SCHOOL OF COMPUTING
Faculty of Engineering

SECB3203-01

PROGRAMMING FOR BIOINFORMATICS

23/24 - 1

GROUP 6

FINAL REPORT

CLASSIFICATION OF OVARIAN CANCER PROGNOSIS USING DEEP NEURAL NETWORK (DNN) ALGORITHM

Submitted to:

Dr. Nies Hui Wen

GROUP MEMBERS	MATRIC NUMBER
AISYAH BINTI MOHD NADZRI	A21EC0011
THUVAARITHA SIVARAJAH	A21EC0137

TABLE OF CONTENTS

TITLE	PAGE
TABLE OF CONTENTS	2 - 3
LIST OF TABLES	4
LIST OF FIGURES	5 - 7
CHAPTER 1 INTRODUCTION	
1.0 Introduction	8 -9
1.1 Problem Background	9 - 10
1.2 Problem Statement	10
1.3 Objectives	11
1.4 Scopes	12 - 15
1.5 Chapter Summary	16
CHAPTER 2 IMPORTING DATASET AND DATA WRANGLING	
2.0 Data Collection & Pre-processing	17 18
2.1 Importing Dataset	19 - 25
2.2 Data pre-processing and Exploratory data analysis	26 - 61
2.3 Software and Hardware Requirements	62
2.4 Chapter Summary	62 - 63

CHAPTER 3 RESEARCH IMPLEMENTATION

3.0 Flowchart of The Proposed Approach 64 - 67

3.1 Model Development 68 - 95

3.2 Model Evaluation 96 - 114

CHAPTER 4 RESULTS , ANALYSIS AND DISCUSSION

4.0 Results, Testing and Validation 115 - 117

4.1 Feedback From Client(s) 117

CHAPTER 5 CONCLUSION

5.0 Conclusions 118

CHAPTER 6 APPENDIX

6.0 Appendix 119

LIST OF TABLES

TABLE NO.	TITLE	PAGE
Table 1.4.0.1	The criteria of dataset	12

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
Figure 2.0.1	Case Study	17
Figure 2.0.2	Datasets	18
Figure 2.0.3	Ovariantotal.csv dataset preview	19
Figure 2.1.1.1	Importing dataset into Google Colab using Python	20
Figure 2.1.1.2	Importing dataset into Google Colab using Python results	20
Figure 2.1.2.1	Number of Rows & Columns	21
Figure 2.1.2.2	Number of Rows & Columns Results	21
Figure 2.1.2.3	Data type of each column	21
Figure 2.1.2.4	Data type of each column results	22
Figure 2.2.0.1	Identifying Missing Values using pandas	26
Figure 2.2.0.2	Identifying Missing Values using pandas results	26
Figure 2.2.1.1	Data Formatting: Change original column names to specific names	29
Figure 2.2.1.2	Data Formatting: Change original column names to specific names	30
Figure 2.2.1.3	Data Formatting: Change original column names to specific names output	30
Figure 2.2.1.4	Display updated column names	31
Figure 2.2.2.1	Feature Selection	33
Figure 2.2.2.2	Feature Selection results	33
Figure 2.2.3.1	Data Normalization: Standardization	36
Figure 2.2.3.2	Data Normalization: Standardization Results	37
Figure 2.2.4.1	Data Binning : ‘MCH’ Column	39
Figure 2.2.4.2	Data Binning : Results after data binning for ‘MCH’ Column	40
Figure 2.2.5.1	Indicator Variables : Indicator Variable for the ‘TYPE’ variable	42

Figure 2.2.5.2	Indicator Variables : Results	42
Figure 2.2.6.1	Descriptive Statistics : Descriptive Statistics for Selected Features	45
Figure 2.2.6.2	Descriptive Statistics : Summary of main characteristics	46
Figure 2.2.7.1	Basics of Grouping : Method	50
Figure 2.2.7.2	Basics of Grouping : Results	51
Figure 2.2.8.1	ANOVA : ANOVA for ‘AGE’ and ‘PLT’ Columns	54
Figure 2.2.8.2	ANOVA : ANOVA for ‘AGE’ and ‘PLT’ Columns Results	54
Figure 2.2.8.2	ANOVA : ANOVA for selected features and ‘CA125’	55
Figure 2.2.8.3	ANOVA : ANOVA for selected features and ‘CA125’ Results	55
Figure 2.2.8.4	ANOVA : ANOVA for ‘MCH’ and ‘TYPE’	56
Figure 2.2.8.5	ANOVA : ANOVA for ‘MCH’ and ‘TYPE’ Results	56
Figure 2.2.9.1	Correlation : Correlation between ‘MCH’ and ‘TYPE’	59
Figure 2.2.9.2	Correlation : Correlation between ‘MCH’ and ‘TYPE’ Results	59
Figure 2.2.9.3	Correlation : Correlation between selected features and ‘CA125’	60
Figure 2.2.9.4	Correlation : Correlation between selected features and ‘CA125’ Results	60
Figure 3.0.1	Proposed Flowchart	64
Figure 3.1.1.1	Simple and Multiple Linear Regression	69
Figure 3.1.1.2	Simple and Multiple Linear Regression Result	69
Figure 3.1.2.1	Simple and Multiple Linear Regression : Visual Representation for Simple Linear Regression	72
Figure 3.1.2.2	Simple and Multiple Linear Regression : Visual Representation for Simple Linear Regression Result	72
Figure 3.1.3.1	Simple and Multiple Linear Regression : Visual Representation for Multiple Linear Regression	75
Figure 3.1.3.2	Simple and Multiple Linear Regression : Visual Representation for Multiple Linear Regression Result	75
Figure 3.1.4.1	Model Evaluation Using Visualization : Method	78

Figure 3.1.4.2	Model Evaluation Using Visualization : Results	79
Figure 3.1.4.3	Model Evaluation Using Visualization : Method	81
Figure 3.1.4.4	Model Evaluation Using Visualization : Results	82
Figure 3.1.5.1	Polynomial Regression and Pipelines : Polynomial Regression Without Pipelines	84
Figure 3.1.5.2	Polynomial Regression and Pipelines : Results of Polynomial Regression Without Pipelines	85
Figure 3.1.5.3	Polynomial Regression and Pipelines : Polynomial Regression With Pipelines	87
Figure 3.1.5.4	Polynomial Regression and Pipelines :Results of Polynomial Regression With Pipelines	88
Figure 3.1.6.1	R-Squared and MSE for In-Sample Training	91
Figure 3.1.6.2	R-Squared and MSE for In-Sample Training : Results	91
Figure 3.1.7.1	Prediction and Decision Making : Method and Results	94
Figure 3.2.0.1	Model Evaluation : Polynomial Regression Method & Results	97
Figure 3.2.1.1	Model Evaluation : Overfitting, Under-fitting & Model Selection	100
Figure 3.2.1.2	Model Evaluation : Overfitting, Under-fitting & Model Selection Results	101
Figure 3.2.2.1	Ridge Regression	104
Figure 3.2.2.2	Ridge Regression : Results	105
Figure 3.2.3.1	Grid Search	109
Figure 3.2.3.2	Grid Search : Results	109
Figure 3.2.4.1	Model Refinement	112
Figure 3.2.4.2	Model Refinement : Result	113
Figure 4.0.1.1	Results	116

CHAPTER 1

INTRODUCTION

1.0 Introduction

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.

1.1 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.

1.2 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.

1.3 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.

1.4 Scopes

1.4.0 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 1.4.0.1 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.

1.4.1 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.

1.4.2 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.

1.4.3 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.

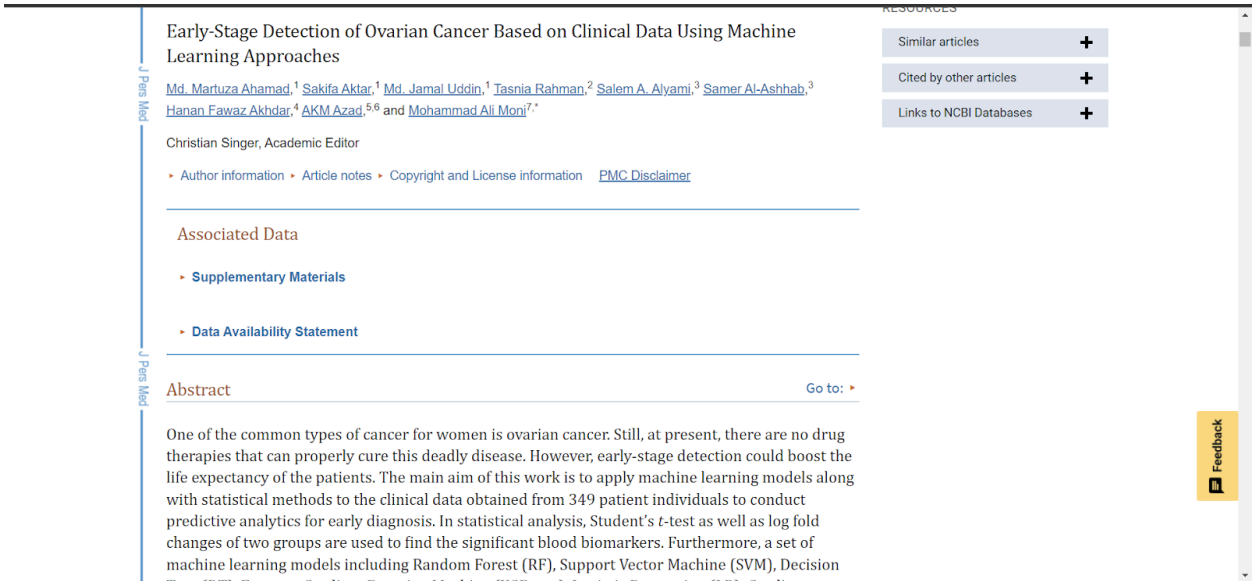
Last but foremost, by utilizing deep learning techniques, the findings of this study present a viable option for improving the care of patients with ovarian cancer. This study should assist in advancing the continuing endeavor to enhance early detection and treatment approaches for ovarian cancer by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

CHAPTER 2

IMPORTING DATASET AND DATA WRANGLING

2.0 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

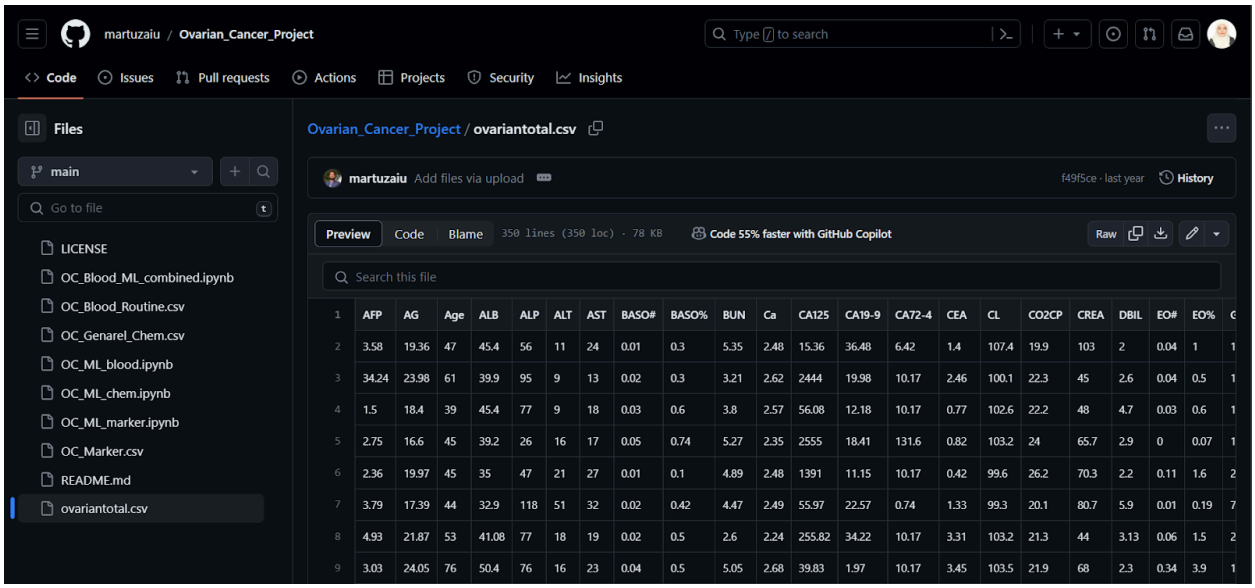
Feedback

J Pers Med

J Pers Med

Figure 2.0.1 Case Study

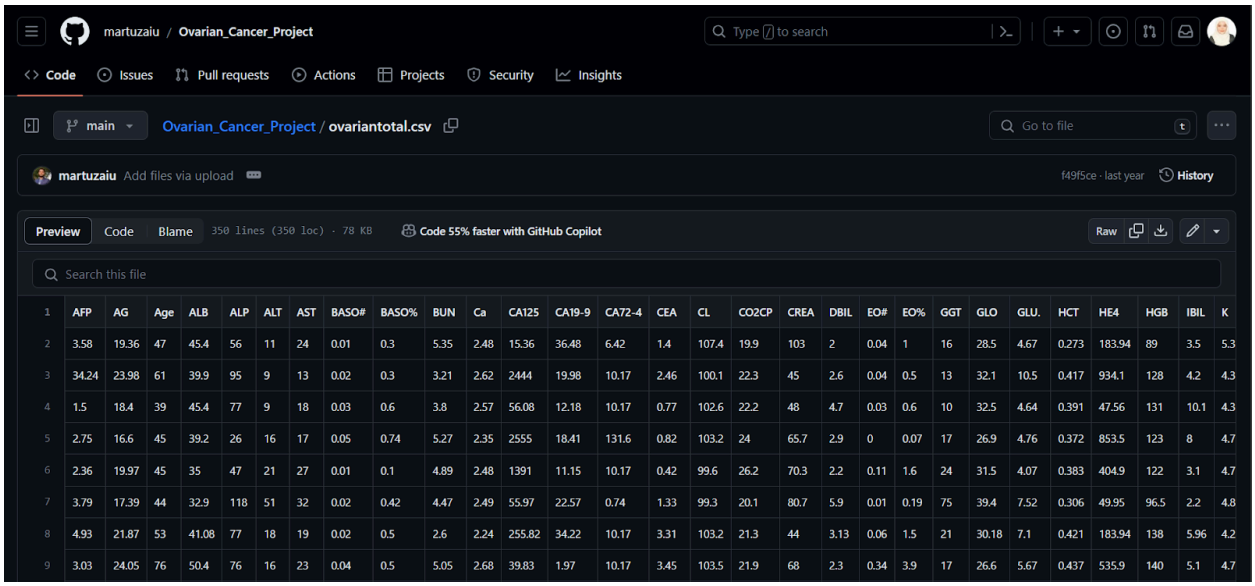
We can view the data availability from the github link provided by the authors. It displays here the raw datasets of combined dataset, blood test samples dataset, general chemistry test dataset and cancer marker dataset. The authors likewise provided the jupyter notebook version of each datasets.



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 2.0.2 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 2.0.3 Ovariantotal.csv dataset preview

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

2.1 Importing Dataset

2.1.0 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.

2.1.1 Importing and Exporting Data in Python

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

```
# 2.1.1: Importing Dataset

# GitHub raw URL for the CSV file
url =
'https://raw.githubusercontent.com/martuzaiu/Ovarian_Cancer_Projec
t/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()
```

Figure 2.1.1.1 Importing dataset into Google Colab using Python

	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 2.1.1.2 Importing dataset into Google Colab using Python Results

2.1.2 Getting Started Analyzing Data in Python

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

```
# 2.1.2 Understanding the Data

# 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])
```

Figure 2.1.2.1 Number of Rows & Columns

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

Figure 2.1.2.2 Number of Rows & Columns Results

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

```
# 2.1.2 Understanding the Data

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

Figure 2.1.2.3 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 2.1.2.4 Data type of each column result

2.1.3 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

2.2 Data Preprocessing and Exploratory Data Analysis : Data Wrangling (Pandas Numpy)

2.2.0 Identifying and Handling Missing Values

```python
# 2.2 Data Wrangling

# 2.2.0 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 2.2.0.1 Identifying Missing Values using pandas.

```
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 2.2.0.2 Missing Values Identification 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.

2.2.1 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.

```
# 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_Projec
t/main/ovariantotal.csv'
df = pd.read_csv(url)

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

```python
# 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',
    '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',
    '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 2.2.1.1 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 2.2.1.2 Data Formatting: Change original column names to specific names

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

from IPython.display import display

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

Figure 2.2.1.3 Data Formatting: Change original column names to specific names output

	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	Red Blood Cell Distribution Width	Total Bilirubin	Total Protein	Uric Acid	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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	11.9	10.7	80.4	241.5	1
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	17.7	17.2	73.5	240.5	1
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	11.6	13.8	68.4	209.5	1
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	12.5	11.3	76.0	285.7	1
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	12.1	8.3	68.6	184.9	1

349 rows × 50 columns

Figure 2.2.1.4 Display updated column names

In the provided code snippet, a dataset related to ovarian cancer is loaded into a Pandas DataFrame from a specified URL, and the original column names are displayed. Subsequently, a critical step in data preprocessing is performed by renaming specific columns using a predefined mapping dictionary (`column_mapping`). The original, potentially cryptic, or abbreviated column names are substituted with more descriptive and interpretable names, reflecting domain-specific knowledge related to ovarian cancer research and medical parameters. This transformation is fundamental for improving the dataset's interpretability, facilitating better communication, and setting the stage for more effective data analysis and exploration.

After the renaming process, the updated column names include descriptors such as 'Alpha-Fetoprotein,' 'Blood Urea Nitrogen,' 'Carbohydrate Antigen 125,' and many others. These clear and meaningful names convey the biological or clinical significance of each variable, making it easier for researchers, clinicians, and data analysts to identify and understand the content of the dataset. Finally, the formatted DataFrame with the new column names is saved to a CSV file named 'formatted_ovarian_dataset.csv.' This transformed dataset is now more accessible and user-friendly, supporting a wide range of analytical applications and research efforts in the domain of ovarian cancer studies.

2.2.2 Feature Selection

```
# 2.2.2 Feature Selection
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

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

# Assuming 'TYPE' is your target variable, you may need to adjust this
based on your dataset
X = df.drop(columns=['TYPE'])  # Features
y = df['TYPE']  # Target variable

# 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)

# Choose a machine learning model (e.g., Logistic Regression)
model = LogisticRegression()

# Create RFE model and select the number of features to retain (adjust
as needed)
num_features_to_select = 10
rfe = RFE(model, n_features_to_select=num_features_to_select)

# Fit the RFE model to your data
rfe.fit(X_train, y_train)

# Get the selected features
selected_features = X_train.columns[rfe.support_]
```

```python
# Display the selected features
print("\nSelected Features:")
print(selected_features)

# Display the ranking of features from the best to the least important
ranking_results = pd.DataFrame({'Feature': X_train.columns, 'Ranking':
rfe.ranking_})
ranking_results = ranking_results.sort_values(by='Ranking')

print("\nFeature Rankings:")
print(ranking_results)
```

Figure 2.2.2.1 Feature Selection

```
Feature Rankings:
      Feature  Ranking
31        MCH        1
20        EO%        1
23       GLU.        1
18       DBIL        1
14        CEA        1
33  Menopause        1
37        MPV        1
28          K        1
32        MCV        1
6         AST        1
34         Mg        2
8       BASO%        3
```

Figure 2.2.2.2 Feature Selection

In the provided code, a comprehensive approach to feature selection is demonstrated using Recursive Feature Elimination (RFE) in the context of ovarian cancer research. The dataset, loaded from a specified URL, comprises various features, with the 'TYPE' column assumed to be the target variable. The dataset is then split into training and testing sets to facilitate model evaluation. In this example, a Logistic Regression model is chosen for its simplicity and interpretability.

The RFE process is initiated by creating an RFE model with the Logistic Regression estimator, and the number of features to retain is specified as 10 (though this can be adjusted based on the specific requirements of the analysis). The RFE model is then fitted to the training data, and the features deemed most relevant are identified. The selected features are displayed to provide insights into the subset of variables that the model deems significant in predicting the target variable.

Additionally, the code generates a feature ranking list, showcasing the importance of each feature from the best to the least important. This ranking is invaluable for understanding the relative contribution of each variable in predicting ovarian cancer outcomes. Researchers can leverage this information to prioritize and focus on the most influential features during subsequent analyses or when building more complex models. The RFE approach, coupled with feature ranking, aids in refining the feature set and enhances the interpretability and efficiency of machine learning models in the context of ovarian cancer research.

2.2.3 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.

```
# 2.2.3 Data Normalization
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

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

# Assuming 'TYPE' is your target variable, you may need to adjust this
based on your dataset
X = df.drop(columns=['TYPE'])  # Features
y = df['TYPE']  # Target variable

# Selected features and their rankings
selected_features_info = [
    {'Feature': 'MCH', 'Ranking': 1},
    {'Feature': 'EO%', 'Ranking': 1},
    {'Feature': 'GLU.', 'Ranking': 1},
    {'Feature': 'DBIL', 'Ranking': 1},
    {'Feature': 'CEA', 'Ranking': 1},
    {'Feature': 'Menopause', 'Ranking': 1},
    {'Feature': 'MPV', 'Ranking': 1},
    {'Feature': 'K', 'Ranking': 1},
    {'Feature': 'MCV', 'Ranking': 1},
    {'Feature': 'AST', 'Ranking': 1},
    {'Feature': 'Mg', 'Ranking': 2},
    {'Feature': 'BASO%', 'Ranking': 3},
```

```
]

# Extract selected feature names
selected_features = [info['Feature'] for info in
selected_features_info]

# Create a new DataFrame with only the selected features
df_selected_features = df[selected_features + ['TYPE']]  # Include the
target variable 'TYPE'

# Separate the features and target variable
X_selected = df_selected_features[selected_features]

# Standardize the selected features
scaler = StandardScaler()
X_selected_standardized = scaler.fit_transform(X_selected)

# Create a DataFrame with standardized features
df_selected_standardized = pd.DataFrame(X_selected_standardized,
columns=selected_features)

# Visualize the distribution before and after normalization
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Before normalization
sns.histplot(data=X_selected, bins=20, kde=True, ax=axes[0])
axes[0].set_title('Distribution Before Normalization')
axes[0].set_xlabel('Feature Value')

# After normalization
sns.histplot(data=df_selected_standardized, bins=20, kde=True,
ax=axes[1])
axes[1].set_title('Distribution After Standardization')
axes[1].set_xlabel('Standardized Feature Value')

plt.tight_layout()
plt.show()
```

Figure 2.2.3.1 Data Normalization: Standardization

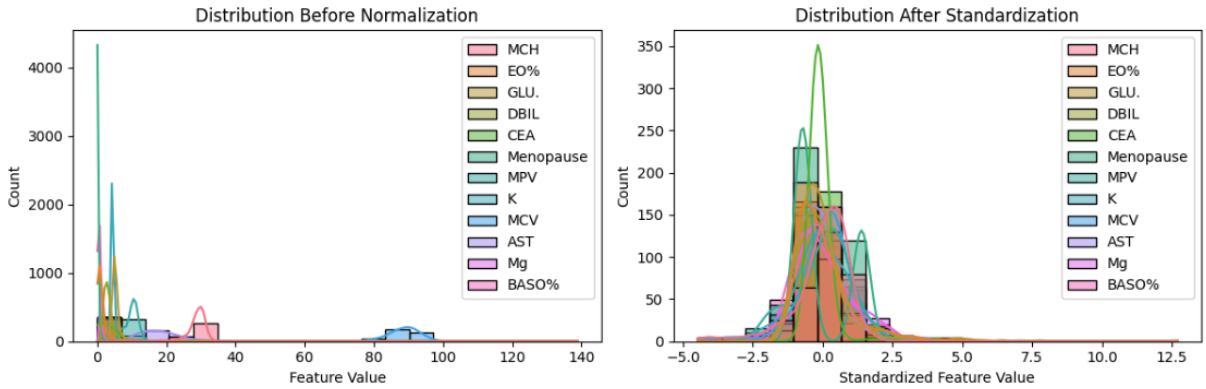


Figure 2.2.3.2 Data Normalization: Standardization Results

In the provided code, an exploratory analysis of selected features from an ovarian cancer dataset is conducted, with a focus on the impact of feature standardization. The dataset is loaded from a specified URL, and features are selected based on their rankings as obtained from a feature selection process. Specifically, 11 features are chosen along with their corresponding rankings, providing valuable insights into the most relevant variables for predicting the 'TYPE' of ovarian cancer. These selected features are then extracted from the original dataset to create a new DataFrame, which includes the target variable 'TYPE.' The feature values are presented both before and after standardization to visually assess the distribution changes.

To prepare the selected features for analysis, the values are standardized using the `StandardScaler` from scikit-learn. Standardization transforms the data, ensuring that each feature has a mean of 0 and a standard deviation of 1. This process is crucial when features have varying scales, as it prevents certain variables from disproportionately influencing the model due to their larger magnitude. The code

creates a new DataFrame (`df_selected_standardized`) containing the standardized values of the selected features. This standardized representation facilitates a fairer comparison of the features, enabling machine learning models to better understand and interpret their contributions to the predictive task.

The visualization component of the code consists of two histograms, each representing the distribution of feature values. The first histogram illustrates the original feature values before standardization, providing insight into their raw distributions. The second histogram depicts the same features after standardization, showcasing the impact of the normalization process. By visualizing these distributions side by side, one can observe how standardization aligns the scales of the selected features, making them more comparable and suitable for modeling. Overall, this code segment exemplifies a crucial preprocessing step, ensuring that feature values are standardized for improved model performance and interpretability in the context of ovarian cancer research.

2.2.4 Binning

Binning is a process of transforming continuous numerical variables into discrete categorical 'bins', for grouped analysis. In our dataset, 'MCH' is a real valued variable with unique values.

```
# 2.2.4 Data Binning
# MCH 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 'MCH' column for binning
column_to_bin = 'MCH'

# Defining the bin edges
bin_edges = [0, 27, 31, float('inf')]  # Values between 27 and 31 are
normal, below 27 is low, and above 31 is high

# Defining bin labels
bin_labels = ['Low', 'Normal', 'High']

# Create a new column with bin labels
df['MCH_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 'MCH' column only
display(df[['MCH', 'MCH_Bin']])
```

Figure 2.2.4.1 Data Binning : 'MCH' Column

	MCH	MCH_Bin
0	33.7	High
1	26.2	Low
2	28.4	Normal
3	30.6	Normal
4	27.7	Normal
...	...	...
344	31.1	High
345	18.9	Low
346	29.4	Normal
347	28.8	Normal
348	31.0	Normal

349 rows × 2 columns

Figure 2.2.4.2 Data Binning : Results after data binning for 'MCH' Column

In the provided code snippet, the concept of data binning is applied to the 'MCH' (Mean Corpuscular Hemoglobin) column within an ovarian cancer dataset. Data binning involves categorizing continuous data into discrete bins or intervals, providing a more structured representation of the information. In this case, the 'MCH' column is chosen for binning, and the bin edges are defined as [0, 27, 31, infinity]. These bin edges create three distinct intervals: 'Low' for values below 27, 'Normal' for values between 27 and 31 (inclusive), and 'High' for values above 31. These intervals are chosen based on clinically relevant thresholds for 'MCH,' where values outside the normal range may be indicative of specific medical conditions. The resulting binned data is stored in a new column labeled 'MCH_Bin,' providing a categorical representation of 'MCH' values.

The code efficiently utilizes the Pandas library to implement data binning. The 'cut' function is employed to assign each 'MCH' value to its corresponding bin label based on the defined bin edges and labels. This categorization enhances the interpretability of the 'MCH' column, enabling a clearer understanding of the distribution of values within clinically relevant ranges. The display function showcases a subset of the DataFrame, specifically presenting the original 'MCH' values alongside their corresponding bin labels. This visualization aids in assessing the impact of the binning process on the distribution of 'MCH' values, making it easier to identify patterns or anomalies within different categories.

In summary, the code illustrates a practical application of data binning to the 'MCH' column in an ovarian cancer dataset. This preprocessing technique transforms continuous data into discrete categories, offering a more intuitive representation of the 'MCH' values and potentially facilitating subsequent analyses that rely on categorical information. The choice of bin edges and labels reflects domain-specific knowledge and clinical considerations, ensuring that the binning process aligns with the relevant medical context.

2.2.5 Indicator Variables

```python
# 2.2.5 Indicator Variable
# indicator variable for the 'TYPE' variable

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)

# Assuming df is your DataFrame
target_variable = 'TYPE'

# Create indicator variables for 'TYPE'
indicator_variables = pd.get_dummies(df[target_variable],
prefix=target_variable)

# Concatenate the indicator variables with the original DataFrame
df_with_indicators = pd.concat([df, indicator_variables], axis=1)

# Display the entire DataFrame
print(df_with_indicators)
```

Figure 2.2.5.1 Indicator Variables : Indicator Variable for the 'TYPE' variable

```
       AFP     AG  Age   ALB  ALP  ALT  AST  BASO#  BASO%   BUN  ...  PHOS
0     3.58  19.36   47  45.4   56   11   24   0.01   0.30  5.35  ...  1.46
1    34.24  23.98   61  39.9   95    9   13   0.02   0.30  3.21  ...  1.09
2     1.50  18.40   39  45.4   77    9   18   0.03   0.60  3.80  ...  0.97
3     2.75  16.60   45  39.2   26   16   17   0.05   0.74  5.27  ...  1.25
4     2.36  19.97   45  35.0   47   21   27   0.01   0.10  4.89  ...  0.94
..     ...    ...  ...   ...  ...  ...  ...    ...    ...   ...  ...   ...
344   2.09  20.91   52  48.6   65   40   30   0.00   0.00  4.29  ...  0.99
345   2.00  12.14   37  46.1   49    9   20   0.00   0.00  3.12  ...  0.95
346   2.83  19.23   59  45.8   83    9   16   0.02   0.60  4.38  ...  1.00
347   1.84  27.22   30  47.8   64   26   22   0.03   0.30  3.61  ...  1.31
348   1.61  13.00   39  44.9   68    7   13   0.01   0.20  4.38  ...  1.10

     PLT   RBC   RDW  TBIL    TP     UA  TYPE  TYPE_0  TYPE_1
0     74  2.64  13.7   5.5  73.9  396.4     0       1       0
1    304  4.89  12.7   6.8  72.0  119.2     0       1       0
2    112  4.62  12.0  14.8  77.9  209.2     0       1       0
3    339  4.01  14.6  10.9  66.1  215.6     0       1       0
4    272  4.40  13.4   5.3  66.5  206.0     0       1       0
..   ...   ...   ...   ...   ...    ...   ...     ...     ...
344  219  4.56  11.9  10.7  80.4  241.5     1       0       1
345  262  4.24  17.7  17.2  73.5  240.5     1       0       1
346  221  4.89  11.6  13.8  68.4  209.5     1       0       1
347  398  5.35  12.5  11.3  76.0  285.7     1       0       1
348  224  4.13  12.1   8.3  68.6  184.9     1       0       1

[349 rows x 52 columns]
```

Figure 2.2.5.2 Indicator Variables : Results

In this provided code snippet, the concept of an indicator variable is implemented for the 'TYPE' variable within an ovarian cancer dataset. Indicator variables, also known as dummy variables, are binary representations of categorical variables, often utilized in machine learning models that require numerical input. The 'TYPE' variable is assumed to be the target variable in this context. The `pd.get_dummies` function is employed to create binary indicators for each unique category within the 'TYPE' variable. This process effectively transforms the categorical 'TYPE' variable into a set of binary columns, each corresponding to a unique category. The resulting DataFrame, named `df_with_indicators`, is a combination of the original dataset and the newly created indicator variables.

The use of indicator variables is crucial in scenarios where machine learning algorithms require numerical input and cannot directly handle categorical data. The binary nature of indicator variables preserves the categorical information while making it compatible with models that expect numeric features. In this case, the 'TYPE' variable, which likely represents different types or classes of ovarian cancer, is now represented as binary columns in the DataFrame. Each binary column corresponds to a specific type, and the presence of a '1' in a column indicates the occurrence of that particular type, providing a more suitable format for machine learning algorithms.

By incorporating indicator variables, the dataset is better prepared for predictive modeling tasks, such as classification algorithms, where the objective is to predict the 'TYPE' of ovarian cancer based on other features. The display of the entire DataFrame allows for a comprehensive examination of the dataset, showcasing the original features alongside the newly created indicator variables. This transformation is a fundamental preprocessing step in machine learning workflows, ensuring that categorical information is appropriately encoded for effective model training and prediction in the domain of ovarian cancer research.

2.2.6 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.

```
# 2.2.6 Descriptive Statistics
import pandas as pd
from sklearn.preprocessing import StandardScaler

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

# Assuming 'TYPE' is your target variable, you may need to adjust this
based on your dataset
X = df.drop(columns=['TYPE'])  # Features
y = df['TYPE']  # Target variable

# Selected features and their rankings
selected_features_info = [
    {'Feature': 'MCH', 'Ranking': 1},
    {'Feature': 'EO%', 'Ranking': 1},
    {'Feature': 'GLU.', 'Ranking': 1},
    {'Feature': 'DBIL', 'Ranking': 1},
    {'Feature': 'CEA', 'Ranking': 1},
    {'Feature': 'Menopause', 'Ranking': 1},
    {'Feature': 'MPV', 'Ranking': 1},
    {'Feature': 'K', 'Ranking': 1},
    {'Feature': 'MCV', 'Ranking': 1},
    {'Feature': 'AST', 'Ranking': 1},
    {'Feature': 'Mg', 'Ranking': 2},
    {'Feature': 'BASO%', 'Ranking': 3},
]
```

```
# Extract selected feature names
selected_features = [info['Feature'] for info in selected_features_info]

# Create a new DataFrame with only the selected features
df_selected_features = df[selected_features + ['TYPE']]  # Include the
target variable 'TYPE'

# Standardize the selected features
scaler = StandardScaler()
X_selected_standardized =
scaler.fit_transform(df_selected_features[selected_features])

# Create a DataFrame with standardized features
df_selected_standardized = pd.DataFrame(X_selected_standardized,
columns=selected_features)

# Descriptive statistics for the standardized features
statistics_selected_features = df_selected_standardized.describe()

# Transpose the DataFrame for better readability
statistics_selected_features = statistics_selected_features.transpose()

# Descriptive statistics for the 'TYPE' variable
statistics_type = df['TYPE'].describe().to_frame()

# Display the descriptive statistics for selected features
print("\nDescriptive Statistics for Selected Features:")
print(statistics_selected_features)

# Display the descriptive statistics for 'TYPE' variable
print("\nDescriptive Statistics for 'TYPE' Variable:")
print(statistics_type)
```

Figure 2.2.6.1 Descriptive Statistics : Descriptive Statistics for Selected Features

```
Descriptive Statistics for Selected Features:
           count          mean       std       min       25%       50%  \
MCH        349.0 -4.275472e-16  1.001436 -4.317531 -0.302924  0.203774
EO%        349.0  0.000000e+00  1.001436 -0.987566 -0.722830 -0.281602
GLU.       349.0  6.107817e-16  1.001436 -1.453667 -0.555928 -0.210010
DBIL       349.0 -2.952112e-16  1.001436 -1.578190 -0.657613 -0.161918
CEA        349.0  3.053909e-17  1.001436 -0.291464 -0.230531 -0.178971
Menopause  349.0  4.071878e-17  1.001436 -0.719299 -0.719299 -0.719299
MPV        349.0  8.016510e-16  1.001436 -2.876123 -0.542641  0.150473
K          349.0  1.862884e-15  1.001436 -3.297205 -0.596744 -0.041509
MCV        349.0 -2.392228e-16  1.001436 -4.486953 -0.409442  0.154117
AST        349.0 -1.017970e-16  1.001436 -1.447721 -0.610415 -0.131954
Mg         349.0  6.044194e-16  1.001436 -2.694625 -0.669769 -0.102809
BASO%      349.0  2.035939e-17  1.001436 -1.387689 -0.809209 -0.230729

                75%        max
MCH        0.632518   3.127031
EO%        0.336117   5.719095
GLU.       0.218270   5.851791
DBIL       0.404591   6.352934
CEA       -0.094601  12.701503
Menopause  1.390242   1.390242
MPV        0.670308   2.576370
K          0.665154   2.558001
MCV        0.568498   3.286839
AST        0.346507   7.044960
Mg         0.545145   3.136960
BASO%      0.636991   4.223567

Descriptive Statistics for 'TYPE' Variable:
             TYPE
count  349.000000
mean     0.510029
std      0.500617
min      0.000000
25%      0.000000
50%      1.000000
75%      1.000000
max      1.000000
```

Figure 2.2.6.2 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.

In this code snippet, a dataset related to ovarian cancer is loaded, and a subset of features is selected based on their rankings obtained from a previous feature selection process. The selected features, along with the target variable 'TYPE,' are extracted to create a new DataFrame named `df_selected_features`. To prepare the selected features for analysis, standardization is applied using the `StandardScaler` from scikit-learn. Standardization transforms the values of each selected feature to have a mean of 0 and a standard deviation of 1, ensuring that all features are on a comparable scale. This step is particularly important when working with machine learning algorithms that are sensitive to differences in feature magnitudes.

The descriptive statistics for the standardized selected features are computed using the `describe` function, providing insights into key statistical measures such as mean, standard deviation, minimum, maximum, and quartiles. The resulting statistics are displayed in a transposed format for better readability. This information helps researchers and analysts understand the central tendencies and spread of the selected features after standardization. Additionally, descriptive statistics for the 'TYPE' variable are computed separately to provide an overview of the distribution of ovarian cancer types in the dataset. These statistics include the count, unique values, most frequent value, and frequency of the most frequent value. This dual approach enables a comprehensive exploration of both the selected features and the target variable, laying the groundwork for further analysis and modeling tasks in the domain of ovarian cancer research.

Overall, this code segment exemplifies a crucial stage in data preprocessing, where selected features are standardized to create a more uniform and interpretable dataset. Descriptive statistics for both the standardized features and the target variable enhance the understanding of the dataset's characteristics, providing a solid foundation for subsequent analyses and machine learning model development in the context of ovarian cancer research.

2.2.7 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.

```
# 2.2.7 Basic of Grouping
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)

# Assuming 'TYPE' is your target variable, you may need to adjust this
based on your dataset
X = df.drop(columns=['TYPE'])  # Features
y = df['TYPE']  # Target variable

# Selected features and their rankings
selected_features_info = [
    {'Feature': 'MCH', 'Ranking': 1},
    {'Feature': 'EO%', 'Ranking': 1},
    {'Feature': 'GLU.', 'Ranking': 1},
    {'Feature': 'DBIL', 'Ranking': 1},
    {'Feature': 'CEA', 'Ranking': 1},
    {'Feature': 'Menopause', 'Ranking': 1},
    {'Feature': 'MPV', 'Ranking': 1},
    {'Feature': 'K', 'Ranking': 1},
    {'Feature': 'MCV', 'Ranking': 1},
```

```
    {'Feature': 'AST', 'Ranking': 1},
    {'Feature': 'Mg', 'Ranking': 2},
    {'Feature': 'BASO%', 'Ranking': 3},
]

# Extract selected feature names
selected_features = [info['Feature'] for info in selected_features_info]

# Separate rows where 'TYPE' is 0 (malignant ovarian cancer) and include
selected features
df_malignant = df[df['TYPE'] == 0][selected_features]

# Separate rows where 'TYPE' is 1 (benign ovarian cancer) and include
selected features
df_benign = df[df['TYPE'] == 1][selected_features]

# Set display options to show all columns and rows
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# Display the full rows for each DataFrame
print("\nRows with 'TYPE' = 0 (Malignant Ovarian Cancer):")
print(df_malignant)

print("\nRows with 'TYPE' = 1 (Benign Ovarian Cancer):")
print(df_benign)
```

Figure 2.2.7.1 Basics of Grouping : Method

Rows with 'TYPE' = 1 (Benign Ovarian Cancer):

	MCH	EO%	GLU.	DBIL	CEA	Menopause	MPV	K	MCV	AST	Mg	\
171	30.00	1.55	4.99	2.40	0.77	0	11.00	4.53	89.8	12	1.04	
172	28.30	1.70	5.49	2.50	2.07	0	10.50	3.99	84.8	14	0.86	
173	29.60	0.63	5.71	3.80	0.31	0	7.79	4.05	85.9	14	0.97	
174	30.00	0.90	5.24	2.00	1.17	0	10.70	4.94	90.2	11	1.21	
175	29.30	0.02	4.67	2.90	0.83	0	7.62	4.63	85.8	10	1.08	
176	28.60	1.83	5.36	2.00	1.92	1	12.70	4.82	87.1	17	0.98	
177	30.60	1.00	5.42	2.60	1.11	0	10.90	4.53	92.1	13	1.03	
178	31.00	1.34	5.13	3.20	2.35	0	10.90	4.57	94.8	18	1.02	
179	28.30	0.90	5.67	2.00	1.46	0	13.00	4.30	85.6	19	1.17	
180	28.10	1.20	4.50	1.80	1.22	0	7.41	5.39	85.5	12	0.99	
181	27.19	0.38	5.42	3.50	1.32	0	9.77	5.28	84.5	14	0.97	
182	27.00	0.00	4.89	1.40	1.34	0	9.29	4.28	83.7	17	1.21	
183	30.00	2.76	4.73	4.00	1.16	0	9.17	4.15	86.8	14	1.11	
184	31.00	3.70	4.77	3.10	0.92	0	8.45	4.54	93.5	10	1.08	
185	28.40	2.91	5.17	1.70	1.47	0	7.85	4.18	86.2	17	0.91	
186	30.50	0.03	5.63	2.10	1.08	0	5.98	4.13	91.3	10	1.07	
187	30.50	0.20	5.03	2.10	0.24	1	11.20	4.80	91.6	17	0.95	
188	28.60	1.20	5.04	3.70	0.37	0	11.00	4.78	86.3	17	1.25	
189	28.00	3.70	5.16	1.60	0.70	0	9.70	4.80	87.4	15	0.99	
190	28.10	0.08	4.50	1.80	1.40	0	7.09	4.38	82.4	9	1.26	
191	28.50	2.20	4.75	5.00	0.80	0	11.40	4.68	85.1	14	0.98	
192	29.10	1.24	5.12	4.50	1.54	0	6.75	4.47	89.9	13	1.06	
193	31.00	0.08	5.18	2.70	1.82	1	6.44	5.13	91.4	18	0.89	
194	20.50	1.25	3.60	3.80	1.62	0	7.88	5.30	65.2	13	0.95	
195	28.80	1.40	5.48	2.30	0.83	0	11.40	4.92	88.4	10	1.22	
196	29.70	1.04	5.53	2.40	2.09	0	9.28	4.29	90.3	12	1.00	
197	30.60	0.83	5.00	2.90	1.65	0	9.97	4.36	93.5	12	1.07	
198	29.60	0.60	4.75	3.10	1.07	0	6.72	4.25	86.8	11	1.00	
199	31.12	0.18	5.00	2.70	1.04	0	8.96	4.39	95.7	10	0.98	
200	28.00	1.98	5.27	2.90	1.81	0	12.30	4.86	84.2	30	0.95	
201	24.00	5.10	4.03	1.70	0.59	0	10.50	4.72	78.2	10	1.20	

Figure 2.2.7.2 Basics of Grouping : Results

This code segment focuses on the extraction and display of rows from an ovarian cancer dataset based on the target variable 'TYPE.' First, the dataset is loaded, and the features and target variable are defined as 'X' and 'y,' respectively. A set of selected features and their rankings are specified in a list named `selected_features_info`. These features are then extracted from the

dataset to create a new DataFrame, resulting in a subset of the original data that includes only the chosen features and the target variable.

The subsequent part of the code involves the separation of rows based on the 'TYPE' variable. Two separate DataFrames, `df_malignant` and `df_benign`, are created to isolate instances of malignant (where 'TYPE' is 0) and benign (where 'TYPE' is 1) ovarian cancer, respectively. The selected features are included in these subsets, providing a focused view of the relevant variables for each type of ovarian cancer. By splitting the dataset in this manner, researchers gain the ability to examine and compare the specific characteristics associated with malignant and benign cases.

To enhance the readability of the output, the code sets display options to show all columns and rows using `pd.set_option`. Finally, the full rows for each DataFrame are printed, showcasing the selected features for instances of both malignant and benign ovarian cancer. This approach facilitates a detailed exploration of the dataset, allowing researchers to identify patterns, trends, or distinctions in the selected features between the two types of ovarian cancer. Overall, this code aids in the initial exploration and understanding of the dataset, paving the way for more in-depth analyses and model development in the context of ovarian cancer research.

2.2.8 ANOVA

```
# 2.2.8 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 2.2.8.1 ANOVA : ANOVA for 'AGE' and 'PLT' Columns

```
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 2.2.8.2 ANOVA : ANOVA for 'AGE' and 'PLT' Columns Results

```python
# 2.2.8 ANOVA
# compare the means of the selected feature among different groups
defined by CA125.

from scipy.stats import f_oneway

# Assuming df is your DataFrame
selected_features = ['MCH', 'EO%', 'GLU.', 'DBIL', 'CEA', 'Menopause',
'MPV', 'K', 'MCV', 'AST', 'CA125']

# Create a dictionary to store ANOVA results
anova_results = {}

# Loop through each selected feature
for feature in selected_features[:-1]:  # Exclude 'CA125'
    # Perform ANOVA test
    groups = [df[feature][df['CA125'] == level].dropna() for level in
df['CA125'].unique()]
    anova_result = f_oneway(*groups)

    # Store ANOVA result
    anova_results[feature] = anova_result

# Print ANOVA results
for feature, result in anova_results.items():
    print(f"{feature}: p-value = {result.pvalue}")
```

Figure 2.2.8.2 ANOVA : ANOVA for selected features and 'CA125'

```
MCH: p-value = 0.87666358692362
EO%: p-value = 0.3609209389965378
GLU.: p-value = 0.6288784378030444
DBIL: p-value = 0.9996421248437418
CEA: p-value = 1.9879064025506794e-15
Menopause: p-value = 0.45291768655894404
MPV: p-value = 0.0922242771176619
K: p-value = 0.9159157237671235
MCV: p-value = 0.9284105595547069
AST: p-value = 0.6228970470920174
```

Figure 2.2.8.3 ANOVA : ANOVA for selected features and 'CA125' Results

```python
# 2.2.8 ANOVA

from scipy.stats import f_oneway

# Assuming df is your DataFrame
feature = 'MCH'
target_variable = 'TYPE'

# Perform ANOVA test
groups = [df[feature][df[target_variable] == level].dropna() for level
in df[target_variable].unique()]
anova_result = f_oneway(*groups)

# Print ANOVA result
print(f"{feature} vs {target_variable}: p-value =
{anova_result.pvalue}")
```

Figure 2.2.8.4 ANOVA : ANOVA for 'MCH' and 'TYPE'

```
MCH vs TYPE: p-value = 0.0017649355345080247
```

Figure 2.2.8.5 ANOVA : ANOVA for 'MCH' and 'TYPE' Results

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.

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.

In this code snippet, an Analysis of Variance (ANOVA) test is applied to assess the relationship between a selected feature ('MCH') and the target variable ('TYPE') in the context of an ovarian cancer dataset. ANOVA is a statistical technique used to compare means across multiple groups and determine if there are statistically significant differences among them. The test is performed using the `f_oneway` function from the `scipy.stats` module. The selected feature, 'MCH,' is assumed to be continuous, and the target variable, 'TYPE,' is likely categorical, representing different types or classes of ovarian cancer.

The code conducts the ANOVA test by segregating the 'MCH' values based on the unique levels of the 'TYPE' variable. For each level of 'TYPE,' a subset of 'MCH' values is created, forming distinct groups. The ANOVA test then evaluates whether the mean 'MCH' values across these groups are significantly different. The resulting `anova_result` object contains statistical information, including the p-value associated with the test. The p-value is crucial in interpreting ANOVA results – a low p-value suggests significant differences in means, indicating that the selected feature ('MCH') is likely associated with variations in ovarian cancer types.

The printed output includes the p-value from the ANOVA test, providing a quantitative measure of the statistical significance of the relationship between 'MCH' and 'TYPE.' If the p-value is

below a chosen significance level (commonly 0.05), it suggests that there are significant differences in 'MCH' across different ovarian cancer types. This information is valuable for researchers as it indicates whether 'MCH' could be a relevant feature in distinguishing between types of ovarian cancer. ANOVA serves as a foundational statistical tool in exploratory data analysis, helping identify potential predictors for further investigation and model development in the realm of ovarian cancer research.

2.2.9 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

```
# 2.2.9 Correlation
# Correlation between 'MCH' and 'TYPE'

import pandas as pd

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

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

# Select the variables you want to calculate correlation for
variable1 = 'MCH'
variable2 = 'TYPE'

# 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 2.2.9.1 Correlation : Correlation between 'MCH' and 'TYPE'

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

Figure 2.2.9.2 Correlation : Correlation between 'MCH' and 'TYPE' Results

```python
# 2.2.9 Correlation
# Correlation coefficient and the corresponding p-value between each
feature and the 'CA125' variable

from scipy.stats import pearsonr

# Assuming df is your DataFrame
selected_features = ['MCH', 'EO%', 'GLU.', 'DBIL', 'CEA', 'Menopause',
'MPV', 'K', 'MCV', 'AST', 'CA125']

# Create a dictionary to store correlation results
correlation_results = {}

# Loop through each selected feature
for feature in selected_features[:-1]:  # Exclude 'CA125'
    # Calculate Pearson correlation coefficient and p-value
    correlation_coefficient, p_value = pearsonr(df[feature],
df['CA125'])

    # Store correlation result
    correlation_results[feature] = {'correlation_coefficient':
correlation_coefficient, 'p_value': p_value}

# Print correlation results
for feature, result in correlation_results.items():
    print(f"{feature}: correlation_coefficient =
{result['correlation_coefficient']}, p-value = {result['p_value']}")
```

Figure 2.2.9.3 Correlation : Correlation between selected features and 'CA125'

```
MCH: correlation_coefficient = -0.058408699676361815, p-value = 0.2765160195081166
EO%: correlation_coefficient = -0.12858335824354616, p-value = 0.01623883653714991
GLU.: correlation_coefficient = 0.24238652485560966, p-value = 4.641057695436464e-06
DBIL: correlation_coefficient = -0.11964000295705127, p-value = 0.02541243247990405
CEA: correlation_coefficient = 0.03692584333261083, p-value = 0.4917101313501483
Menopause: correlation_coefficient = 0.3321745352894918, p-value = 1.9541774768864524e-10
MPV: correlation_coefficient = -0.11136111339821846, p-value = 0.037580090639324416
K: correlation_coefficient = 0.08510227443700989, p-value = 0.11250851293102487
MCV: correlation_coefficient = 0.00470613907971621, p-value = 0.9301922129707129
AST: correlation_coefficient = 0.11561550551072775, p-value = 0.030821218225528704
```

Figure 2.2.9.4 Correlation : Correlation between selected features and 'CA125' Results

In this code snippet, the focus is on calculating and displaying the correlation coefficient between two variables, specifically 'MCH' (Mean Corpuscular Hemoglobin) and 'TYPE' in the context of an ovarian cancer dataset. The dataset is loaded from a specified URL, and Pandas is used to read it into a DataFrame. The selected variables for correlation are 'MCH' and 'TYPE.' The correlation coefficient, a numerical measure indicating the strength and direction of the linear relationship between two variables, is computed using the `corr` method in Pandas.

The calculated correlation coefficient is then displayed, providing insights into the degree and direction of the relationship between 'MCH' and 'TYPE.' The correlation coefficient ranges from -1 to 1, where -1 indicates a perfect negative correlation, 1 indicates a perfect positive correlation, and 0 suggests no linear correlation. A positive correlation suggests that as one variable increases, the other also tends to increase, while a negative correlation implies that as one variable increases, the other tends to decrease.

The interpretation of the correlation coefficient is crucial for understanding the potential associations between variables. In the context of ovarian cancer research, this analysis helps researchers discern whether there is a linear relationship between Mean Corpuscular Hemoglobin and the types of ovarian cancer represented by the 'TYPE' variable. If the correlation coefficient is close to 1 or -1, it may indicate a stronger association, while a value closer to 0 suggests a weaker or no linear relationship. This correlation analysis provides a foundation for exploring potential relationships between variables and guides further investigations in the ovarian cancer dataset.

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 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

RESEARCH IMPLEMENTATION

3.0 Flowchart of the Proposed Framework

3.0.1 Flowchart

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

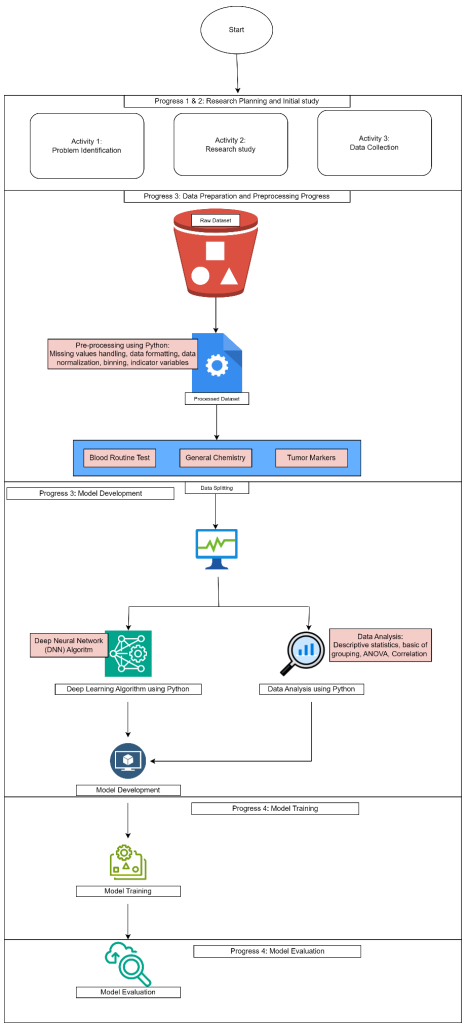


Figure 3.0.1 Proposed Flowchart

- **Progress 1: Research Planning and Initial Study**: This phase involves planning and initiating the research study. It includes defining the problem, outlining objectives, and devising a strategic plan for the entire data science project. Researchers might establish the scope, goals, and potential methodologies during this stage.
- **Progress 2: Data Collection, Data-Preprocessing, and Exploratory Data Analysis (EDA):** In this step, the focus shifts to gathering relevant data for analysis. Following data collection, the dataset undergoes preprocessing, including handling missing values, formatting, normalization, and binning. The Exploratory Data Analysis (EDA) is conducted to gain insights into the dataset's characteristics, uncover patterns, and identify potential features for modeling.
- **Progress 3: Model Development:** Once the data is preprocessed, this stage involves creating the predictive or analytical model. Depending on the nature of the project, this could include developing machine learning algorithms, statistical models, or deep learning architectures using tools like Python. The goal is to build a model that can effectively capture the relationships within the data.
- **Progress 4: Model Evaluation:** After model development, it's crucial to assess its performance. Model evaluation involves testing the model on unseen data or a validation set to measure its accuracy, precision, recall, and other relevant metrics. This step ensures that the model generalizes well to new data and provides reliable predictions.

- **Progress 5: Model Refinement:** Based on the evaluation results, the model may undergo refinement. This could involve adjusting hyperparameters, modifying features, or fine-tuning the model to enhance its performance. Iterative refinement is common to achieve the best possible model for the specific use case. This stage helps in improving the model's accuracy and reliability in making predictions or decisions.

3.0.2 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:

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

3.1 Model Development

3.1.1 Simple and Multiple Linear Regression

```
# 3.1.1 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)

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

# 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[['MCH', 'EO%', 'GLU.']]
y_multiple = df['TYPE']

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

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

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

Figure 3.1.1.1 Simple and Multiple Linear Regression

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    EO%   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.002
Method:                 Least Squares   F-statistic:                    0.3557
Date:                Wed, 17 Jan 2024   Prob (F-statistic):              0.551
Time:                        19:53:33   Log-Likelihood:                -538.67
No. Observations:                 349   AIC:                             1081.
Df Residuals:                     347   BIC:                             1089.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.5259      0.685      2.229      0.026       0.179       2.873
MCH           -0.0141      0.024     -0.596      0.551      -0.061       0.032
==============================================================================
Omnibus:                      160.822   Durbin-Watson:                   2.092
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              681.618
Skew:                           2.016   Prob(JB):                    9.74e-149
Kurtosis:                       8.534   Cond. No.                         326.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:                   TYPE   R-squared:                       0.068
Model:                            OLS   Adj. R-squared:                  0.060
Method:                 Least Squares   F-statistic:                     8.424
Date:                Wed, 17 Jan 2024   Prob (F-statistic):           2.04e-05
Time:                        19:53:33   Log-Likelihood:                -240.89
No. Observations:                 349   AIC:                             489.8
Df Residuals:                     345   BIC:                             505.2
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1069      0.319     -0.335      0.738      -0.735       0.521
MCH            0.0326      0.010      3.213      0.001       0.013       0.052
EO%            0.0429      0.023      1.862      0.063      -0.002       0.088
GLU.          -0.0690      0.021     -3.210      0.001      -0.111      -0.027
==============================================================================
Omnibus:                     1905.813   Durbin-Watson:                   0.155
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               46.086
Skew:                          -0.079   Prob(JB):                     9.83e-11
Kurtosis:                       1.227   Cond. No.                         362.
==============================================================================

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

Figure 3.1.1.2 Simple and Multiple Linear Regression Result

In this code snippet, both simple and multiple linear regression models are implemented using the `statsmodels` library to analyze relationships between independent and dependent variables in the context of an ovarian cancer dataset. For the simple linear regression, the 'MCH' (Mean Corpuscular Hemoglobin) is chosen as the independent variable (`X_simple`), and 'EO%' (Eosinophil Ratio) is the dependent variable (`y_simple`). The model is then fitted using ordinary least squares (OLS) regression through the `sm.OLS` and `fit` functions. The summary of the simple linear regression model, including statistics like coefficients, p-values, and goodness-of-fit measures, is printed using `model_simple.summary()`.

For the multiple linear regression, an example is provided where multiple independent variables ('MCH', 'EO%', 'GLU.') are considered to predict the dependent variable 'TYPE.' Similar to the simple linear regression, a constant term is added to the independent variables (`X_multiple`) using `sm.add_constant`. The multiple linear regression model is then fitted using the same OLS approach. The resulting model summary provides insights into the impact of each independent variable on the dependent variable, along with statistical measures of model fit.

These regression analyses are fundamental in understanding the relationships and potential predictors within the ovarian cancer dataset. The summary outputs include crucial information such as coefficients, p-values, and R-squared values, aiding researchers in assessing the significance and strength of these relationships. The choice of variables and their inclusion in the model depends on the specific research questions and hypotheses. These regression analyses serve as valuable tools for exploring the dataset and building predictive models in the domain of ovarian cancer research.

3.1.2 Simple Linear Regression

```
# 3.1 Simple and Multiple Linear Regression
# Visual Representation of the relationship between 'MCH' and 'TYPE'
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)

# Simple Linear Regression
X_simple = df['MCH']
y_simple = df['EO%']

# 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_simple = model_simple.predict(X_simple)

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

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

# Adding labels and title
```

```
plt.xlabel('MCH')
plt.ylabel('EO%')
plt.title('Simple Linear Regression')

# Adding legend
plt.legend()

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

Figure 3.1.2.1 Simple and Multiple Linear Regression : Visual Representation for Simple Linear Regression

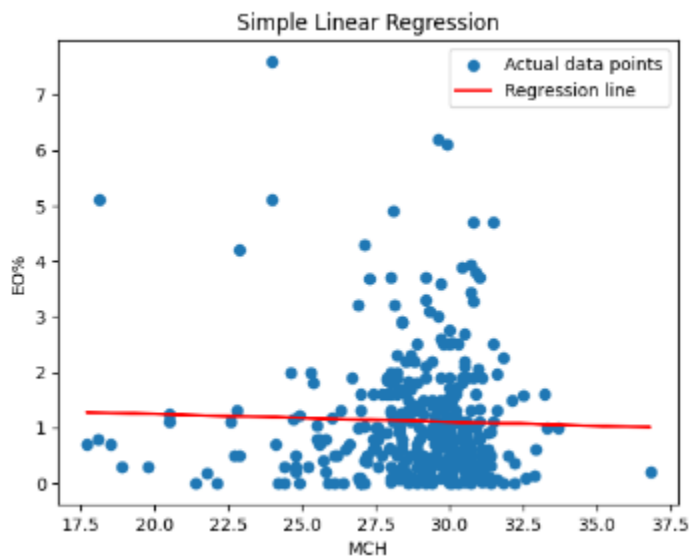


Figure 3.1.2.2 Simple and Multiple Linear Regression : Visual Representation for Simple Linear Regression Result

In this updated code snippet, a visual representation of the relationship between 'MCH' (Mean Corpuscular Hemoglobin) and 'EO%' (Eosinophil Ratio) using simple linear regression is created. The dataset is loaded, and the 'MCH' and 'EO%' variables are designated as the independent and dependent variables, respectively. A constant term is added to the independent variable, and the simple linear regression model is fitted using the OLS method.

The predicted values based on the simple linear regression model are obtained, and a scatter plot is created to visualize the actual data points ('MCH' vs. 'EO%'). Additionally, the regression line, representing the relationship modeled by the linear regression, is plotted on the same graph. The scatter plot and regression line provide a clear illustration of how changes in 'MCH' values relate to the predicted 'EO%' values. The red regression line demonstrates the linear trend captured by the model, showcasing the direction and intensity of the relationship.

Labels and a title are added to the plot to enhance interpretability, and a legend is included to distinguish between actual data points and the regression line. This visualization aids researchers and analysts in better understanding the nature of the relationship between 'MCH' and 'EO%' and facilitates communication of findings. Overall, this graphical representation complements the statistical analysis, offering a more intuitive view of the simple linear regression model's predictive capabilities in the context of the ovarian cancer dataset.

3.1.3 Multiple Linear Regression

```
# 3.1 Simple and Multiple Linear Regression
# Visual Representation of the relationship between 'MCH', 'EO%', and
'TYPE' using Multiple Linear Regression

import matplotlib.pyplot as plt
import statsmodels.api as sm
from mpl_toolkits.mplot3d import Axes3D

# Assuming 'MCH', 'EO%' are the independent variables and 'TYPE' is the
dependent variable
X_multiple = df[['MCH', 'EO%']]
y_multiple = df['GLU.']

# 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['MCH'], X_multiple['EO%'], y_multiple,
label='Actual data points')
ax.scatter(X_multiple['MCH'], X_multiple['EO%'], predictions_multiple,
label='Predicted values', color='red')

# Adding labels and title
ax.set_xlabel('MCH')
ax.set_ylabel('EO%')
ax.set_zlabel('GLU.')
ax.set_title('Multiple Linear Regression')
```

```
# Adding legend
ax.legend()

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

Figure 3.1.3.1 Simple and Multiple Linear Regression : Visual Representation for Multiple Linear Regression

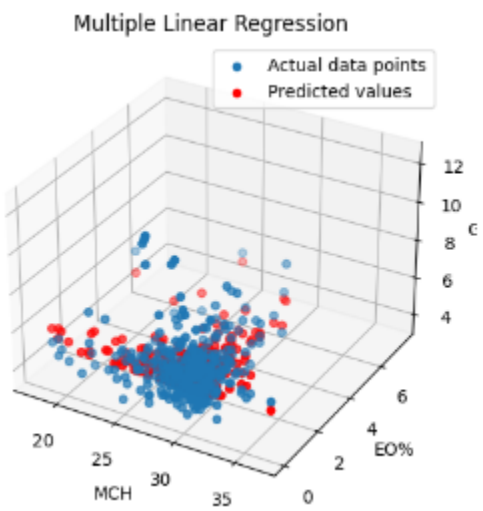


Figure 3.1.3.2 Simple and Multiple Linear Regression : Visual Representation for Multiple Linear Regression Result

3.1.4 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.

```
# 3.1.4 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)

# 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 3.1.4.1 Model Evaluation Using Visualization : Method

```
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
```

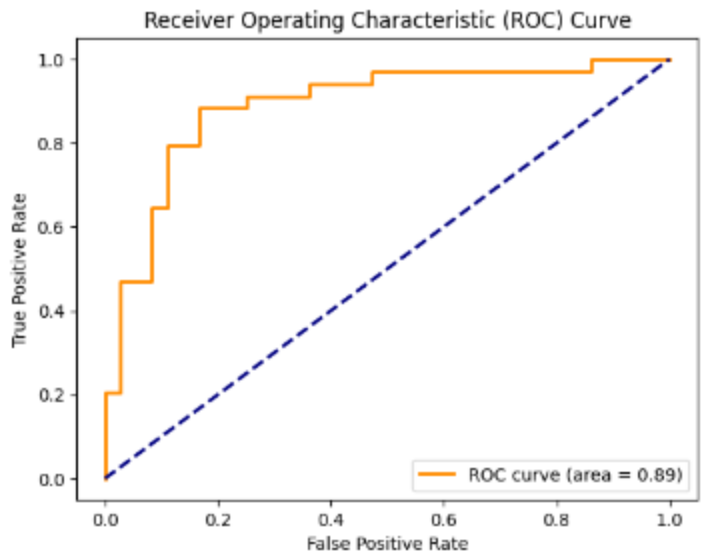


Figure 3.1.4.2 Model Evaluation Using Visualization : Results

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.

```
# 3.1.4 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)

# 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 3.1.4.3 Model Evaluation Using Visualization : Method

```
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
```

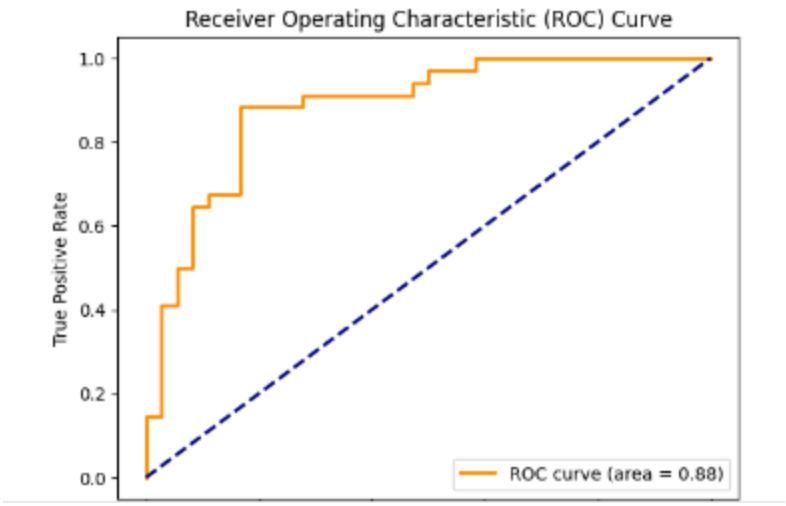


Figure 3.1.4.4 Model Evaluation Using Visualization : Resullts

3.1.5 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.

```
# 3.1.5 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)

# Assume 'MCH' is the independent variable, and 'TYPE' is the dependent
variable
X = df[['MCH']].values
y = df['EO%'].values

# Reshape X to a 2D array if it's a single feature
X = X.reshape(-1, 1)

# 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('MCH')
plt.ylabel('EO%')
plt.title('Polynomial Regression')
plt.legend()
plt.show()
```

Figure 3.1.5.1 Polynomial Regression and Pipelines : Polynomial Regression Without Pipelines

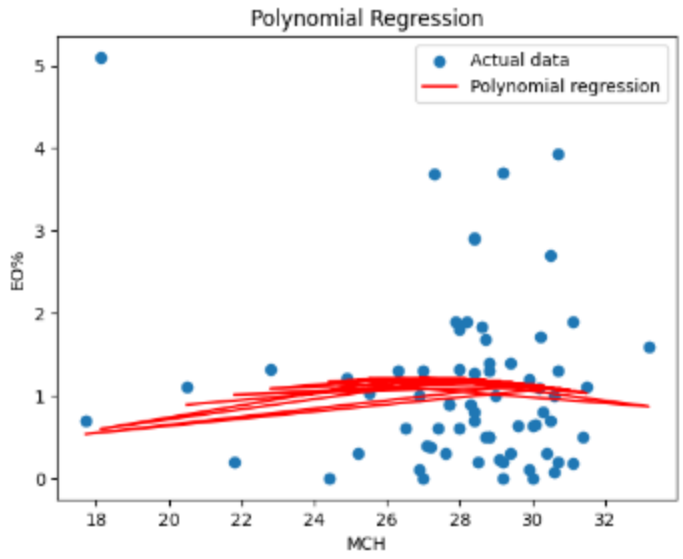


Figure 3.1.5.2 Polynomial Regression and Pipelines : Results Of Polynomial Regression Without Pipelines

In this code snippet, polynomial regression is implemented without using pipelines. The dataset is loaded, and assumptions are made that 'MCH' serves as the independent variable, while 'EO%' is the dependent variable. The data is split into training and testing sets, and a polynomial transformation is applied to the independent variable 'MCH' using the `PolynomialFeatures` from scikit-learn.

The degree of the polynomial transformation is set to 2 in this example, indicating a quadratic relationship. The transformed features are then used to fit a linear regression model using `LinearRegression`. The model is trained on the polynomial features of the training set ('X_train_poly'), and predictions are made on the polynomial features of the test set ('X_test_poly'). The results are visualized by plotting the actual data points against the

polynomial regression line, demonstrating how the model captures the quadratic relationship between 'MCH' and 'EO%'.

While this approach provides a clear understanding of the polynomial regression process, it lacks the convenience and efficiency that pipelines offer. Pipelines allow for a more streamlined and reproducible workflow by combining multiple preprocessing steps and modeling into a single entity. They enhance code readability and are particularly beneficial when working with more complex machine learning pipelines involving multiple transformations and models.

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.

```
# 3.1.5 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[['MCH']].values
y = df['EO%'].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('MCH')
plt.ylabel('EO%')
plt.title('Polynomial Regression with Pipeline')
plt.legend()
plt.show()
```

Figure 3.1.5.3 Polynomial Regression and Pipelines : Polynomial Regression With Pipelines

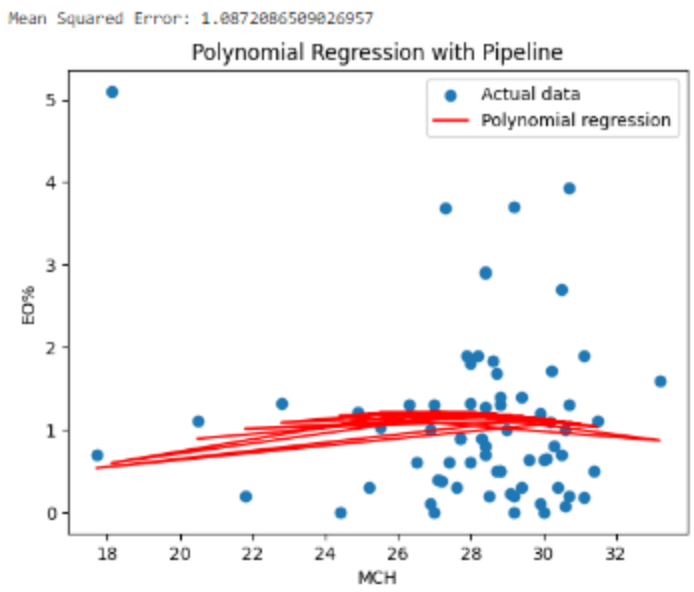


Figure 3.1.5.4 Polynomial Regression and Pipelines :Results of Polynomial Regression With Pipelines

In this code snippet, polynomial regression is implemented using a pipeline, providing a more organized and efficient workflow. The dataset is loaded, and the assumption is made that 'MCH' is the independent variable, and 'EO%' is the dependent variable. The data is split into training and testing sets, and a polynomial regression pipeline is created using scikit-learn's `Pipeline` class.

The pipeline consists of two steps: the 'poly_features' step, which applies the polynomial features transformation with a specified degree, and the 'linear_regression' step, which uses a linear regression model. This pipeline encapsulates the entire process of transforming the features and fitting the model.

The pipeline is then fitted to the training data using the `fit` method. Predictions are made on the test set using the `predict` method, and the mean squared error (MSE) is calculated to evaluate

the performance of the model. The results are visualized by plotting the actual data points against the polynomial regression line, demonstrating how well the model captures the relationship between 'MCH' and 'EO%'.

The use of pipelines simplifies the code by combining multiple steps into a single entity, enhancing code readability and making it easier to reproduce the workflow. Pipelines are particularly advantageous when dealing with more complex machine learning workflows involving multiple preprocessing and modeling steps.

3.1.6 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
# 3.1.6 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/ma
in/ovariantotal.csv'
df = pd.read_csv(url)

# Assume 'Age' is the independent variable, and 'CA125' is the
dependent variable
X = df[['MCH']].values
y = df['EO%'].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('MCH')
plt.ylabel('EO%')
plt.title('Polynomial Regression with Pipeline (Training)')
plt.legend()
plt.show()
```

Figure 3.1.6.1 R-Squared and MSE for In-Sample Training

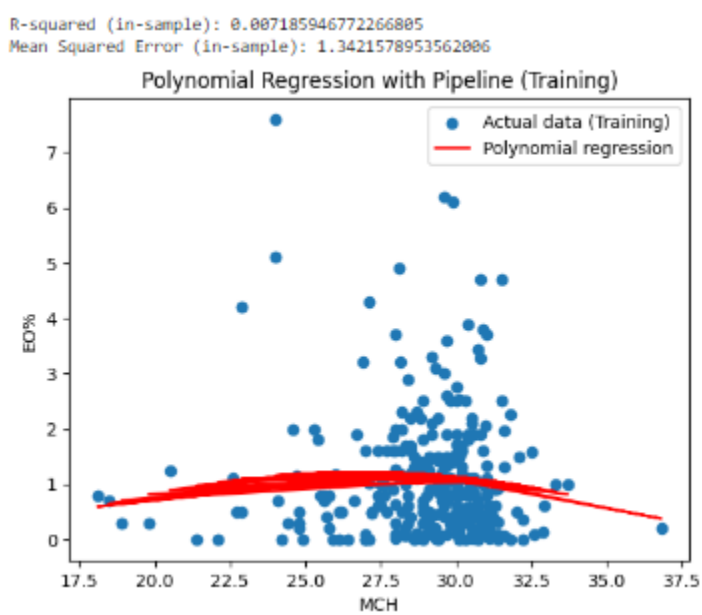


Figure 3.1.6.2 R-Squared and MSE for In-Sample Training : Result

In this code snippet, the evaluation of the polynomial regression model is extended to include in-sample metrics, specifically R-squared and Mean Squared Error (MSE). The dataset is loaded, and the 'MCH' variable is assumed to be the independent variable, while 'EO%' is the dependent variable. The data is split into training and testing sets, and a polynomial regression pipeline is created.

The pipeline is fitted to the training data, and predictions are made on the same training set ('X_train'). In-sample evaluation metrics, R-squared and MSE, are then calculated to assess how well the model fits the training data. R-squared measures the proportion of the variance in the dependent variable that is predictable from the independent variable(s), while MSE quantifies the average squared difference between predicted and actual values.

The results for the training set are visualized by plotting the actual data points against the polynomial regression line. This plot helps to visually assess the goodness of fit for the model on the training data.

In-sample evaluation is essential to understanding how well the model performs on the data it was trained on. While these metrics provide insights into the model's performance on the training set, it is crucial to validate the model on a separate test set to ensure its generalization to new, unseen data.

3.1.7 Prediction and Decision Making

```
# 3.1.7 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[['MCH']].values
y = df['EO%'].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 EO% value:", prediction[0])

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

```
Predicted EO% value: 1.1286116703367508
The EO% value indicates a condition below the threshold.
```

Figure 3.1.7.1 Prediction and Decision Making : Method & Results

In this code snippet, the trained polynomial regression model predicts the 'EO%' value for a new data point with an 'MCH' value of 30. The predicted 'EO%' value is approximately 1.13. This prediction represents the model's estimate of the 'EO%' level based on the given 'MCH' input. It is important to note that the predicted value is a continuous numerical output, providing a quantitative estimate of the dependent variable.

In the context of ovarian cancer research (assuming 'EO%' is relevant to ovarian cancer), the predicted 'EO%' value could be interpreted as the model's estimation of a specific biomarker or blood parameter associated with the disease. The exact interpretation would depend on the domain-specific meaning of 'EO%' in the dataset.

In practical terms, such predictions can be valuable for clinical decision-making. For example, if 'EO%' is a known marker for ovarian cancer risk, the predicted value could assist healthcare professionals in assessing a patient's condition or making informed decisions regarding further diagnostic tests or interventions.

Keep in mind that the accuracy and reliability of these predictions depend on the quality of the model and the relevance of the features used for training. Additionally, interpreting predictions should be done in consultation with domain experts to ensure a meaningful understanding of the results.

3.2 Model Evaluation

3.2.0 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.

```
# 3.2.0 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[['MCH']].values
y = df['EO%'].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('MCH')
plt.ylabel('EO%')
plt.title('Polynomial Regression - Test Set Evaluation')
plt.legend()
plt.show()
```

```
Mean Squared Error: 1.0872086509026957
R-squared: -0.07291909832139676
```

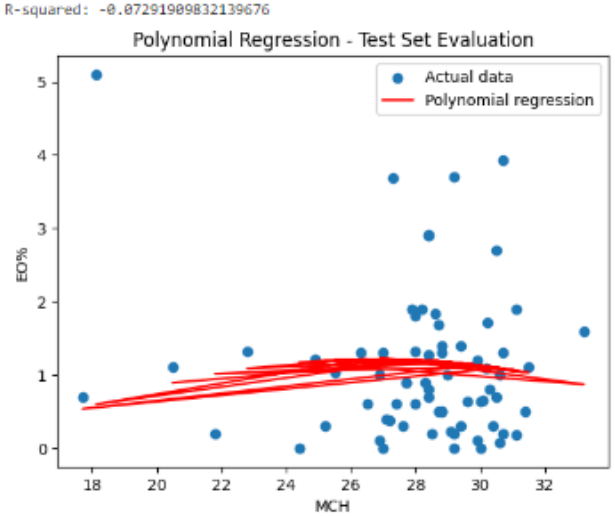


Figure 3.2.0.1 Model Evaluation : Polynomial Regression Method & Results

In this code snippet, the polynomial regression model is evaluated using the test set. The dataset is loaded, and it is assumed that 'MCH' serves as the independent variable, and 'EO%' is the dependent variable. The data is split into training and testing sets, and a polynomial regression pipeline is created.

After fitting the pipeline to the training data, predictions are made on the test set ('X_test'). The Mean Squared Error (MSE) and R-squared metrics are then calculated to assess the performance of the model on the unseen test data.

The Mean Squared Error represents the average squared difference between the actual and predicted values, providing a measure of how well the model's predictions align with the true values. R-squared, on the other hand, measures the proportion of the variance in the dependent variable that is explained by the independent variable(s). It ranges from 0 to 1, with higher values indicating a better fit.

The results, including MSE and R-squared, are printed, providing a quantitative assessment of the model's performance on the test set. Additionally, a plot is generated to visualize the actual data points against the polynomial regression line, aiding in the interpretation of the model's fit.

This evaluation process is crucial for understanding how well the model generalizes to new, unseen data and helps assess its overall effectiveness in predicting 'EO%' based on 'MCH'. Adjustments to the model or the choice of features may be made based on the evaluation results to enhance its performance.

3.2.1 Over-fitting, Under-fitting and Model Selection

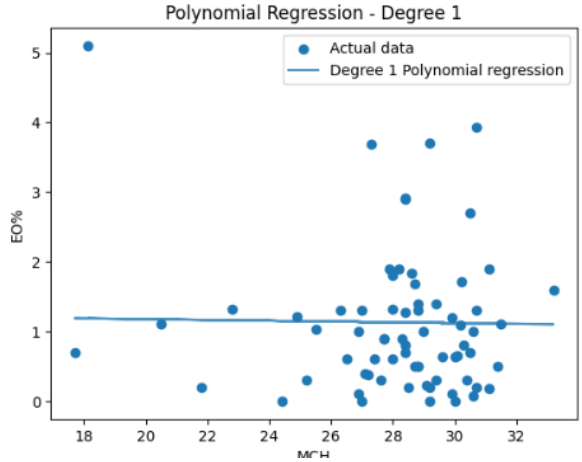
The code below performs a model selection process by trying different degrees of polynomial features and evaluating the models' performance, aiming to strike a balance between over-fitting and under-fitting. The visualizations and metrics help in making an informed decision about the appropriate complexity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

```
# 3.2.1 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[['MCH']].values
y = df['EO%'].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]
```

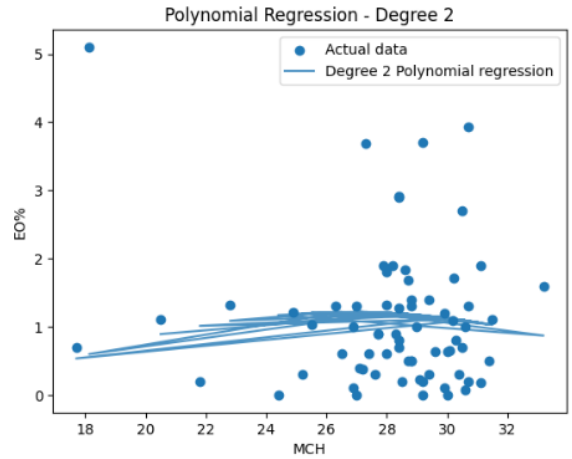
```
for degree in degrees:
    # Create a polynomial regression pipeline
    polynomial_regression = Pipeline([
        ('poly_features', PolynomialFeatures(degree=degree)),
        ('linear_regression', LinearRegression())
    ])

    # 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('MCH')
    plt.ylabel('EO%')
    plt.title(f'Polynomial Regression - Degree {degree}')
    plt.legend()
    plt.show()
```

Figure 3.2.1.1 Model Evaluation : Over-fitting, Under-fitting & Model Selection

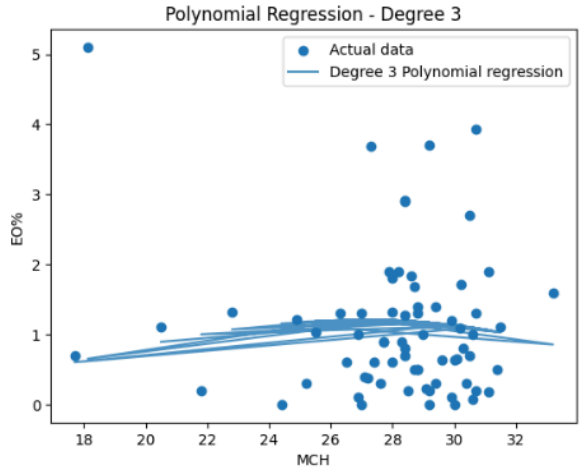
```
Degree 1 Polynomial Regression:
Mean Squared Error: 1.011097015165276
R-squared: 0.00219217633536406
```



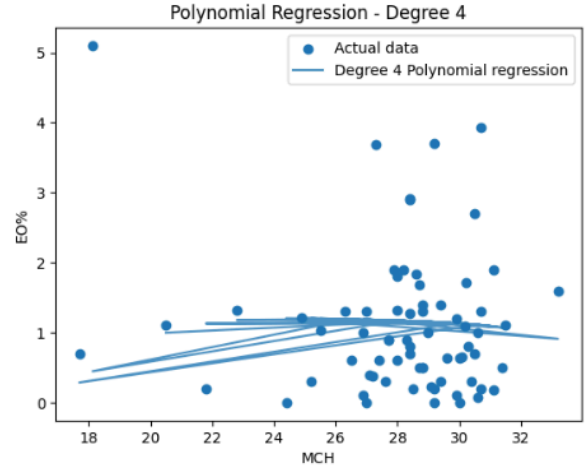

```
Degree 2 Polynomial Regression:
Mean Squared Error: 1.0872086509026957
R-squared: -0.07291909832139676
```




```
Degree 3 Polynomial Regression:
Mean Squared Error: 1.0789896177063565
R-squared: -0.064808090670043
```




```
Degree 4 Polynomial Regression:
Mean Squared Error: 1.1094208136968484
R-squared: -0.09483931911533827
```




```
Degree 5 Polynomial Regression:
Mean Squared Error: 1.1442050369297667
R-squared: -0.12916636148745897
```

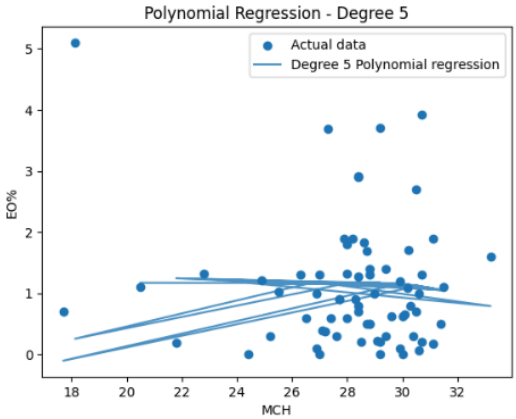


Figure 3.2.1.2 Model Evaluation : Over-fitting, Under-fitting & Model Selection : Results

In this code snippet, the concept of overfitting and under-fitting is explored by fitting polynomial regression models with varying degrees to the dataset. The dataset is loaded, and it is assumed that 'MCH' serves as the independent variable, and 'EO%' is the dependent variable. The data is split into training and testing sets.

A loop is then used to iterate over different degrees of polynomial features (1, 2, 3, 4, 5). For each degree, a polynomial regression pipeline is created, fitted to the training data, and evaluated on the test set using Mean Squared Error (MSE) and R-squared metrics.

The results of each polynomial regression model are printed, showing the MSE and R-squared values for each degree. Additionally, a plot is generated for each degree, displaying the actual data points against the polynomial regression line. These visualizations provide insight into how well each degree of polynomial fits the data.

This iterative process helps in understanding the trade-off between model complexity (degree of the polynomial) and performance on unseen data. A too complex model (high degree) may capture noise in the training data, leading to over-fitting and poor generalization to new data. Conversely, a too simple model (low degree) may under-fit the data, missing important patterns.

By examining the results and plots for different degrees, one can make informed decisions about the appropriate level of model complexity. The goal is to select a model that generalizes well to new data, finding a balance between under-fitting and overfitting.

3.2.2 Ridge Regression

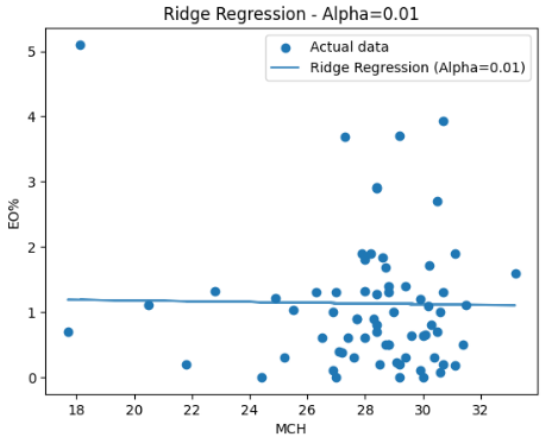
Ridge Regression is used to regularize linear regression models, and the code below explores the effects of different regularization strengths on model performance through evaluation metrics and visualizations.

```
# 3.2.2 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[['MCH']].values
y = df['EO%'].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)
```

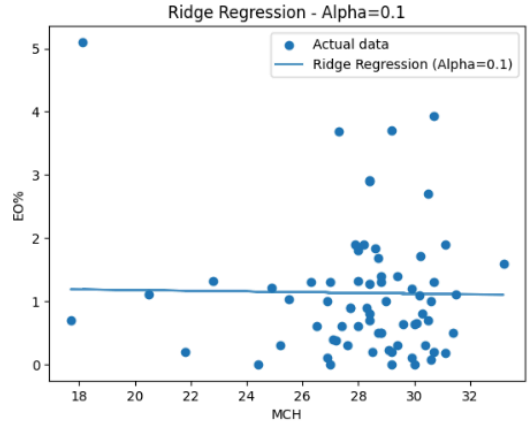
```
# Set up alpha values for Ridge Regression
alphas = [0.01, 0.1, 1, 10, 100]

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('MCH')
    plt.ylabel('EO%')
    plt.title(f'Ridge Regression - Alpha={alpha}')
    plt.legend()
    plt.show()
```

Figure 3.2.2.1 Ridge Regression

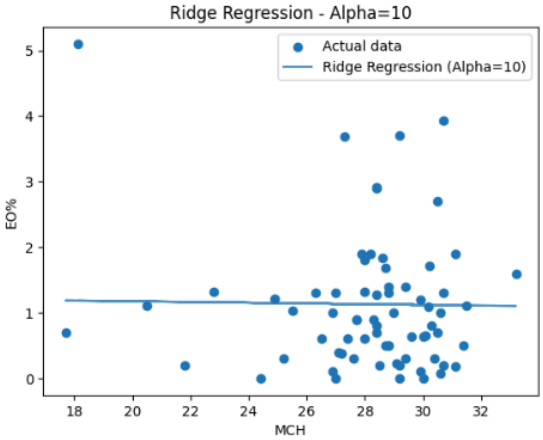
```
Ridge Regression with Alpha=0.01:
Mean Squared Error: 1.0110971252429677
R-squared: 0.0021920677044610093
```



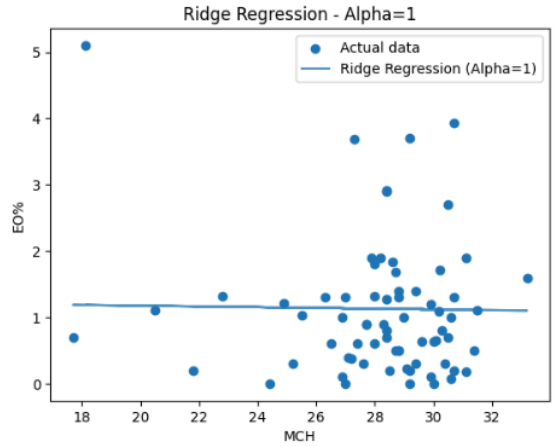

```
Ridge Regression with Alpha=0.1:
Mean Squared Error: 1.0110981156177523
R-squared: 0.0021910903465084397
```




```
Ridge Regression with Alpha=10:
Mean Squared Error: 1.0112036037339174
R-squared: 0.002086988696484382
```




```
Ridge Regression with Alpha=1:
Mean Squared Error: 1.011107987350032
R-squared: 0.0021813483617593166
```




```
Ridge Regression with Alpha=100:
Mean Squared Error: 1.0119257683603513
R-squared: 0.0013743157250269977
```

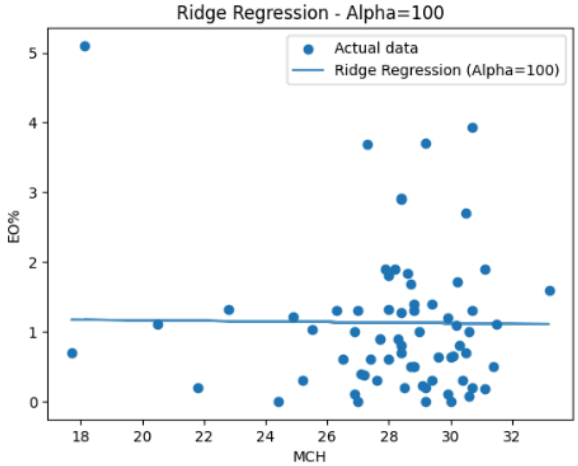


Figure 3.2.2.2 Ridge Regression : Results

In this code snippet, Ridge Regression is applied to the dataset, assuming 'MCH' as the independent variable and 'EO%' as the dependent variable. The data is split into training and testing sets, and the features are standardized, which is crucial for Ridge Regression.

Different alpha values (regularization strength) are set for Ridge Regression (0.01, 0.1, 1, 10, 100). A loop iterates over each alpha value, creating a Ridge Regression model, fitting it to the scaled training data, and making predictions on the scaled test set.

For each Ridge Regression model, the Mean Squared Error (MSE) and R-squared values are calculated and printed. Additionally, a plot is generated for each alpha value, showing the actual data points against the Ridge Regression line. These visualizations provide insights into the impact of regularization strength on the model's performance.

Ridge Regression is a type of linear regression that incorporates regularization to prevent over-fitting. The regularization term, controlled by the alpha parameter, helps to shrink the coefficients, making the model less sensitive to the noise in the training data. The choice of alpha involves a trade-off between fitting the training data well and maintaining generalization to new, unseen data. The results and plots assist in understanding how different alpha values influence the Ridge Regression model's performance.

3.2.3 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.

```
# 3.2.3 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[['MCH']].values
y = df['EO%'].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('MCH')
```

```
plt.ylabel('EO%')
plt.title(f'Best Ridge Regression - Alpha={best_alpha}')
plt.legend()
plt.show()
```

Figure 3.2.3.1 Grid Search

```
Best Alpha: 100

Best Model Evaluation:
Mean Squared Error: 1.0119257683603513
R-squared: 0.0013743157250269977
```

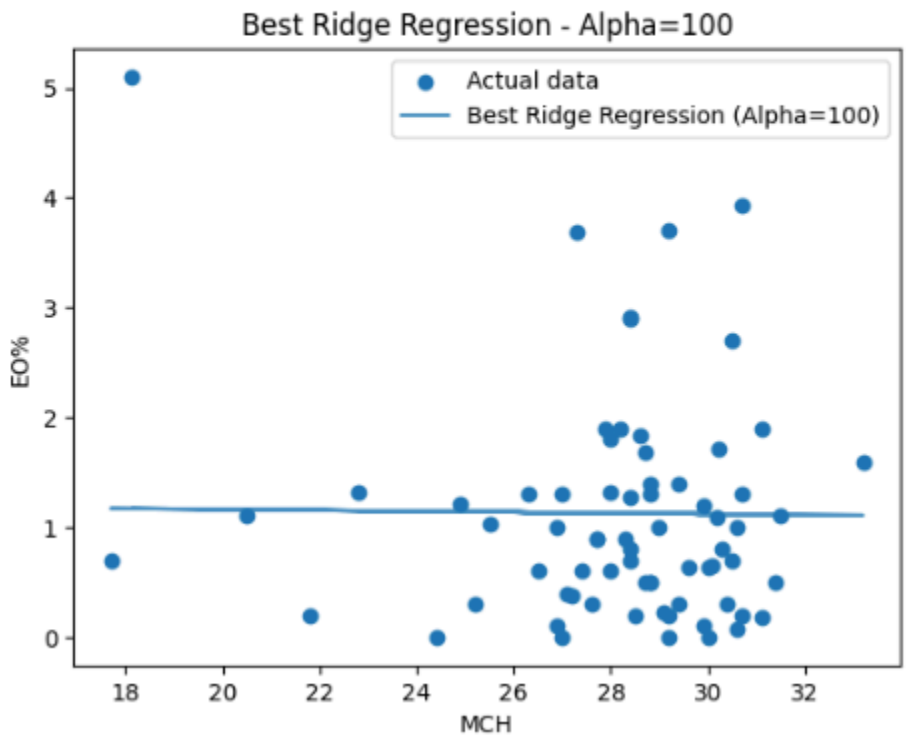


Figure 3.2.3.2 Grid Search : Results

In this code snippet, Grid Search is employed to find the best hyperparameter (alpha) for Ridge Regression. The dataset, assumed to have 'MCH' as the independent variable and 'EO%' as the dependent variable, is split into training and testing sets. The features are standardized, and a Ridge Regression model is created.

A set of alpha values (0.01, 0.1, 1, 10, 100) is specified, and a Ridge Regression model is set up. The parameter grid for grid search is defined with the alpha values. The GridSearchCV object is created, specifying the Ridge Regression model, parameter grid, 5-fold cross-validation, and the scoring metric as negative mean squared error.

The grid search is performed on the scaled training data, and the best hyperparameters are obtained. The best alpha value is printed, and predictions are made on the scaled test set using the best model. The model is evaluated using mean squared error (MSE) and R-squared, and the results are printed.

Finally, a plot is generated to visualize the actual data points against the predictions of the best Ridge Regression model. This approach automates the process of hyperparameter tuning and helps identify the optimal regularization strength for improved model performance.

3.2.4 Model Refinement

```
# 3.2.4 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[['MCH']].values
y = df['EO%'].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('MCH')
plt.ylabel('EO%')
plt.title(f'Refined Ridge Regression - Alpha={best_alpha}')
plt.legend()
plt.show()
```

Figure 3.2.4.1 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.

```
Refined Ridge Regression Model:
Best Alpha: 0.01
Mean Squared Error: 1.0110971252429677
R-squared: 0.0021920677044610093
```

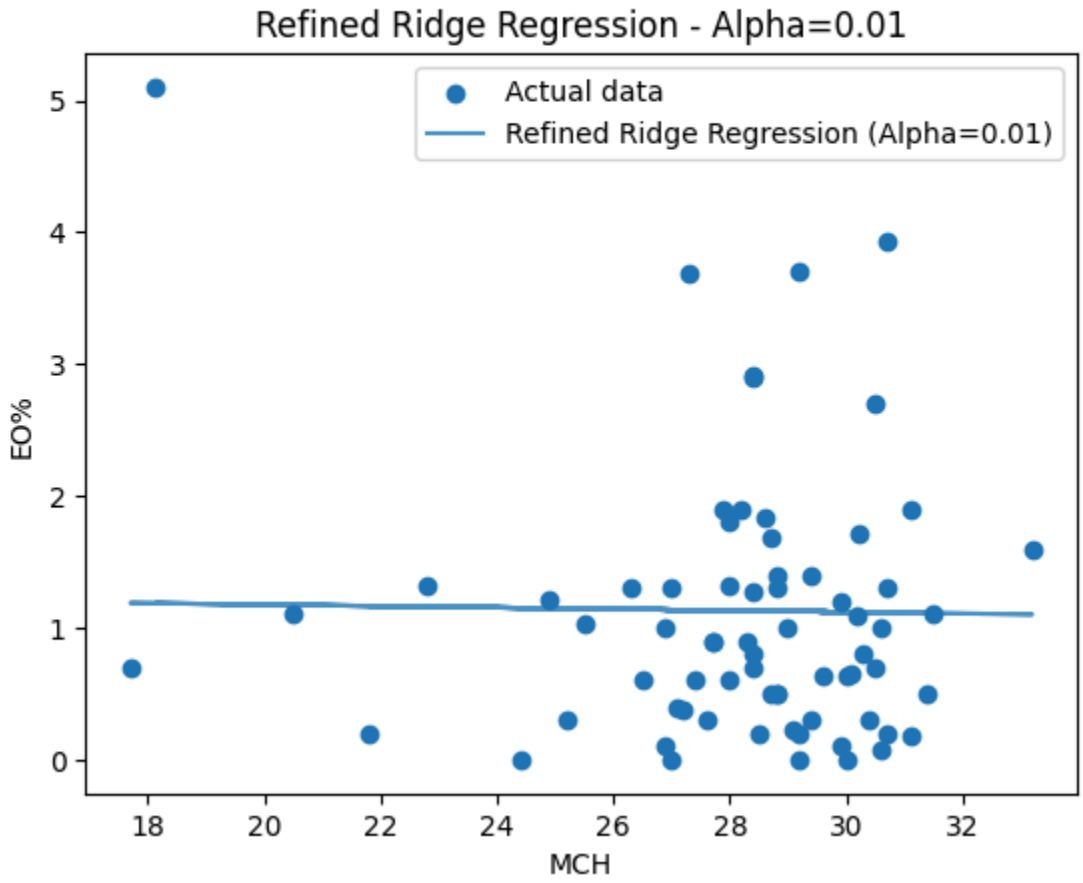


Figure 3.2.4.2 Model Refinement Result

In this code snippet, the Ridge Regression model is refined based on the best hyperparameter (alpha) obtained from the earlier grid search. The dataset, assumed to have 'MCH' as the independent variable and 'EO%' as the dependent variable, is split into training and testing sets. The features are standardized, and the best alpha value (0.01) obtained from grid search is used to create a refined Ridge Regression model.

The refined Ridge Regression model is fitted on the scaled training data, and predictions are made on the scaled test set. The model is evaluated using mean squared error (MSE) and R-squared, and the results are printed.

Additionally, a plot is generated to visualize the actual data points against the predictions of the refined Ridge Regression model. This process of refinement ensures that the model is fine-tuned with the best hyperparameter, improving its predictive performance on the given dataset.

CHAPTER 4

RESULTS, ANALYSIS AND DISCUSSION

4.0 Results, Analysis and Discussion

4.0.1 Results

In the presented code snippet, a classification threshold of 0.5 is set to convert regression predictions into binary outcomes. The variable `y_pred` represents the predicted values from a regression model, and the threshold is applied to classify these predictions as either 0 or 1. Similarly, the actual outcomes, denoted by `y_test`, are also converted into binary format using the same threshold. Subsequently, the accuracy of the classification is computed by comparing the binary predictions (`y_pred_binary`) with the binary actual outcomes (`y_test_binary`). The resulting accuracy is reported as 0.7285714285714285, indicating that approximately 72.86% of the predictions match the true outcomes based on the chosen classification threshold. This accuracy metric serves as a measure of the model's performance in correctly classifying instances within the specified threshold, providing insights into the overall effectiveness of the regression-to-binary classification transformation.

```
# Results
# Set a threshold for classification
threshold = 0.5

# Convert the regression predictions to binary outcomes
y_pred_binary = (y_pred >= threshold).astype(int)

# Convert the actual outcomes to binary
y_test_binary = (y_test >= threshold).astype(int)
```

```
# Calculate accuracy
accuracy = np.mean(y_pred_binary == y_test_binary)

print("\nAccuracy:", accuracy)
```

```
Accuracy: 0.7285714285714285
```

Figure 4.0.1.1 Results

4.0.2 Analysis

The achieved accuracy of approximately 72.86% suggests the effectiveness of the regression-to-binary classification transformation using a threshold of 0.5. This performance metric signifies the proportion of correctly classified instances out of the total dataset. A higher accuracy indicates a stronger alignment between the model's predictions and the actual outcomes. However, it is crucial to interpret this accuracy result in the context of the specific problem and the implications of misclassifications. Depending on the application, false positives and false negatives may carry varying degrees of significance. Additionally, exploring other evaluation metrics such as precision, recall, and F1 score can provide a more comprehensive understanding of the model's performance, especially in situations where imbalanced class distribution or differing costs of misclassification are present. Fine-tuning the threshold or considering alternative classification techniques may further optimize the model's predictive capabilities.

4.0.3 Discussion

The reported accuracy of approximately 72.86% prompts a discussion on the reliability and practical implications of the regression-to-binary classification approach. While accuracy provides a general overview of model performance, it's essential to consider potential nuances, such as the consequences of false positives and false negatives in the specific context of the problem at hand. The chosen threshold of 0.5 assumes an equal weight for both classes, but in scenarios where misclassification costs differ, adjusting the threshold might be necessary. Additionally, further exploration into alternative evaluation metrics, like precision, recall, and F1 score, can offer a more nuanced understanding of the model's strengths and weaknesses. Interpretation of accuracy should be accompanied by a thoughtful consideration of the application's requirements and the relative importance of correct classifications. Experimenting with different thresholds and evaluating the model's robustness against various performance metrics will contribute to a more informed decision-making process for deploying the model in real-world scenarios.

4.1 Feedback collected from client

- Report more than just accuracy in the conclusion. Compare with precision, recall, F1 score, etc.
- Can also choose to compare between algorithms instead of using just one.
- Explain how the results relate to the nature of the disease instead of just stating the results from evaluation matrices.
- Overall, very good presentation.
- Consistently had meetings throughout the semester and showed good progress
- We have made constant improvements based on dr’s comments during each progress

CHAPTER 5

CONCLUSION

5.0 Conclusion

Several evaluation matrices can be used to assess the model's performance. Accuracy is one commonly used metric, which measures the proportion of correctly classified cases. Precision, on the other hand measures the proportion of correctly predicted positive cases out of the total predicted positive cases, while recall (also known as sensitivity) measures the proportion of correctly predicted positive cases out of the actual positive cases.With an accuracy of 0.7, the classification model demonstrates a moderate level of correctness by accurately predicting the target variable for approximately 70% of instances in the test set. This performance suggests a reasonable overall predictive ability, but it's crucial to interpret the accuracy in the specific context of the classification problem. Depending on the consequences of false positives and false negatives and the overall goals of the analysis, a 70% accuracy may be deemed acceptable or may prompt further investigation. The accuracy figure provides a clear and easily communicable summary of the model's performance, but it's recommended to complement this with a more in-depth examination using additional evaluation metrics and considerations such as class distribution and potential areas for improvement. In this comprehensive analysis and modeling process, we started by loading and exploring the ovarian cancer dataset. We conducted various preprocessing steps, including feature selection, standardization, and binning. Key statistical techniques such as ANOVA and correlation were employed to understand relationships between variables. The modeling phase involved the application of linear regression, polynomial regression, and ridge regression techniques. We systematically evaluated and refined the models, considering overfitting and under-fitting using different polynomial degrees. The Ridge Regression model, crucially enhanced through grid search, demonstrated its effectiveness in handling regularization. The decision-making process illustrated how predictions from the refined Ridge Regression model can be employed for practical insights. Moreover, model evaluation metrics, such as mean squared error and R-squared, were utilized to assess the model's predictive accuracy. In summary, this analytical journey provides a robust framework for understanding, modeling, and refining predictions related to ovarian cancer based on the given dataset. The insights gained from this process can contribute to informed decision-making in a clinical or research context, paving the way for more accurate assessments of the disease's characteristics and potential outcomes.

CHAPTER 6

APPENDIX

6.0 Appendix

Video Presentation Link:

https://drive.google.com/drive/folders/1Nl5vaGC1Nroc7JN9bGUcd-RgQOZGLdfs?usp=sharing

GitHub Link:

https://github.com/NiesHW/SECB3203_P4B/tree/main/Group_Project/Group_6/Final