and consistent data types. Proper formatting also contributes to data quality, simplifies database integration, and is essential for compliance in regulated industries. In summary, data formatting is a foundational step that enhances the reliability, usability, and accuracy of data, laying the groundwork for meaningful insights and decisions in the field of data science.

```
[ ] # 3.2.1 Data Formatting
    # a) To change original column names with specific column names

    import pandas as pd

    # Load the dataset from the given URL
    url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
    df = pd.read_csv(url)

    # Display the original column names
    print("Original Column Names:")
    print(df.columns)

    # Rename specific columns
    column_mapping = {
        'AFP': 'Alpha-Fetoprotein',
        'AG': 'Anion Gap',
        'ALB': 'Albumin',
        'ALP': 'Alkaline Phosphatase',
        'ALT': 'Alanine Aminotransferase',
        'AST': 'Aspartate Aminotransferase',
        'BASO#': 'Basophil Cell Count',
        'BASO%': 'Basophil Cell Ratio',
        'BUN': 'Blood Urea Nitrogen',
        'Ca': 'Calcium',
        'CA125': 'Carbohydrate Antigen 125',
        'CA19-9': 'Carbohydrate Antigen 19-9',
        'CA72-4': 'Carbohydrate Antigen 72-4',
        'CEA': 'Carcinoembryonic Antigen',
```

Figure 3.2.1.1 Data Formatting: Change original column names to specific names

```
[ ]     'CL': 'Chlorine',
        'COA2CP': 'Carban Dioxide-Combining Power',
        'CREA': 'Creatinine',
        'DBIL': 'Direct Bilirubin',
        'EO#': 'Eosinophil Count',
        'EO%': 'Eosinophil Ratior6',
        'GGT': 'Gama Glutamyltransferasey',
        'GLO': 'Globulin',
        'GLU.': 'Glucose',
        'HCT': 'Hematocrit',
        'HE4': 'Human Epididymis Protein 4',
        'HGB': 'Hemoglobin',
        'IBIL': 'Indirect Bilirubin',
        'K': 'Kalium',
        'LYM#': 'Lymphocyte Count',
        'LYM%': 'Lymphocyte Ratio',
        'MCH': 'Mean Corpuscular Hemoglubin',
        'MCV': 'Mean Corpuscular Volume',
        'Mg': 'Magnesium',
        'MONO#': 'Mononuclear Cell Count',
        'MONO%': 'Mononuclear Cell Ratio',
        'MPV': 'Mean Platelet Volume',
        'Na': 'Natrium',
        'NEU': 'Neutrophil Ratio',
        'PCT': 'Thrombocytocrit',
        'PDW': 'Platelet Distribution Width',
        'PHOS': 'Phosphorus',
        'PLT': 'Platelet Count',
        'RBC': 'Red Blood Cell Count',
        'RDW': 'Red Blood Cell Distribution Width',
```

Figure 3.2.1.2 Data Formatting: Change original column names to specific names

```
    'TBIL': 'Total Bilirubin',
    'TP': 'Total Protein',
    'UA': 'Uric Acid',
}

df = df.rename(columns=column_mapping)

# Display the updated column names
print("\nUpdated Column Names:")
print(df.columns)

# Save the formatted DataFrame to a new CSV file
df.to_csv('formatted_ovarian_dataset.csv', index=False)
```

Figure 3.2.1.3 Data Formatting: Change original column names to specific names

```
Original Column Names:
Index(['AFP', 'AG', 'Age', 'ALB', 'ALP', 'ALT', 'AST', 'BASO#', 'BASO%', 'BUN',
       'Ca', 'CA125', 'CA19-9', 'CA72-4', 'CEA', 'CL', 'CO2CP', 'CREA', 'DBIL',
       'EO#', 'EO%', 'GGT', 'GLO', 'GLU.', 'HCT', 'HE4', 'HGB', 'IBIL', 'K',
       'LYM#', 'LYM%', 'MCH', 'MCV', 'Menopause', 'Mg', 'MONO#', 'MONO%',
       'MPV', 'Na', 'NEU', 'PCT', 'PDW', 'PHOS', 'PLT', 'RBC', 'RDW', 'TBIL',
       'TP', 'UA', 'TYPE'],
      dtype='object')

Updated Column Names:
Index(['Alpha-Fetoprotein', 'Anion Gap', 'Age', 'Albumin',
       'Alkaline Phosphatase', 'Alanine Aminotransferase',
       'Aspartate Aminotransferase', 'Basophil Cell Count',
       'Basophil Cell Ratio', 'Blood Urea Nitrogen', 'Calcium',
       'Carbohydrate Antigen 125', 'Carbohydrate Antigen 19-9',
       'Carbohydrate Antigen 72-4', 'Carcinoembryonic Antigen', 'Chlorine',
       'CO2CP', 'Creatinine', 'Direct Bilirubin', 'Eosinophil Count',
       'Eosinophil Ratior6', 'Gama Glutamyltransferasey', 'Globulin',
       'Glucose', 'Hematocrit', 'Human Epididymis Protein 4', 'Hemoglobin',
       'Indirect Bilirubin', 'Kalium', 'Lymphocyte Count', 'Lymphocyte Ratio',
       'Mean Corpuscular Hemoglubin', 'Mean Corpuscular Volume', 'Menopause',
       'Magnesium', 'Mononuclear Cell Count', 'Mononuclear Cell Ratio',
       'Mean Platelet Volume', 'Natrium', 'Neutrophil Ratio',
       'Thrombocytocrit', 'Platelet Distribution Width', 'Phosphorus',
       'Platelet Count', 'Red Blood Cell Count',
       'Red Blood Cell Distribution Width', 'Total Bilirubin', 'Total Protein',
       'Uric Acid', 'TYPE'],
      dtype='object')
```

Figure 3.2.1.4 Data Formatting: Change original column names to specific names output

```
# 3.2.1 Data Formatting
# a.i) To display updated column names

from IPython.display import display

# Display the entire DataFrame
display(df)
```

	Alpha-Fetoprotein	Anion Gap	Age	Albumin	Alkaline Phosphatase	Alanine Aminotransferase	Aspartate Aminotransferase	Basophil Cell Count	Basophil Cell Ratio	Blood Urea Nitrogen	...	Thrombocytocrit	Platelet Distribution Width	Phosphorus	Platelet Count	Red Blood Cell Count	Dis
0	3.58	19.36	47	45.4	56	11	24	0.01	0.30	5.35	...	0.09	13.4	1.46	74	2.64	
1	34.24	23.98	61	39.9	95	9	13	0.02	0.30	3.21	...	0.30	11.2	1.09	304	4.89	
2	1.50	18.40	39	45.4	77	9	18	0.03	0.60	3.80	...	0.13	15.2	0.97	112	4.62	
3	2.75	16.60	45	39.2	26	16	17	0.05	0.74	5.27	...	0.25	17.4	1.25	339	4.01	
4	2.36	19.97	45	35.0	47	21	27	0.01	0.10	4.89	...	0.28	11.9	0.94	272	4.40	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
344	2.09	20.91	52	48.6	65	40	30	0.00	0.00	4.29	...	0.22	10.8	0.99	219	4.56	
345	2.00	12.14	37	46.1	49	9	20	0.00	0.00	3.12	...	0.31	17.3	0.95	262	4.24	
346	2.83	19.23	59	45.8	83	9	16	0.02	0.60	4.38	...	0.21	10.4	1.00	221	4.89	
347	1.84	27.22	30	47.8	64	26	22	0.03	0.30	3.61	...	0.42	13.4	1.31	398	5.35	
348	1.61	13.00	39	44.9	68	7	13	0.01	0.20	4.38	...	0.23	11.0	1.10	224	4.13	

Figure 3.2.1.5 Display updated column names

It is then proceeded with the second data formatting in which we move the Age column to the first column of the dataset.

```
[ ] # 3.2.1 Data Formatting
    # b) To move 'Age' column to the first column of the dataset

    import pandas as pd

    # Load the dataset from the given URL
    url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
    df = pd.read_csv(url)

    # Display the original column names
    print("Original Column Names:")
    print(df.columns)

    # Move the 'Age' column to the first position
    column_order = ['Age'] + [col for col in df.columns if col != 'Age']
    df = df[column_order]

    # Display the updated column order
    print("\nUpdated Column Order:")
    print(df.columns)

    # Save the formatted DataFrame to a new CSV file
    df.to_csv('formatted_ovarian_dataset.csv', index=False)
```

Figure 3.2.1.6 Data Formatting: Move 'Age' column to the first column of the dataset

```
Original Column Names:
Index(['AFP', 'AG', 'Age', 'ALB', 'ALP', 'ALT', 'AST', 'BASO#', 'BASO%', 'BUN',
       'Ca', 'CA125', 'CA19-9', 'CA72-4', 'CEA', 'CL', 'CO2CP', 'CREA', 'DBIL',
       'EO#', 'EO%', 'GGT', 'GLO', 'GLU.', 'HCT', 'HE4', 'HGB', 'IBIL', 'K',
       'LYM#', 'LYM%', 'MCH', 'MCV', 'Menopause', 'Mg', 'MONO#', 'MONO%',
       'MPV', 'Na', 'NEU', 'PCT', 'PDW', 'PHOS', 'PLT', 'RBC', 'RDW', 'TBIL',
       'TP', 'UA', 'TYPE'],
      dtype='object')

Updated Column Order:
Index(['Age', 'AFP', 'AG', 'ALB', 'ALP', 'ALT', 'AST', 'BASO#', 'BASO%', 'BUN',
       'Ca', 'CA125', 'CA19-9', 'CA72-4', 'CEA', 'CL', 'CO2CP', 'CREA', 'DBIL',
       'EO#', 'EO%', 'GGT', 'GLO', 'GLU.', 'HCT', 'HE4', 'HGB', 'IBIL', 'K',
       'LYM#', 'LYM%', 'MCH', 'MCV', 'Menopause', 'Mg', 'MONO#', 'MONO%',
       'MPV', 'Na', 'NEU', 'PCT', 'PDW', 'PHOS', 'PLT', 'RBC', 'RDW', 'TBIL',
       'TP', 'UA', 'TYPE'],
      dtype='object')
```

Figure 3.2.1.7 Data Formatting: Move 'Age' column to the first column of the dataset output

```python
# 3.2.1 Data Formatting
# b.i) To display 'Age' column to the first column of the dataset

import pandas as pd
from IPython.display import display

# Load the dataset from the given URL
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Display the original column names
print("Original Column Names:")
print(df.columns)

# Rename specific columns
column_mapping = {
    'AFP': 'Alpha-Fetoprotein',
    'AG': 'Anion Gap',
    'ALB': 'Albumin',
    'ALP': 'Alkaline Phosphatase',
    'ALT': 'Alanine Aminotransferase',
    'AST': 'Aspartate Aminotransferase',
    'BASO#': 'Basophil Cell Count',
    'BASO%': 'Basophil Cell Ratio',
    'BUN': 'Blood Urea Nitrogen',
    'Ca': 'Calcium',
    'CA125': 'Carbohydrate Antigen 125',
    'CA19-9': 'Carbohydrate Antigen 19-9',
    'CA72-4': 'Carbohydrate Antigen 72-4',
```

Figure 3.2.1.8 Data Formatting: Move 'Age' column to the first column of the dataset

```
    'CL': 'Chlorine',
    'COA2CP': 'Carban Dioxide-Combining Power',
    'CREA': 'Creatinine',
    'DBIL': 'Direct Bilirubin',
    'EO#': 'Eosinophil Count',
    'EO%': 'Eosinophil Ratior6',
    'GGT': 'Gama Glutamyltransferasey',
    'GLO': 'Globulin',
    'GLU.': 'Glucose',
    'HCT': 'Hematocrit',
    'HE4': 'Human Epididymis Protein 4',
    'HGB': 'Hemoglobin',
    'IBIL': 'Indirect Bilirubin',
    'K': 'Kalium',
    'LYM#': 'Lymphocyte Count',
    'LYM%': 'Lymphocyte Ratio',
    'MCH': 'Mean Corpuscular Hemoglubin',
    'MCV': 'Mean Corpuscular Volume',
    'Mg': 'Magnesium',
    'MONO#': 'Mononuclear Cell Count',
    'MONO%': 'Mononuclear Cell Ratio',
    'MPV': 'Mean Platelet Volume',
    'Na': 'Natrium',
    'NEU': 'Neutrophil Ratio',
    'PCT': 'Thrombocytocrit',
    'PDW': 'Platelet Distribution Width',
    'PHOS': 'Phosphorus',
    'PLT': 'Platelet Count',
    'RBC': 'Red Blood Cell Count',
    'RDW': 'Red Blood Cell Distribution Width',
```

Figure 3.2.1.9 Data Formatting: Move 'Age' column to the first column of the dataset

```
    'TBIL': 'Total Bilirubin',
    'TP': 'Total Protein',
    'UA': 'Uric Acid',
}

df = df.rename(columns=column_mapping)

# Display the updated column names
print("\nUpdated Column Names:")
print(df.columns)

# Move the 'Age' column to the first position
column_order = ['Age'] + [col for col in df.columns if col != 'Age']
df = df[column_order]

# Display the entire DataFrame after column updates
display(df)
```

Figure 3.2.1.10 Data Formatting: Move 'Age' column to the first column of the dataset

	Age	Alpha-Fetoprotein	Anion Gap	Albumin	Alkaline Phosphatase	Alanine Aminotransferase	Aspartate Aminotransferase	Basophil Cell Count	Basophil Cell Ratio	Blood Urea Nitrogen	...	Thrombocytocrit	Platelet Distribution Width	Phosphorus	Platelet Count	Red Blood Cell Count
0	47	3.58	19.36	45.4	56	11	24	0.01	0.30	5.35	...	0.09	13.4	1.46	74	2.64
1	61	34.24	23.98	39.9	95	9	13	0.02	0.30	3.21	...	0.30	11.2	1.09	304	4.89
2	39	1.50	18.40	45.4	77	9	18	0.03	0.60	3.80	...	0.13	15.2	0.97	112	4.62
3	45	2.75	16.60	39.2	26	16	17	0.05	0.74	5.27	...	0.25	17.4	1.25	339	4.01
4	45	2.36	19.97	35.0	47	21	27	0.01	0.10	4.89	...	0.28	11.9	0.94	272	4.40
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
344	52	2.09	20.91	48.6	65	40	30	0.00	0.00	4.29	...	0.22	10.8	0.99	219	4.56
345	37	2.00	12.14	46.1	49	9	20	0.00	0.00	3.12	...	0.31	17.3	0.95	262	4.24
346	59	2.83	19.23	45.8	83	9	16	0.02	0.60	4.38	...	0.21	10.4	1.00	221	4.89
347	30	1.84	27.22	47.8	64	26	22	0.03	0.30	3.61	...	0.42	13.4	1.31	398	5.35
348	39	1.61	13.00	44.9	68	7	13	0.01	0.20	4.38	...	0.23	11.0	1.10	224	4.13

349 rows × 50 columns

Figure 3.2.1.11 Data Formatting: Move 'Age' column to the first column of the dataset output

3.2.2 Data Normalization (centering/scaling)

Data normalization, specifically centering, is crucial for statistical analysis and machine learning as it eliminates the scale differences among variables, ensuring that each feature contributes equally to the model. By centering the data, the mean of each variable becomes zero, which simplifies interpretation and aids in model convergence. This process is particularly important for algorithms sensitive to scale, such as gradient descent-based optimization methods. Normalization enhances the stability and efficiency of models, leading to improved performance and more reliable insights from the data. Overall, centering is an essential step in preprocessing data, promoting the effectiveness of various analytical techniques and enhancing the robustness of machine learning models.

```python
# 3.2.2 Data Normalization (centering)
# a) To perform centering of data normalization

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from IPython.display import display

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Get the names of numeric columns
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Centering (Z-score Normalization without scaling to unit variance)
scaler_centering = StandardScaler(with_mean=True, with_std=False)
df_centered = df.copy()
df_centered[numeric_columns] = scaler_centering.fit_transform(df_centered[numeric_columns])

# Display the centered table
display(df_centered.head())
```

Figure 3.2.2.1 Data Normalization: Centering

```python
# Function to plot histograms for the specified columns
def plot_histograms(df, columns, title):
    plt.figure(figsize=(24, 12))
    plt.suptitle(title, fontsize=16)

    for i, col in enumerate(columns, 1):
        plt.subplot(2, len(columns)//2, i)
        sns.histplot(df[col], kde=True)
        plt.title(col)

    plt.tight_layout()
    plt.show()

# After Centering
plot_histograms(df_centered, numeric_columns, 'After Centering')
```

Figure 3.2.2.2 Data Normalization: Centering

	AFP	AG	Age	ALB	ALP	ALT	AST	BASO#	BASO%	BUN	...	PCT	PDW	PHOS	PLT	RBC	RDW	TBIL
0	-7.808367	0.039484	1.948424	4.322923	-21.083095	-7.011461	4.896848	-0.018883	-0.179771	1.338883	...	-0.16145	-0.930487	0.339398	-181.432665	-1.719513	0.150086	-3.589685
1	22.851633	4.659484	15.948424	-1.177077	17.916905	-9.011461	-6.103152	-0.008883	-0.179771	-0.801117	...	0.04855	-3.130487	-0.030602	48.567335	0.530487	-0.849914	-2.289685
2	-9.888367	-0.920516	-6.051576	4.322923	-0.083095	-9.011461	-1.103152	0.001117	0.120229	-0.211117	...	-0.12145	0.869513	-0.150602	-143.432665	0.260487	-1.549914	5.710315
3	-8.638367	-2.720516	-0.051576	-1.877077	-51.083095	-2.011461	-2.103152	0.021117	0.260229	1.258883	...	-0.00145	3.069513	0.129398	83.567335	-0.349513	1.050086	1.810315
4	-9.028367	0.649484	-0.051576	-6.077077	-30.083095	2.988539	7.896848	-0.018883	-0.379771	0.878883	...	0.02855	-2.430487	-0.180602	16.567335	0.040487	-0.149914	-3.789685

5 rows × 50 columns

Figure 3.2.2.3 Data Normalization: Centering output

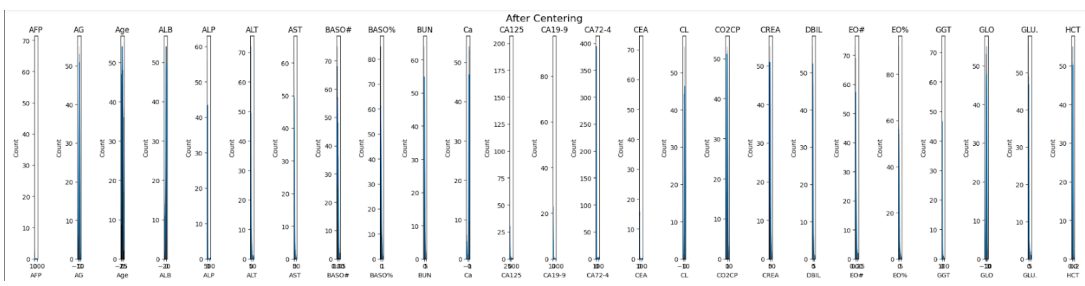


Figure 3.2.2.4 Data Normalization: After Centering output visualization

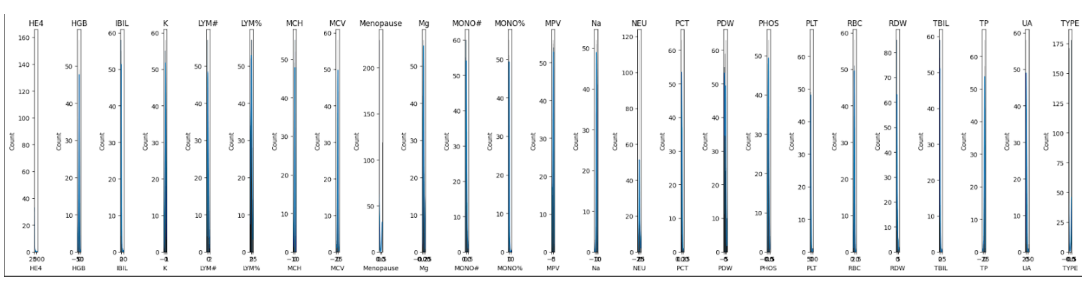


Figure 3.2.2.5 Data Normalization: After Centering output visualization

3.2.3 Binning

Binning is a process of transforming continuous numerical variables into discrete categorical 'bins', for grouped analysis. In our dataset, 'AGE', 'HGB' and 'PLT' are real valued variables with unique values.

```
# 3.2.3 Data Binning
# Age Column
import pandas as pd

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Choosing the 'Age' column for binning
column_to_bin = 'Age'

# Defining the bin edges
bin_edges = [0, 20, 40, 60, 80, 100]

# Defining bin labels
bin_labels = ['0-20', '21-40', '41-60', '61-80', '81-100']

# Create a new column with bin labels
df['Age_Bin'] = pd.cut(df[column_to_bin], bins=bin_edges, labels=bin_labels, include_lowest=True)

# Display the binned data and original data for the age column only
display(df[['Age', 'Age_Bin']])
```

Figure 3.2.3.1 Data Binning : 'AGE' Column

	Age	Age_Bin
0	47	41-60
1	61	61-80
2	39	21-40
3	45	41-60
4	45	41-60
...	...	...
344	52	41-60
345	37	21-40
346	59	41-60
347	30	21-40
348	39	21-40

349 rows × 2 columns

Figure 3.2.3.2 Data Binning: Result after Data Binning for 'AGE' Column

```python
# 3.2.3 Data Binning
# Hemoglobin and Platelet Count Columns
import pandas as pd

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Convert 'HGB' unit from g/L to g/dL
df['HGB_g_dL'] = df['HGB'] / 10

# Display the original and converted 'HGB' column
display(df[['HGB' , 'HGB_g_dL']])
```

Figure 3.2.3.3 Data Binning : Converting the 'HGB' unit from g/L to g/dL

	HGB	HGB_g_dL
0	89.0	8.9
1	128.0	12.8
2	131.0	13.1
3	123.0	12.3
4	122.0	12.2
...	...	...
344	142.0	14.2
345	80.0	8.0
346	144.0	14.4
347	154.0	15.4
348	128.0	12.8

349 rows × 2 columns

Figure 3.2.3.4 Data Binning : Result after converting the 'HGB' unit from g/L to g/dL

```python
# 3.2.3 Data Binning
# Hemoglobin and Platelet Count Columns
import pandas as pd

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Convert 'HGB' unit from g/L to g/dL
df['HGB_g_dL'] = df['HGB'] / 10

# Define bin edges for Hemoglobin (HGB) in g/dL
hgb_bin_edges = [0, 12, 16, 25]  # To consider 12-16 g/dL as 'Normal'

# Bin labels for Hemoglobin (HGB)
hgb_bin_labels = ['Low', 'Normal', 'High']

# Create a new column with bin labels for Hemoglobin (using the converted column)
df['HGB_Bin'] = pd.cut(df['HGB_g_dL'], bins=hgb_bin_edges, labels=hgb_bin_labels, include_lowest=True)

# Define bin edges and labels for Platelet Count (PLT)
plt_bin_edges = [0, 150, 450, 1000]  # You can adjust these values based on your requirements
plt_bin_labels = ['Low', 'Normal', 'High']

# Create a new column with bin labels for Platelet Count
df['PLT_Bin'] = pd.cut(df['PLT'], bins=plt_bin_edges, labels=plt_bin_labels, include_lowest=True)

# Display the binned data
display(df[['HGB_g_dL', 'HGB_Bin', 'PLT', 'PLT_Bin']])
```

Figure 3.2.3.5 Data Binning : 'HGB' & 'PLT' columns

	HGB_g_dL	HGB_Bin	PLT	PLT_Bin
0	8.9	Low	74	Low
1	12.8	Normal	304	Normal
2	13.1	Normal	112	Low
3	12.3	Normal	339	Normal
4	12.2	Normal	272	Normal
...	...	...	...	...
344	14.2	Normal	219	Normal
345	8.0	Low	262	Normal
346	14.4	Normal	221	Normal
347	15.4	Normal	398	Normal
348	12.8	Normal	224	Normal

349 rows × 4 columns

Figure 3.2.3.6 Data Binning : Result after Data Binning for 'HGB' and 'PLT' columns

3.2.4 Indicator Variables

```python
# 3.2.4 Indicator Variables

# Example assuming the columns are named 'HGB_Bin' and 'PLT_Bin'
hgb_indicator = pd.get_dummies(df['HGB_Bin'], prefix='HGB')
plt_indicator = pd.get_dummies(df['PLT_Bin'], prefix='PLT')

# Concatenate indicator variables back to the original DataFrame
df = pd.concat([df, hgb_indicator, plt_indicator], axis=1).drop(['HGB_Bin', 'PLT_Bin'], axis=1)

# Display the data with indicator variables
display(df[['HGB_g_dL', 'HGB_Low', 'HGB_Normal', 'HGB_High', 'PLT_Low', 'PLT_Normal', 'PLT_High']])
```

	HGB_g_dL	HGB_Low	HGB_Normal	HGB_High	PLT_Low	PLT_Normal	PLT_High
0	8.9	1	0	0	1	0	0
1	12.8	0	1	0	0	1	0
2	13.1	0	1	0	1	0	0
3	12.3	0	1	0	0	1	0
4	12.2	0	1	0	0	1	0
...	...	...	...	...	...	...	...
344	14.2	0	1	0	0	1	0
345	8.0	1	0	0	0	1	0
346	14.4	0	1	0	0	1	0
347	15.4	0	1	0	0	1	0
348	12.8	0	1	0	0	1	0

349 rows × 7 columns

Figure 3.2.4.1 Indicator Variables : Using 0 and 1

In the context of indicator variables, 0 and 1 indicate the absence or presence of a specific category or attribute.

- 'HGB_Low': This variable will be 1 if the observation falls into the 'Low' category in the 'HGB_Bin' column, and 0 otherwise.
- 'HGB_Normal': This variable will be 1 if the observation falls into the 'Normal' category in the 'HGB_Bin' column, and 0 otherwise.
- 'HGB_High': This variable will be 1 if the observation falls into the 'High' category in the 'HGB_Bin' column, and 0 otherwise.

The same concept applies to the ‘PLT_Bin’ column:

- 'PLT_Low', 'PLT_Normal', 'PLT_High': These variables will be 1 if the observation falls into the respective categories ('Low', 'Normal', 'High') in the 'PLT_Bin' column, and 0 otherwise.

3.3 Chapter Summary

In this chapter, we delve into the essential steps of data preparation and analysis using Python for effective data science workflows. The first section focuses on importing datasets, emphasizing the significance of acquiring clean and well-organized data for meaningful insights. We explore various methods to import data into Python, utilizing popular libraries such as Pandas to efficiently handle diverse file formats and structures.

Moving on to the analysis phase, the second section explores Python's powerful data analysis capabilities. We introduce key data science packages like NumPy, Pandas, and Matplotlib, demonstrating how these tools facilitate data exploration, manipulation, and visualization. Through real-world examples, readers gain insights into transforming raw data into actionable information, enabling them to make informed decisions.

The third section of the chapter delves into data wrangling, a critical process for refining raw datasets into a format suitable for analysis. The focus is on identifying and handling missing values, employing techniques such as imputation or deletion to ensure data completeness. Additionally, we cover data formatting, including changing column names and reordering columns to enhance dataset clarity. The chapter also addresses data normalization techniques, encompassing centering, binning, and the creation of indicator variables, showcasing how these methods contribute to more robust and reliable analyses. Overall, readers acquire a comprehensive understanding of foundational data science processes through practical examples and hands-on exercises.

CHAPTER 4

EXPLORATORY DATA ANALYSIS

4.0 Overview

This chapter discusses the descriptive statistics, basics of grouping, ANOVA and correlation.

4.1 Descriptive Statistics

Descriptive statistics generated by 'df.describe()' provide a summary of the main characteristics of the numerical columns in the dataset. Descriptive statistics are essential for understanding the distribution, central tendency, and variability of the data.

```
# 4.1 Descriptive Statistics

import pandas as pd

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

df.describe()
```

Figure 4.1.1 Descriptive Statistics : Summary of main characteristics

Here is what the key descriptive statistics mean :

- Count : The number of non-null entries for each column. This helps identify missing values.
- Mean (Average) : The average value of each numerical column. It gives an indication of the central tendency.
- Std (Standard Deviation) : A measure of the amount of variation or dispersion in a set of values. A higher standard deviation indicates greater variability.
- Min (Minimum) : The smallest value in each column.
- 25%, 50%, 75% (Quartiles) : These values represent the quartiles or percentiles of the data. The 25th percentile (Q1) is the value below which 25% of the data falls, the 50th percentile (Q2) is the median and the 75th percentile (Q3) is the value below which 75% of the data falls.
- Max (Maximum) : The largest value in each column.

	AFP	AG	Age	ALB	ALP	ALT	AST	BASO#	BASO%	BUN	Ca	CA125	CA19-9	CA72-4	CEA
count	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000
mean	11.388367	19.320516	45.051576	41.077077	77.083095	18.011461	19.103152	0.028883	0.479771	4.011117	2.390630	345.776476	45.873381	10.171060	3.309140
std	95.090229	4.342422	15.130949	5.604478	43.733473	11.080940	8.372143	0.020890	0.346230	1.285569	0.364975	777.057485	121.528096	14.056757	10.682624
min	0.610000	6.200000	15.000000	22.000000	26.000000	4.000000	7.000000	0.000000	0.000000	1.120000	0.920000	3.750000	0.600000	0.200000	0.200000
25%	1.670000	17.300000	33.000000	38.500000	60.000000	11.000000	14.000000	0.010000	0.200000	3.160000	2.320000	20.320000	8.340000	10.170000	0.850000
50%	2.420000	19.820000	45.000000	41.900000	71.000000	15.000000	18.000000	0.030000	0.400000	3.830000	2.470000	49.280000	15.110000	10.170000	1.400000
75%	3.770000	22.160000	57.000000	45.100000	86.000000	20.000000	22.000000	0.040000	0.700000	4.780000	2.580000	255.820000	34.220000	10.170000	2.300000
max	1210.000000	33.330000	83.000000	51.500000	763.000000	86.000000	78.000000	0.120000	1.940000	10.190000	2.830000	5000.000000	1000.000000	158.500000	138.800000

Figure 4.1.2 Descriptive Statistics : Summary of main characteristics Results

CL	CO2CP	CREA	DBIL	EO#	EO%	GGT	GLO	GLU.	HCT	HE4	HGB	IBIL	K	LYM#
349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000
100.812894	24.277880	64.250143	3.128653	0.066017	1.119112	21.303725	30.176791	5.334986	0.383874	183.948453	125.336103	5.961032	4.386447	1.557593
2.985526	2.683421	11.714643	1.414186	0.067704	1.134829	17.907152	4.479258	1.215904	0.042177	369.280717	15.534192	2.910527	0.396798	0.561449
84.600000	16.200000	38.200000	0.900000	0.000000	0.000000	4.000000	14.100000	3.570000	0.224000	16.710000	61.800000	1.000000	3.080000	0.350000
99.200000	22.400000	56.000000	2.200000	0.020000	0.300000	12.000000	27.100000	4.660000	0.362000	42.580000	119.000000	4.000000	4.150000	1.190000
100.900000	24.100000	63.300000	2.900000	0.050000	0.800000	16.000000	30.180000	5.080000	0.388000	54.940000	127.000000	5.500000	4.370000	1.500000
102.600000	26.000000	71.000000	3.700000	0.090000	1.500000	23.000000	32.800000	5.600000	0.410000	166.200000	135.000000	7.200000	4.650000	1.870000
109.400000	34.300000	114.000000	12.100000	0.400000	7.600000	176.000000	47.600000	12.440000	0.569000	3537.600000	189.000000	28.400000	5.400000	3.490000

Figure 4.1.3 Descriptive Statistics : Summary of main characteristics Results

LYM%	MCH	MCV	Menopause	Mg	MONO#	MONO%	MPV	Na	NEU	PCT	PDW	PHOS	PLT	RBC	RDW
349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000	349.000000
26.070831	28.777192	88.070201	0.340974	0.982693	0.355158	5.579083	10.039484	140.494842	66.576705	0.251450	14.330487	1.120602	255.432665	4.359513	13.549914
10.386006	2.569315	6.041754	0.474717	0.123643	0.150504	1.924260	1.733804	2.860347	9.979656	0.085062	2.991921	0.187649	95.241974	0.465894	1.802818
3.900000	17.700000	61.000000	0.000000	0.650000	0.070000	0.300000	5.060000	125.100000	37.200000	0.070000	8.800000	0.570000	74.000000	2.620000	10.920000
18.400000	28.000000	85.600000	0.000000	0.900000	0.250000	4.400000	9.100000	138.900000	62.300000	0.192000	11.900000	1.000000	202.000000	4.110000	12.400000
26.600000	29.300000	89.000000	0.000000	0.970000	0.320000	5.430000	10.300000	140.500000	66.580000	0.240000	13.800000	1.120000	236.000000	4.370000	13.100000
32.800000	30.400000	91.500000	1.000000	1.050000	0.430000	6.550000	11.200000	142.400000	71.300000	0.300000	16.800000	1.240000	292.000000	4.660000	14.400000
51.600000	36.800000	107.900000	1.000000	1.370000	0.970000	21.300000	14.500000	150.700000	92.000000	0.690000	22.800000	1.750000	868.000000	6.740000	22.200000

Figure 4.1.4 Descriptive Statistics : Summary of main characteristics Results

TBIL	TP	UA	TYPE
349.000000	349.000000	349.000000	349.000000
9.089685	71.078797	243.706590	0.510029
4.104614	7.258465	68.581936	0.500617
2.500000	32.900000	96.000000	0.000000
6.300000	67.500000	200.100000	0.000000
8.600000	72.200000	235.400000	1.000000
10.700000	75.800000	276.600000	1.000000
38.300000	86.800000	632.000000	1.000000

Figure 4.1.5 Descriptive Statistics : Results after performing 'df.describe()'

4.2 Basic of Grouping

The basics of grouping usually means the process of splitting data into groups based on a certain criteria and then performing some analysis or computation for each group separately. This code removes the first column 'AFP' followed by the second column 'AG' and prints the remaining column names. Then, it creates two groups, 'group1' and 'group2' based on a condition where the 'AG' column is equal to 'Benign Ovarian Cancer' for 'group1' and not equal for 'group2'. Finally, it prints the contents of the two groups.

```python
# 4.2 Basic of Grouping

import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

cols=list(df)
cols.pop(0)
```

```
'AFP'
```

```python
cols.pop(0)
```

```
'AG'
```

```python
print(cols)
```

```
['Age', 'ALB', 'ALP', 'ALT', 'AST', 'BASO#', 'BASO%', 'BUN', 'Ca', 'CA125', 'CA19-9', 'CA72-4', 'CEA', 'CL', 'CO2CP', 'CREA', 'DBIL', 'EO#'
```

Figure 4.2.1 Basics of Grouping : Method

```python
cond=df['AG']=='Benign Ovarian Cancer'
```

```python
group1=df.loc[cond, cols]
group2=df.loc[~cond, cols]
```

```python
print(group1)
```

```
Empty DataFrame
Columns: [Age, ALB, ALP, ALT, AST, BASO#, BASO%, BUN, Ca, CA125, CA19-9, CA72-4, CEA, CL, CO2CP, CREA, DBIL, EO#, EO%, GGT, GLO,
Index: []

[0 rows x 48 columns]
```

```python
print(group2)
```

```
     Age   ALB  ALP  ALT  AST  BASO#  BASO%   BUN    Ca    CA125  ...   PCT  \
0     47  45.4   56   11   24   0.01   0.30  5.35  2.48    15.36  ...  0.09
1     61  39.9   95    9   13   0.02   0.30  3.21  2.62  2444.00  ...  0.30
2     39  45.4   77    9   18   0.03   0.60  3.80  2.57    56.08  ...  0.13
3     45  39.2   26   16   17   0.05   0.74  5.27  2.35  2555.00  ...  0.25
4     45  35.0   47   21   27   0.01   0.10  4.89  2.48  1391.00  ...  0.28
..    ...   ...  ...  ...  ...    ...    ...   ...   ...      ...  ...   ...
344   52  48.6   65   40   30   0.00   0.00  4.29  2.81    50.80  ...  0.22
345   37  46.1   49    9   20   0.00   0.00  3.12  2.29    21.93  ...  0.31
346   59  45.8   83    9   16   0.02   0.60  4.38  2.65    51.33  ...  0.21
347   30  47.8   64   26   22   0.03   0.30  3.61  2.82    53.87  ...  0.42
348   39  44.9   68    7   13   0.01   0.20  4.38  2.58    46.45  ...  0.23

      PDW  PHOS  PLT   RBC   RDW  TBIL    TP     UA  TYPE
0    13.4  1.46   74  2.64  13.7   5.5  73.9  396.4     0
1    11.2  1.09  304  4.89  12.7   6.8  72.0  119.2     0
2    15.2  0.97  112  4.62  12.0  14.8  77.9  209.2     0
3    17.4  1.25  339  4.01  14.6  10.9  66.1  215.6     0
4    11.9  0.94  272  4.40  13.4   5.3  66.5  206.0     0
..    ...   ...  ...   ...   ...   ...   ...    ...   ...
344  10.8  0.99  219  4.56  11.9  10.7  80.4  241.5     1
345  17.3  0.95  262  4.24  17.7  17.2  73.5  240.5     1
346  10.4  1.00  221  4.89  11.6  13.8  68.4  209.5     1
347  13.4  1.31  398  5.35  12.5  11.3  76.0  285.7     1
348  11.0  1.10  224  4.13  12.1   8.3  68.6  184.9     1

[349 rows x 48 columns]
```

Figure 4.2.2 Basics of Grouping : Method and Results

4.3 ANOVA

ANOVA or Analysis of Variance is a statistical method used to compare means among three or more groups to determine if there are statistically significant differences. In the context of this code, ANOVA is applied to investigate whether there are significant differences in the mean values of a variable specified by 'variable_to_analyze' which is actually the 'PLT' (Platelet Count) across different age groups.

```
# 4.3 ANOVA
import pandas as pd
from scipy.stats import f_oneway

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Choosing the 'Age' column for binning
column_to_bin = 'Age'

# Defining the bin edges
bin_edges = [0, 20, 40, 60, 80, 100]

# Defining bin labels
bin_labels = ['0-20', '21-40', '41-60', '61-80', '81-100']

# Create a new column with bin labels
df['Age_Bin'] = pd.cut(df[column_to_bin], bins=bin_edges, labels=bin_labels, include_lowest=True)

# Specifying the variable to analyze (PLT - Platelet Count)
variable_to_analyze = 'PLT'

# Grouping the data by the 'Age_Bin' and calculating the mean for each group
grouped_data = df.groupby('Age_Bin')[variable_to_analyze].mean()

# Display the mean values for each group
print("Mean values for each group:")
print(grouped_data)

# Performing ANOVA on Platelet Count (PLT)
anova_result = f_oneway(df[df['Age_Bin'] == '0-20'][variable_to_analyze],
                        df[df['Age_Bin'] == '21-40'][variable_to_analyze],
                        df[df['Age_Bin'] == '41-60'][variable_to_analyze],
                        df[df['Age_Bin'] == '61-80'][variable_to_analyze],
                        df[df['Age_Bin'] == '81-100'][variable_to_analyze])

# Display ANOVA results for Platelet Count (PLT)
print("\nANOVA Results for", variable_to_analyze, ":")
print(anova_result)
```

Figure 4.3.1 ANOVA : ANOVA on the Platelet Count variable across different age groups

```
Mean values for each group:
Age_Bin
0-20      240.700000
21-40     243.992424
41-60     254.269504
61-80     282.938462
81-100    289.000000
Name: PLT, dtype: float64

ANOVA Results for PLT :
F_onewayResult(statistic=1.9486112038615206, pvalue=0.10199340093136981)
```

Figure 4.3.2 ANOVA : Results

The p-value is greater than the commonly used significance level of 0.05. Therefore, based on this ANOVA test, there is no statistically significant difference in mean platelet counts among the different age groups. In other words, the variations observed in mean platelet count could be due to random chance rather than a systematic effect related to age. Researchers typically use the p-value to make decisions about rejecting or failing to reject the null hypothesis. A p-value above the significance level suggests that there is not enough evidence to reject the null hypothesis, which in this context, would mean that age does not have a significant impact on platelet counts.

4.4 Correlation

Correlation measures the strength and direction of a linear relationship between two numerical variables. In the context of this dataset, we decided to find the correlation between 'PLT' (Platelet Count) and 'AGE'.

This code calculates the Pearson correlation coefficient that ranges from -1 to, where :

- 1 indicates a perfect positive linear relationship
- -1 indicates a perfect negative linear relationship
- 0 indicates no linear relationship

```
# 4.4 Correlation
# Correlation between 'PLT' and 'AGE'

import pandas as pd

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Select the variables you want to calculate correlation for
variable1 = 'PLT'
variable2 = 'Age'

# Calculate the correlation coefficient
correlation_coefficient = df[variable1].corr(df[variable2])

# Display the correlation coefficient
print(f'Correlation between {variable1} and {variable2}: {correlation_coefficient}')
```

Figure 4.4.1 Correlation : Correlation between 'PLT' and 'AGE'

```
Correlation between PLT and Age: 0.13964058283821001
```

Figure 4.4.2 Correlation : Results

Since the correlation coefficient is positive and relatively close to 0, it suggests a weak positive linear relationship between 'PLT' and 'AGE' in your dataset. However, the strength of this relationship is considered weak, meaning that changes in 'AGE' are not strongly predictive of changes in 'PLT', and vice versa.

4.5 Chapter Summary

CHAPTER 5

MODEL DEVELOPMENT

5.0 Overview

This chapter discusses simple and multiple linear regression, model evaluation using visualization, polynomial regression and pipelines, R-Squared and MSE for In-Sample Evaluation and prediction and decision making.

5.1 Simple and Multiple Linear Regression

The code demonstrates simple linear regression using the 'Age' column as the independent variable (X_simple) and the 'CA125' column as the dependent variable (y_simple). The model is then fitted, and the summary is printed, providing information about the coefficients, statistical significance, and goodness of fit. The code also shows an example of multiple linear regression. In this case, the independent variables (X_multiple) include 'Age', 'ALB' (Albumin), and 'ALP' (Alkaline Phosphatase), while the dependent variable is still 'CA125'. The model is fitted, and the summary provides information about the coefficients for each independent variable, statistical significance, and other relevant statistics.

```python
# 5.1 Simple and Multiple Linear Regression

import pandas as pd
import statsmodels.api as sm

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Print the column names
print("Column Names:", df.columns)

# Simple Linear Regression
# Assuming 'Age' is the independent variable and 'CA125' is the dependent variable
X_simple = df['Age']
y_simple = df['CA125']

# Add a constant term to the independent variable
X_simple = sm.add_constant(X_simple)

# Fit the model
model_simple = sm.OLS(y_simple, X_simple).fit()

# Print the summary
print(model_simple.summary())

# Multiple Linear Regression
# Assuming 'Age', 'ALB', 'ALP', ... are the independent variables and 'CA125' is the dependent variable
X_multiple = df[['Age', 'ALB', 'ALP']]
y_multiple = df['CA125']

# Add a constant term to the independent variables
X_multiple = sm.add_constant(X_multiple)

# Fit the model
model_multiple = sm.OLS(y_multiple, X_multiple).fit()

# Print the summary
print(model_multiple.summary())
```

Figure 5.1.1 Simple and Multiple Linear Regression : Using 'CA125' as the dependent variable

```
No. Observations:                 349   AIC:                             5598.
Df Residuals:                     347   BIC:                             5606.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -420.4957    123.608     -3.402      0.001    -663.612    -177.380
Age           17.0088      2.601      6.539      0.000      11.892      22.125
==============================================================================
Omnibus:                      263.927   Durbin-Watson:                   1.746
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2910.512
Skew:                           3.221   Prob(JB):                         0.00
Kurtosis:                      15.596   Cond. No.                         149.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                  CA125   R-squared:                       0.160
Model:                            OLS   Adj. R-squared:                  0.153
Method:                 Least Squares   F-statistic:                     21.91
Date:                Sun, 07 Jan 2024   Prob (F-statistic):           5.19e-13
Time:                        09:25:38   Log-Likelihood:                -2787.1
No. Observations:                 349   AIC:                             5582.
Df Residuals:                     345   BIC:                             5598.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        623.2689    348.569      1.788      0.075     -62.319    1308.857
Age           13.2723      2.665      4.981      0.000       8.031      18.514
ALB          -25.8306      7.098     -3.639      0.000     -39.792     -11.869
ALP            2.4080      0.890      2.706      0.007       0.658       4.158
==============================================================================
Omnibus:                      271.822   Durbin-Watson:                   1.876
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3317.891
Skew:                           3.310   Prob(JB):                         0.00
Kurtosis:                      16.577   Cond. No.                         956.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 5.1.2 Simple and Multiple Linear Regression : Results

1. Model with ‘AGE’ as the independent variable:

    - R-squared: 0.110, indicating that approximately 11.0% of the variance in 'CA125' is explained by 'AGE'.
    - The coefficient for 'Age' is 17.0088, suggesting that for each unit increase in 'Age', 'CA125' is expected to increase by 17.0088 units.

2. Model with ‘AGE’ 'ALB', and 'ALP' as independent variables:

- R-squared: 0.160, indicating that approximately 16.0% of the variance in 'CA125' is explained by 'AGE', 'ALB', and 'ALP'
- The coefficients provide the expected change in 'CA125' for each unit increase in the respective independent variable, holding other variables constant. For example, the coefficient for 'Age' is 13.2723, suggesting that for each unit increase in 'AGE', 'CA125' is expected to increase by 13.2723 units, assuming 'ALB' and 'ALP' are constant.

The significance of each coefficient is assessed using the p-value. A p-value less than a chosen significance level (commonly 0.05) indicates that the variable is statistically significant.

The code below performs simple linear regression on the 'Age' variable against the 'CA125' variable using the OLS (Ordinary Least Squares) method from the statsmodels library. This code visually represents the relationship between 'Age' and 'CA125' and how well a simple linear regression line fits the data. If the line fits well, it suggests a linear relationship between the two variables. The scatter plot helps to evaluate the goodness of fit and understand the trend in the data.

```python
# 5.1 Simple and Multiple Linear Regression
# Visual Representation of the relatiuonship between 'AGE' and 'CA125' using Simple Linear Regression
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

# GitHub raw URL for the CSV file
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(url)

# Print the column names
print("Column Names:", df.columns)

# Simple Linear Regression
X_simple = df['Age']
y_simple = df['CA125']

# Add a constant term to the independent variable
X_simple = sm.add_constant(X_simple)

# Fit the model
model_simple = sm.OLS(y_simple, X_simple).fit()

# Get the predicted values
predictions = model_simple.predict(X_simple)

# Plotting the data points
plt.scatter(X_simple['Age'], y_simple, label='Actual data points')

# Plotting the regression line
plt.plot(X_simple['Age'], predictions, label='Regression line', color='red')

# Adding labels and title
plt.xlabel('Age')
plt.ylabel('CA125')
plt.title('Simple Linear Regression')

# Adding legend
plt.legend()

# Show the plot
plt.show()
```

Figure 5.1.3 Simple and Multiple Linear Regression : Visual Representation for Simple Linear Regression

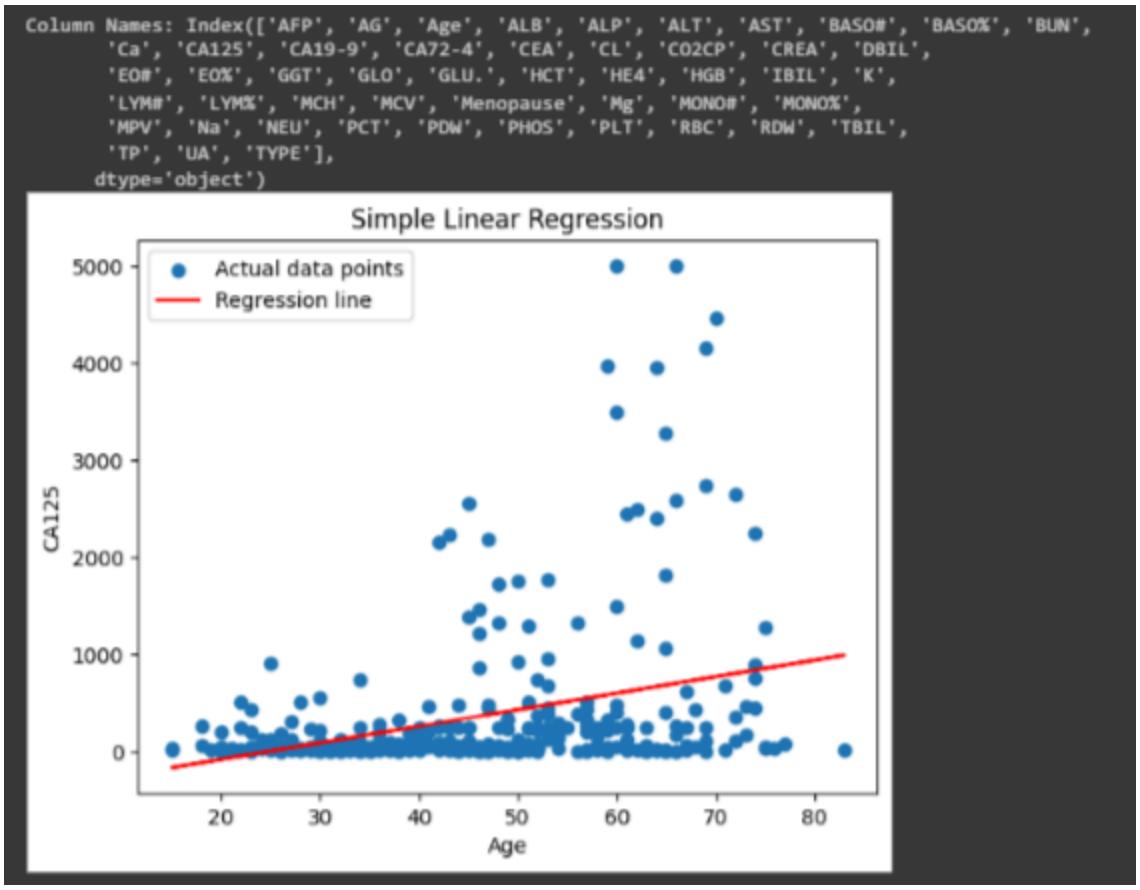


Figure 5.1.4 Simple and Multiple Linear Regression : Visual Representation Results

The code below performs multiple linear regression on the variables 'Age' and 'ALB' against the 'CA125' variable and visualizes the results in a 3D scatter plot. This code helps visualize the relationship between 'Age', 'ALB', and 'CA125' in a three-dimensional space. The scatter plot includes both the actual data points and the predicted values, allowing you to visually assess how well the multiple linear regression model captures the relationship between these variables.

```python
# 5.1 Simple and Multiple Linear Regression
# Visual Representation of the relatiuonship between 'AGE' , 'ALB' and 'CA125' using Multiple Linear Regression

from mpl_toolkits.mplot3d import Axes3D

# Assuming 'Age', 'ALB', 'CA125' are the variables
X_multiple = df[['Age', 'ALB']]
y_multiple = df['CA125']

# Add a constant term to the independent variables
X_multiple = sm.add_constant(X_multiple)

# Fit the model
model_multiple = sm.OLS(y_multiple, X_multiple).fit()

# Get the predicted values
predictions_multiple = model_multiple.predict(X_multiple)

# Plotting the 3D scatter plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X_multiple['Age'], X_multiple['ALB'], y_multiple, label='Actual data points')
ax.scatter(X_multiple['Age'], X_multiple['ALB'], predictions_multiple, label='Predicted values', color='red')

# Adding labels and title
ax.set_xlabel('Age')
ax.set_ylabel('ALB')
ax.set_zlabel('CA125')
ax.set_title('Multiple Linear Regression')

# Adding legend
ax.legend()

# Show the plot
plt.show()
```

Figure 5.1.5 Simple and Multiple Linear Regression : Visual Representation for Multiple Linear Regression

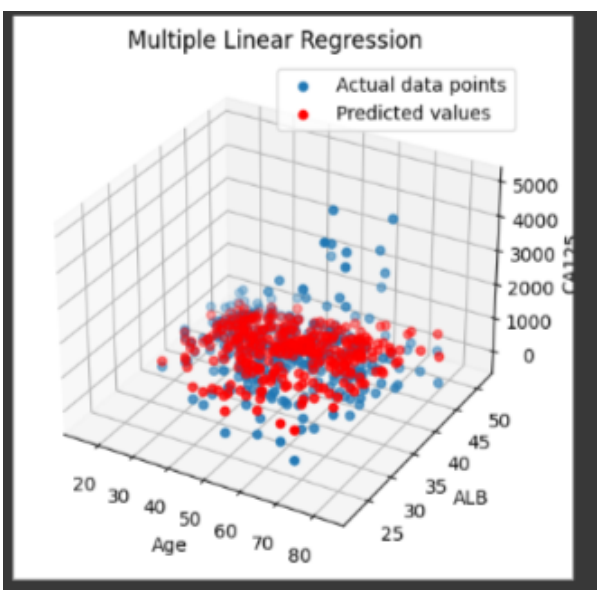


Figure 5.1.6 Simple and Multiple Linear Regression : Visual Representation Results

5.2 Model Evaluation Using Visualization

Model evaluation using visualization in the context of this project refers to the process of assessing the performance of a machine learning model through visual representations of key metrics and characteristics. The primary focus is on visually interpreting the model's effectiveness in making predictions. The main visualizations used for model evaluation here include confusion matrix, classification report, receiver operating characteristic (ROC) curve and ROC Curve visualization.

The code below uses TensorFlow and Keras to create and train a simple neural network for binary classification. It assumes a binary classification task without specifying the nature of the target variable. It treats the last column as the target variable and trains the neural network model accordingly.

```python
# 5.2 Model Evaluation Using Visualization
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import matplotlib.pyplot as plt

# Assume you have a CSV file with features and labels
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume the last column is the target variable (label)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split the dataset 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)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a simple neural network model using TensorFlow and Keras
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Make predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Figure 5.2.1 Model Evaluation Using Visualization : No specific nature of target variable

```python
# Classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Figure 5.2.2 Model Evaluation Using Visualization : No specific nature of target variable

```
Epoch 1/10
9/9 [==============================] - 1s 31ms/step - loss: 0.7890 - accuracy: 0.5412 - val_loss: 0.6375 - val_accuracy: 0.6714
Epoch 2/10
9/9 [==============================] - 0s 8ms/step - loss: 0.5556 - accuracy: 0.7778 - val_loss: 0.5559 - val_accuracy: 0.7429
Epoch 3/10
9/9 [==============================] - 0s 6ms/step - loss: 0.4542 - accuracy: 0.8315 - val_loss: 0.4901 - val_accuracy: 0.8286
Epoch 4/10
9/9 [==============================] - 0s 6ms/step - loss: 0.3871 - accuracy: 0.8423 - val_loss: 0.4563 - val_accuracy: 0.8571
Epoch 5/10
9/9 [==============================] - 0s 6ms/step - loss: 0.3448 - accuracy: 0.8638 - val_loss: 0.4428 - val_accuracy: 0.8571
Epoch 6/10
9/9 [==============================] - 0s 6ms/step - loss: 0.3072 - accuracy: 0.8853 - val_loss: 0.4391 - val_accuracy: 0.8571
Epoch 7/10
9/9 [==============================] - 0s 8ms/step - loss: 0.2790 - accuracy: 0.8961 - val_loss: 0.4383 - val_accuracy: 0.8571
Epoch 8/10
9/9 [==============================] - 0s 6ms/step - loss: 0.2538 - accuracy: 0.9283 - val_loss: 0.4405 - val_accuracy: 0.8429
Epoch 9/10
9/9 [==============================] - 0s 6ms/step - loss: 0.2303 - accuracy: 0.9355 - val_loss: 0.4386 - val_accuracy: 0.8571
Epoch 10/10
9/9 [==============================] - 0s 7ms/step - loss: 0.2086 - accuracy: 0.9462 - val_loss: 0.4403 - val_accuracy: 0.8429
3/3 [==============================] - 0s 4ms/step
Confusion Matrix:
[[30  6]
 [ 5 29]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.83      0.85        36
           1       0.83      0.85      0.84        34

    accuracy                           0.84        70
   macro avg       0.84      0.84      0.84        70
weighted avg       0.84      0.84      0.84        70
```

Figure 5.2.3 Model Evaluation Using Visualization : No specific nature of target variable

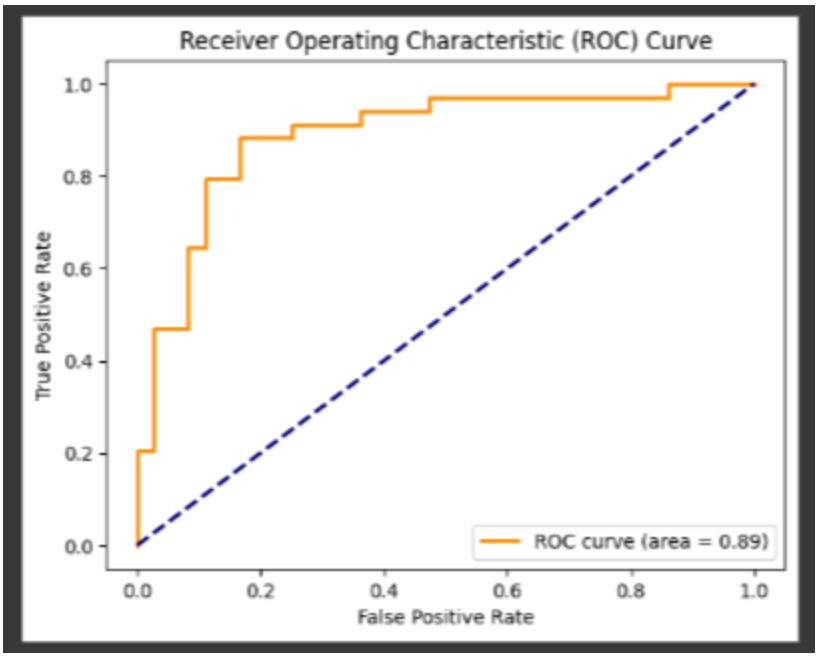


Figure 5.2.4 Model Evaluation Using Visualization : Results

The code below explicitly states that the target variable is binary, where 0 represents benign and 1 represents malignant. This additional information allows for a more specific interpretation of the model's predictions.

```
# 5.2 Model Evaluation Using Visualization

import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import matplotlib.pyplot as plt

# Assume you have a CSV file with features and labels
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume the last column is the target variable (label: 0 for benign, 1 for malignant)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split the dataset 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)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Make predictions on the test set
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Figure 5.2.5 Model Evaluation Using Visualization : Binary target variable

```python
# Classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:")
print(class_report)

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Figure 5.2.6 Model Evaluation Using Visualization : Binary target variable

```
Epoch 1/10
9/9 [==============================] - 2s 44ms/step - loss: 0.6455 - accuracy: 0.6631 - val_loss: 0.5850 - val_accuracy: 0.7000
Epoch 2/10
9/9 [==============================] - 0s 11ms/step - loss: 0.4840 - accuracy: 0.7993 - val_loss: 0.4961 - val_accuracy: 0.7714
Epoch 3/10
9/9 [==============================] - 0s 13ms/step - loss: 0.3999 - accuracy: 0.8459 - val_loss: 0.4582 - val_accuracy: 0.8429
Epoch 4/10
9/9 [==============================] - 0s 12ms/step - loss: 0.3526 - accuracy: 0.8710 - val_loss: 0.4410 - val_accuracy: 0.8429
Epoch 5/10
9/9 [==============================] - 0s 15ms/step - loss: 0.3141 - accuracy: 0.8853 - val_loss: 0.4372 - val_accuracy: 0.8429
Epoch 6/10
9/9 [==============================] - 0s 12ms/step - loss: 0.2832 - accuracy: 0.8961 - val_loss: 0.4344 - val_accuracy: 0.8571
Epoch 7/10
9/9 [==============================] - 0s 11ms/step - loss: 0.2586 - accuracy: 0.9032 - val_loss: 0.4369 - val_accuracy: 0.8429
Epoch 8/10
9/9 [==============================] - 0s 16ms/step - loss: 0.2361 - accuracy: 0.9176 - val_loss: 0.4382 - val_accuracy: 0.8571
Epoch 9/10
9/9 [==============================] - 0s 12ms/step - loss: 0.2131 - accuracy: 0.9211 - val_loss: 0.4349 - val_accuracy: 0.8429
Epoch 10/10
9/9 [==============================] - 0s 12ms/step - loss: 0.1922 - accuracy: 0.9391 - val_loss: 0.4377 - val_accuracy: 0.8571
3/3 [==============================] - 0s 4ms/step
Confusion Matrix:
[[30  6]
 [ 4 30]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.83      0.86        36
           1       0.83      0.88      0.86        34

    accuracy                           0.86        70
   macro avg       0.86      0.86      0.86        70
weighted avg       0.86      0.86      0.86        70
```

Figure 5.2.7 Model Evaluation Using Visualization : Binary target variable

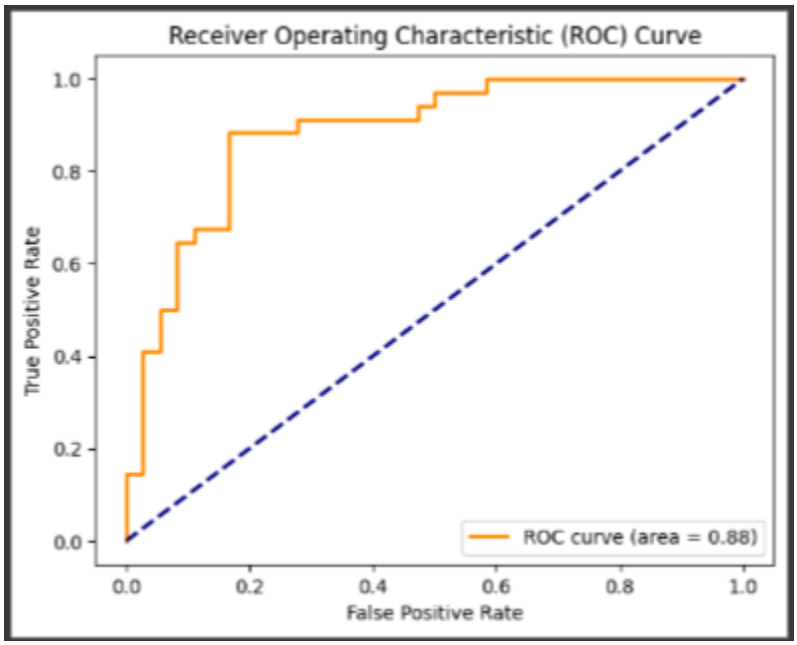


Figure 5.2.8 Model Evaluation Using Visualization : Results

5.3 Polynomial Regression and Pipelines

In the context of these codes, polynomial regression is a technique used to model the relationship between the independent variable 'Age' and the dependent variable 'CA125' in ovarian cancer data. Polynomial regression introduces non-linearity by fitting a polynomial equation of a specified degree to the data. The degree of the polynomial is a parameter that can be adjusted to control the complexity of the model. Pipelines in machine learning are a way to streamline and organize the machine learning workflow. They automate a sequence of data processing steps, making it easier to manage and reproduce the entire process. Each step in the pipeline is a tuple consisting of a name and an instance of a transformer or an estimator.

The code below is a basic polynomial regression without using pipelines. It manually applies polynomial features transformation using PolynomialFeatures and then fits a linear regression model on the transformed features.

```
# 5.3 Polynomial Regression and Pipelines
# Polynomial Regression Without Using Pipelines

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Print the column names
print("Column Names:", df.columns)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Polynomial features transformation
degree = 2  # You can adjust the degree of the polynomial
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Linear regression on the polynomial features
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Predictions on the test set
y_pred = model.predict(X_test_poly)

# Plot the results
plt.scatter(X_test, y_test, label='Actual data')
plt.plot(X_test, y_pred, label='Polynomial regression', color='red')
plt.xlabel('Age')
plt.ylabel('CA125')
plt.title('Polynomial Regression')
plt.legend()
plt.show()
```

Figure 5.3.1 Polynomial Regression and Pipelines : Polynomial Regression Without Pipelines

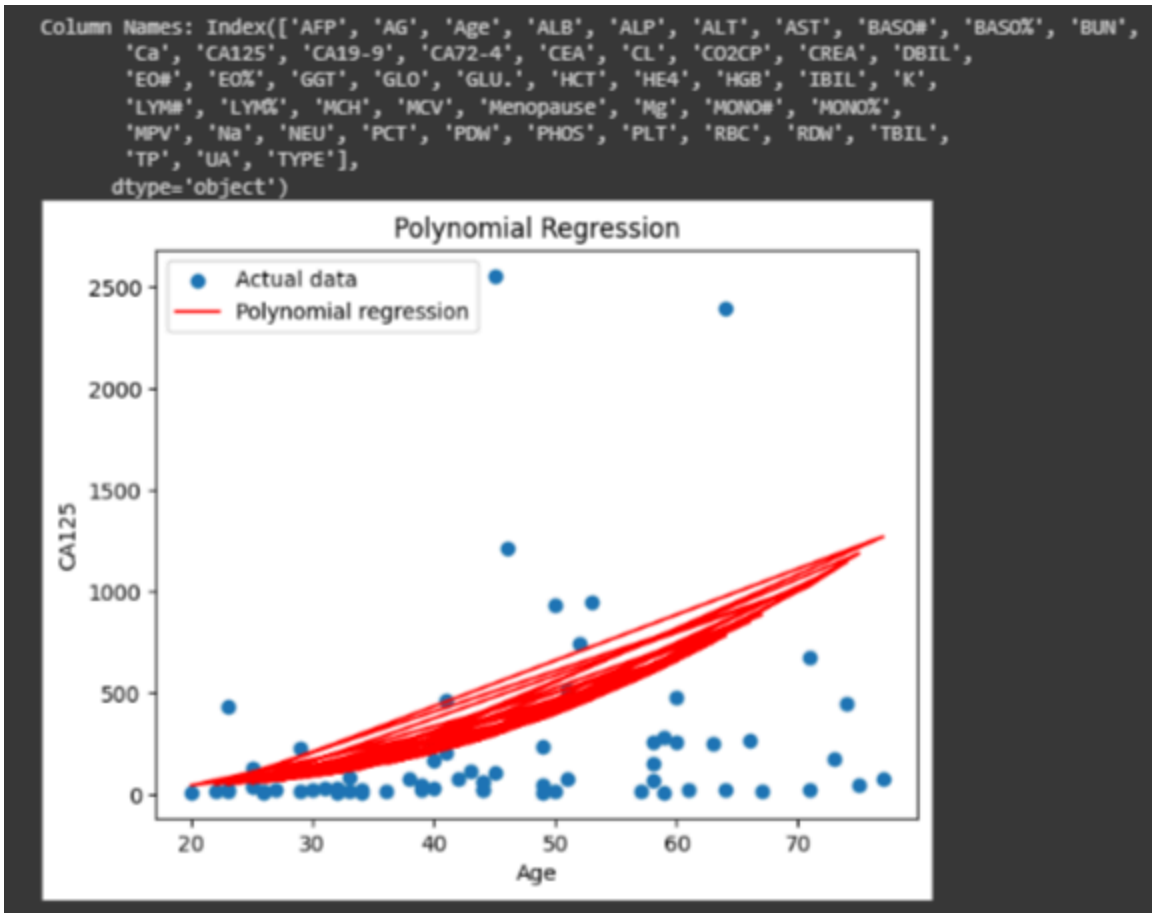


Figure 5.3.2 Polynomial Regression and Pipelines : Results Of Polynomial Regression Without Pipelines

The code below introduces the concept of pipelines in scikit-learn. It uses the Pipeline class to create a sequence of data processing steps, including polynomial feature transformation and linear regression. This makes the code more modular and readable.

```
# 5.3 Polynomial Regression and Pipelines
# Polynomial Regression With Pipeline

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Create a polynomial regression pipeline
degree = 2  # You can adjust the degree of the polynomial
polynomial_regression = Pipeline([
    ('poly_features', PolynomialFeatures(degree=degree)),
    ('linear_regression', LinearRegression())
])

# Fit the pipeline on the training data
polynomial_regression.fit(X_train, y_train)

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

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Plot the results
plt.scatter(X_test, y_test, label='Actual data')
plt.plot(X_test, y_pred, label='Polynomial regression', color='red')
plt.xlabel('Age')
plt.ylabel('CA125')
plt.title('Polynomial Regression with Pipeline')
plt.legend()
plt.show()
```

Figure 5.3.3 Polynomial Regression and Pipelines : Polynomial Regression With Pipelines

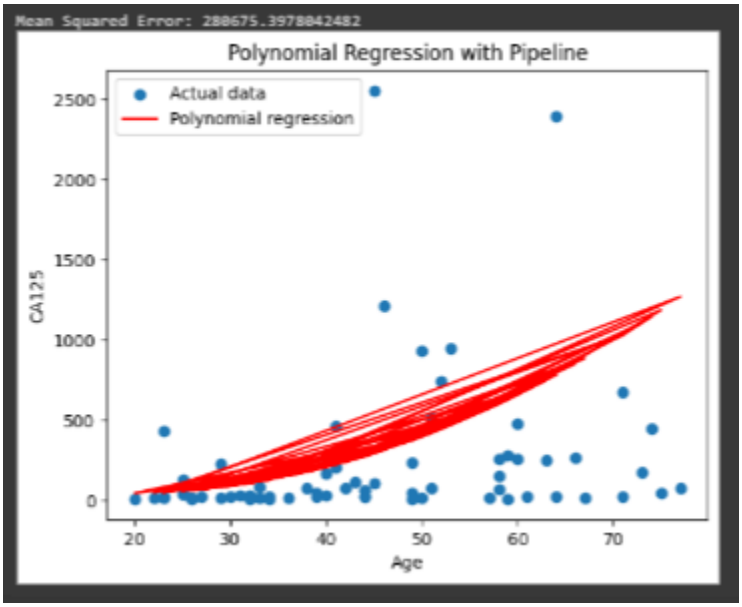


Figure 5.3.4 Polynomial Regression and Pipelines : Results Of Polynomial Regression With Pipelines

5.4 R-Squared and MSE For In-Sample Evaluation

R-squared (in-sample): It is a measure of how well the model explains the variance in the training data. R-squared values range from 0 to 1, where 1 indicates a perfect fit. In this context, it tells you the proportion of the variance in the CA125 values that is captured by the polynomial regression model.

Mean Squared Error (MSE) (in-sample): It measures the average squared difference between the predicted and actual CA125 values in the training set. Lower MSE values indicate better model performance. It provides a sense of how well the model fits the training data.

These metrics are commonly used to assess the goodness of fit and performance of a regression model on the data it was trained on. In this case, they give you insights into how well the polynomial regression model, with a degree of 2, fits the training data for the 'Age' and 'CA125' variables.

```python
# 5.4 R-Squared & MSE for In-Sample Evaluation

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Create a polynomial regression pipeline
degree = 2  # You can adjust the degree of the polynomial
polynomial_regression = Pipeline([
    ('poly_features', PolynomialFeatures(degree=degree)),
    ('linear_regression', LinearRegression())
])

# Fit the pipeline on the training data
polynomial_regression.fit(X_train, y_train)

# Make predictions on the training set
y_train_pred = polynomial_regression.predict(X_train)

# Calculate R-squared and MSE for in-sample evaluation
r2_train = r2_score(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)

print("R-squared (in-sample):", r2_train)
print("Mean Squared Error (in-sample):", mse_train)

# Plot the results for the training set
plt.scatter(X_train, y_train, label='Actual data (Training)')
plt.plot(X_train, y_train_pred, label='Polynomial regression', color='red')
plt.xlabel('Age')
plt.ylabel('CA125')
plt.title('Polynomial Regression with Pipeline (Training)')
plt.legend()
plt.show()
```

Figure 5.4.1 R-Squared and MSE for In-Sample Training

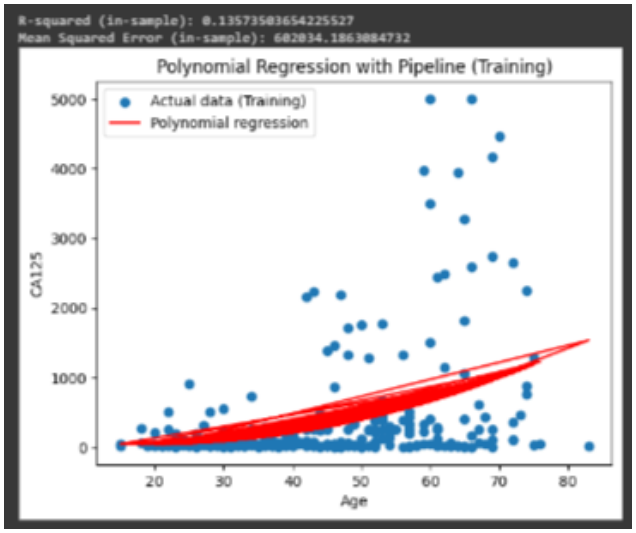


Figure 5.4.2 R-Squared and MSE for In-Sample Training : Result

5.5 Prediction and Decision Making

The code below simulates using a polynomial regression model to predict CA125 values and then makes decisions based on whether the predicted value exceeds a specified threshold.

```python
# 5.5 Prediction & Decision-Making

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Create a polynomial regression pipeline
degree = 2  # You can adjust the degree of the polynomial
polynomial_regression = Pipeline([
    ('poly_features', PolynomialFeatures(degree=degree)),
    ('linear_regression', LinearRegression())
])

# Fit the pipeline on the training data
polynomial_regression.fit(X_train, y_train)

# Make predictions on new data
new_data = np.array([30]).reshape(1, -1)  # Replace 30 with the new value you want to predict
prediction = polynomial_regression.predict(new_data)

print("Predicted CA125 value:", prediction[0])

# Decision-making based on the prediction
threshold = 100  # Replace with your desired threshold
if prediction[0] > threshold:
    print("The CA125 value indicates a condition above the threshold.")
    # Add additional decisions or actions based on the condition
else:
    print("The CA125 value indicates a condition below the threshold.")
    # Add additional decisions or actions based on the condition
```

Figure 5.5.1 Prediction and Decision Making

```
Predicted CA125 value: 87.53537718876225
The CA125 value indicates a condition below the threshold.
```

Figure 5.5.2 Prediction and Decision Making : Results

The predicted CA125 value is 87.54, and based on the decision-making logic in the code, it indicates a condition below the specified threshold of 100. This means that, according to the model and the chosen threshold, the CA125 value is considered to be within a normal or acceptable range.

5.6 Chapter Summary

CHAPTER 6

MODEL EVALUATION (SKLEARN)

6.0 Overview

This chapter discusses model evaluation, over-fitting, under-fitting, model selection,

6.1 Model Evaluation

The code below performs model evaluation for polynomial regression.It provides quantitative metrics (MSE and R2) and a visual representation of the model's performance on the test set.

```python
# 6.1 Model Evaluation

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Create a polynomial regression pipeline
degree = 2  # You can adjust the degree of the polynomial
polynomial_regression = Pipeline([
    ('poly_features', PolynomialFeatures(degree=degree)),
    ('linear_regression', LinearRegression())
])

# Fit the pipeline on the training data
polynomial_regression.fit(X_train, y_train)

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

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Plot the results
plt.scatter(X_test, y_test, label='Actual data')
plt.plot(X_test, y_pred, label='Polynomial regression', color='red')
plt.xlabel('Age')
plt.ylabel('CA125')
plt.title('Polynomial Regression - Test Set Evaluation')
plt.legend()
plt.show()
```

Figure 6.1.1 Model Evaluation

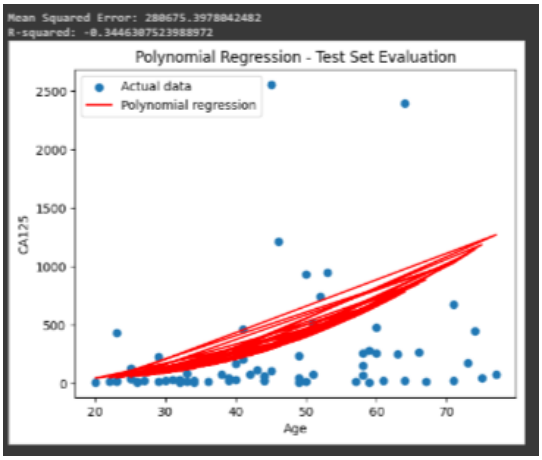


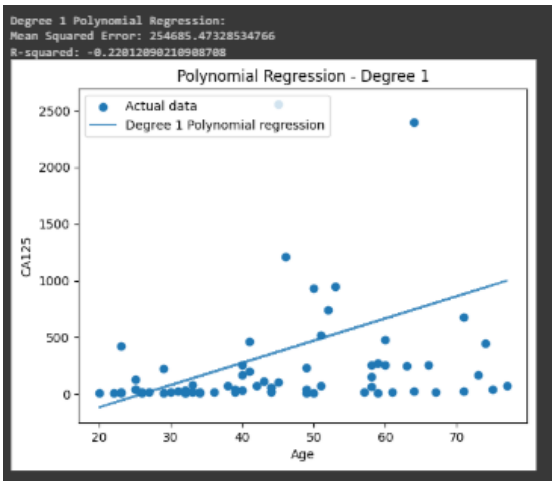
Figure 6.1.2 Model Evaluation : Results

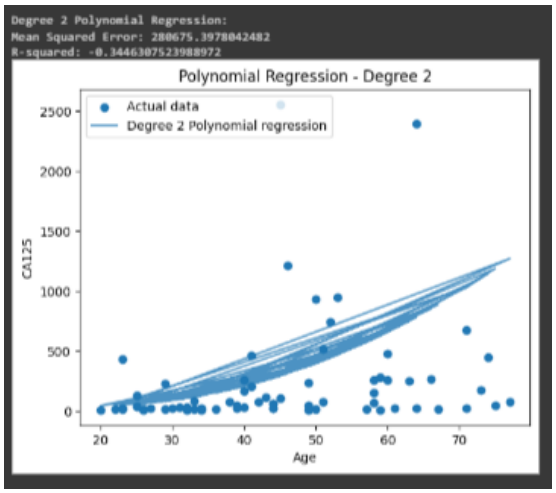
6.2 Over-fitting, Under-fitting and Model Selection

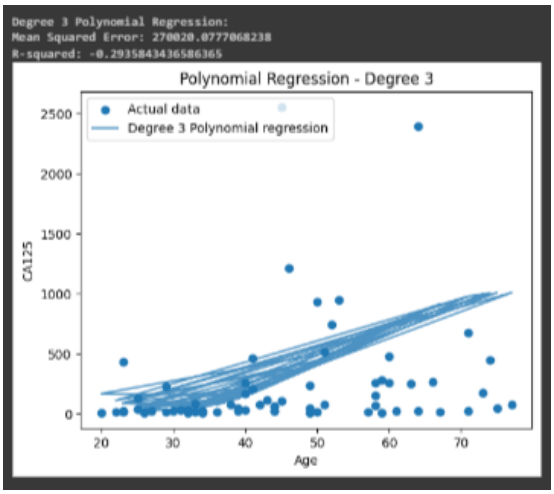
The code below performs a model selection process by trying different degrees of polynomial features and evaluating the models' performance, aiming to strike a balance between over-fitting and under-fitting. The visualizations and metrics help in making an informed decision about the appropriate complexity of the polynomial regression model for the given dataset**.**

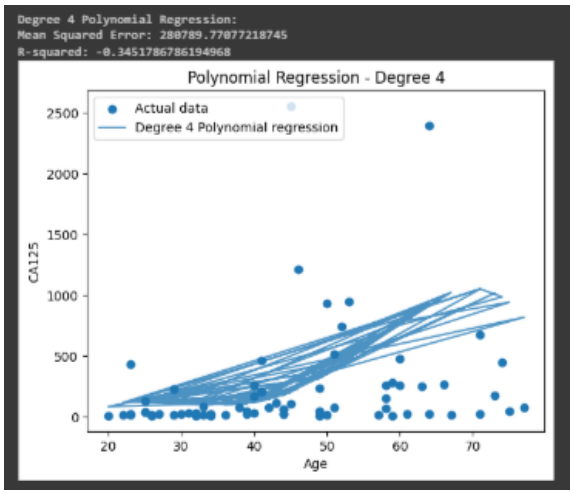
```python
# 6.2 Over-Fitting, Under-Fitting & Model Selection

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Set up degrees for polynomial features
degrees = [1, 2, 3, 4, 5]

for degree in degrees:
    # Create a polynomial regression pipeline
    polynomial_regression = Pipeline([
        ('poly_features', PolynomialFeatures(degree=degree)),
        ('linear_regression', LinearRegression())
    ])

    # Fit the pipeline on the training data
    polynomial_regression.fit(X_train, y_train)

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

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"\nDegree {degree} Polynomial Regression:")
    print("Mean Squared Error:", mse)
    print("R-squared:", r2)

    # Plot the results
    plt.scatter(X_test, y_test, label='Actual data')
    plt.plot(X_test, y_pred, label=f'Degree {degree} Polynomial regression', alpha=0.8)
    plt.xlabel('Age')
    plt.ylabel('CA125')
    plt.title(f'Polynomial Regression - Degree {degree}')
    plt.legend()
    plt.show()
```

Figure 6.2.1 Over-fitting, Under-fitting and Model Selection









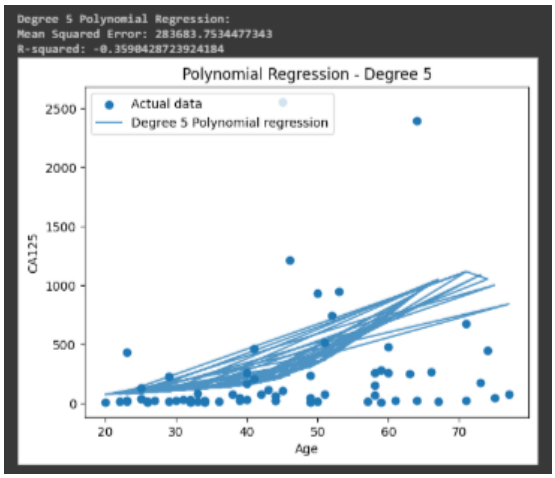


Figure 6.2.2 Over-fitting, Under-fitting and Model Selection : Results

6.3 Ridge Regression

Ridge Regression is used to regularize linear regression models, and the code below explores the effects of different regularization strengths on model performance through evaluation metrics and visualizations.

```python
# 6.3 Ridge Regression

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Standardize the features (important for Ridge Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Set up alpha values for Ridge Regression
alphas = [0.01, 0.1, 1, 10, 100]

for alpha in alphas:
    # Create Ridge Regression model
    ridge_model = Ridge(alpha=alpha)

    # Fit the model on the scaled training data
    ridge_model.fit(X_train_scaled, y_train)

    # Make predictions on the scaled test set
    y_pred = ridge_model.predict(X_test_scaled)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"\nRidge Regression with Alpha={alpha}:")
    print("Mean Squared Error:", mse)
    print("R-squared:", r2)

    # Plot the results
    plt.scatter(X_test, y_test, label='Actual data')
    plt.plot(X_test, y_pred, label=f'Ridge Regression (Alpha={alpha})', alpha=0.8)
    plt.xlabel('Age')
    plt.ylabel('CA125')
    plt.title(f'Ridge Regression - Alpha={alpha}')
    plt.legend()
    plt.show()
```

Figure 6.3.1 Ridge Regression

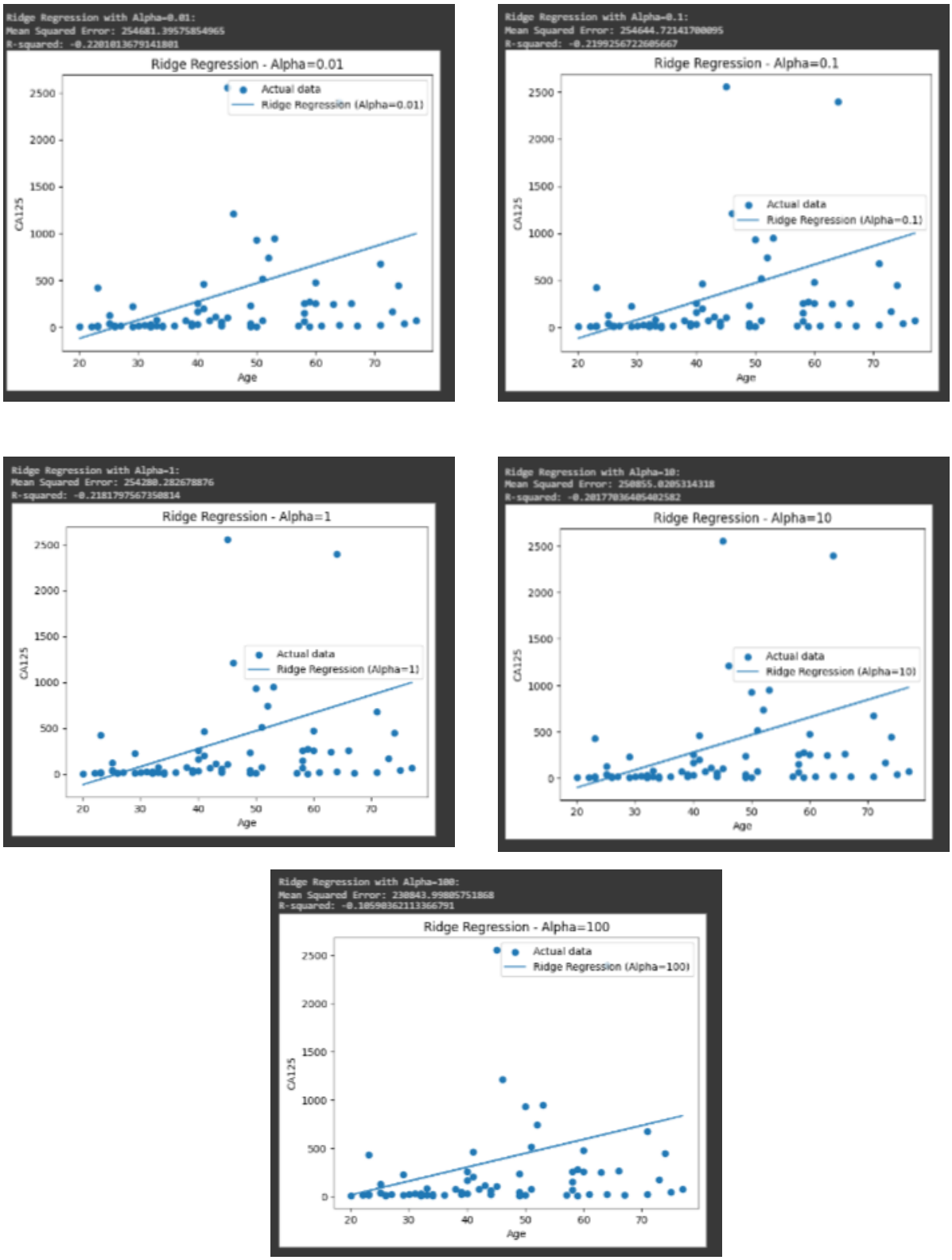


Figure 6.3.2 Ridge Regression : Results

6.4 Grid Search

The code below is implementing Grid Search for hyperparameter tuning, specifically for Ridge Regression. Grid Search is applied to Ridge Regression to systematically search for the best alpha value, and the code demonstrates how to use it to find the optimal hyperparameter for improved model performance.

```python
# 6.4 Grid Search

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Standardize the features (important for Ridge Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Set up alpha values for Ridge Regression
alphas = [0.01, 0.1, 1, 10, 100]

# Create a Ridge Regression model
ridge_model = Ridge()

# Set up the parameter grid for grid search
param_grid = {'alpha': alphas}

# Create the GridSearchCV object
grid_search = GridSearchCV(ridge_model, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the grid search on the scaled training data
grid_search.fit(X_train_scaled, y_train)

# Get the best hyperparameters from the grid search
best_alpha = grid_search.best_params_['alpha']

# Print the best hyperparameters
print("Best Alpha:", best_alpha)

# Make predictions on the scaled test set using the best model
y_pred = grid_search.predict(X_test_scaled)

# Evaluate the best model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nBest Model Evaluation:")
print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Plot the results
plt.scatter(X_test, y_test, label='Actual data')
plt.plot(X_test, y_pred, label=f'Best Ridge Regression (Alpha={best_alpha})', alpha=0.8)
plt.xlabel('Age')
plt.ylabel('CA125')
plt.title(f'Best Ridge Regression - Alpha={best_alpha}')
plt.legend()
plt.show()
```

Figure 6.4.1 Grid Search

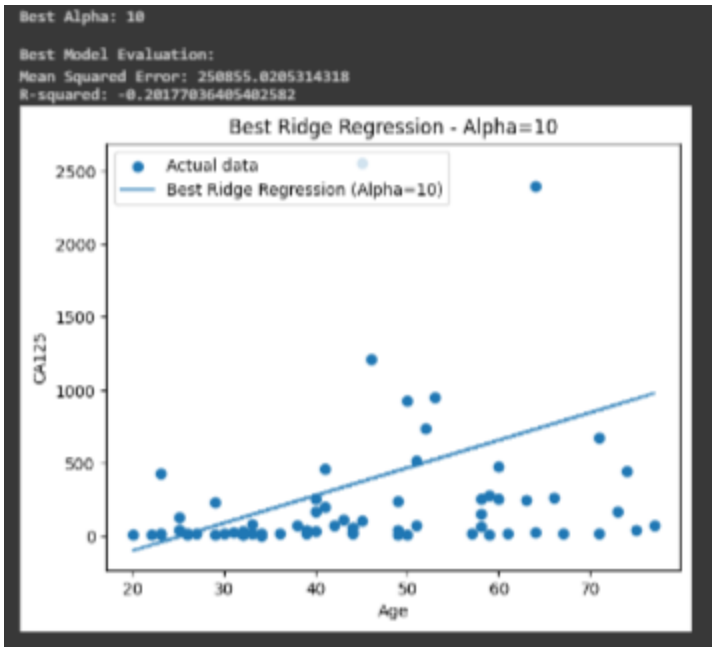


Figure 6.4.2 Grid Search : Results

6.5 Model Refinement

Model refinement in this code involves adjusting the hyperparameter (alpha) of the Ridge Regression model based on the best value found during a grid search. The refined model is then evaluated and visualized to assess its improved performance.

```python
# 6.5 Model Refinement

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume you have a CSV file with your data
url = 'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Project/main/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the dependent variable
X = df[['Age']].values
y = df['CA125'].values

# 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)

# Standardize the features (important for Ridge Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Refine the Ridge Regression model based on grid search results
best_alpha = 0.01  # Replace with the best alpha value obtained from grid search

# Create the refined Ridge Regression model
refined_ridge_model = Ridge(alpha=best_alpha)

# Fit the refined model on the scaled training data
refined_ridge_model.fit(X_train_scaled, y_train)

# Make predictions on the scaled test set
y_pred = refined_ridge_model.predict(X_test_scaled)

# Evaluate the refined model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nRefined Ridge Regression Model:")
print("Best Alpha:", best_alpha)
print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Plot the results
plt.scatter(X_test, y_test, label='Actual data')
plt.plot(X_test, y_pred, label=f'Refined Ridge Regression (Alpha={best_alpha})', alpha=0.8)
plt.xlabel('Age')
plt.ylabel('CA125')
plt.title(f'Refined Ridge Regression - Alpha={best_alpha}')
plt.legend()
plt.show()
```

Figure 6.5.1 Model Refinement

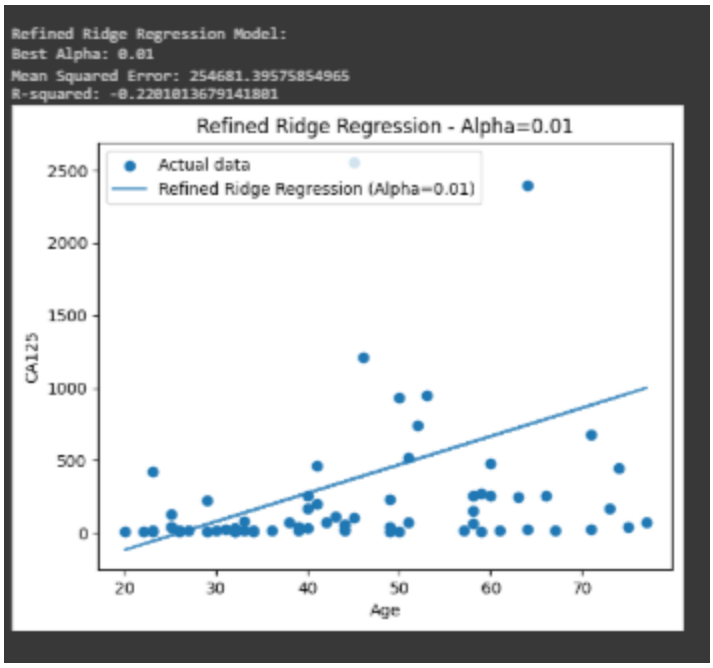


Figure 6.5.2 Model Refinement : Results

6.6 Chapter Summary