HACETTEPE UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ARTIFICIAL INTELLIGENCE ENGINEERING

BASIC MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE PROJECT REPORT

AIN422 - INTRODUCTION TO DEEP LEARNING LABORATORY

Assignment 1

Hüseyin Eren DOĞAN
2210765009

2024-04-01

1. Introduction:

In this project, the utilization of basic machine learning and artificial intelligence techniques was employed for the prediction of heart diseases. The objective revolved around identifying and analyzing correlations between heart diseases and various medical attributes, subsequently classifying the diseases to facilitate the development of a solution for medical researchers in their diagnosis processes.

The goals of this project are as follows:

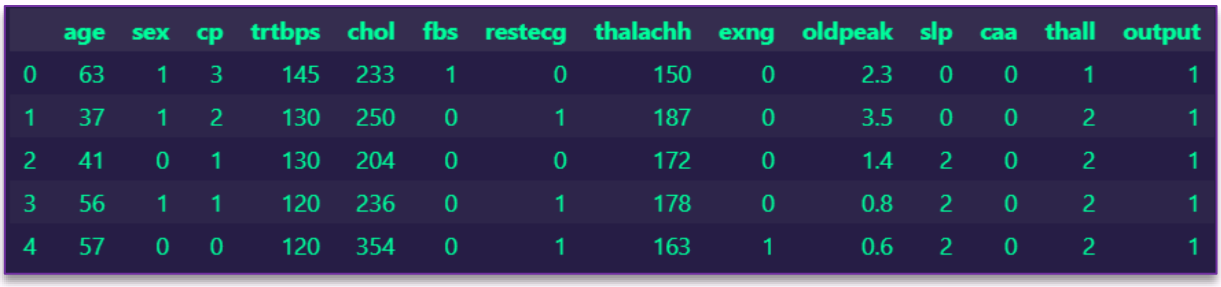
- To establish a data analysis system capable of detecting and recognizing patterns, trends, and insights within the data, particularly focusing on its correlation with the risks of heart attacks.
- To undertake a comparative analysis aimed at enhancing the performance of various supervised machine learning models concerning medical diagnosis prediction tasks.
- To effectively interpret the prediction results by selecting appropriate evaluation metrics and conducting thorough analyses of the data and outcomes to ensure precision and accuracy.

2. Data:

The dataset provided for this project, named 'medical_heart.csv', is structured in a tabular format comprising 304 rows and 14 columns. It encompasses information on 303 patients, some of whom have a history of previous heart attacks while others do not exhibit any heart-related issues.

The dataset's columns represent various medical and biological attributes of the patients, described as follows:

age:	Age of the patient	(years)
sex:	Sex of the patient	(1 = male, 0 = female)
cp:	Type of chest pain	(ranging in integers 0 to 3)
trestbps:	Resting blood pressure	(mmHg)
chol:	Serum cholesterol	(mg/dL)
fbs:	Fasting blood sugar	(1 = greater than 120 mg/dL, 0 = otherwise)
restecg:	Resting electrocardiographic results	(ranging in integers 0 to 2)
thalachh:	Maximum heart rate achieved	
exng:	Exercise induced angina	(1 = yes, 0 = no)
oldpeak:	ST depression induced by exercise	
slp:	Slope of the peak exercise ST segment	
caa:	Major vessels colored by fluoroscopy	(ranging in integers 0 to 4)
thall:	Thallassemia, a blood disorder	(ranging in integers 0 to 3)
output:	Diagnosis of heart disease	(1 = yes, 0 = no)



	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Figure 1: First five rows of the dataset

Following the dataset description, visualizations were generated to illustrate the distribution of each column's data through histograms. Additionally, a correlation matrix was constructed to explore the relationships between the attributes and the target attribute, which in this context is the 'output' column denoting the diagnosis of heart disease. These visual aids serve to facilitate a comprehensive analysis of the dataset and aid in identifying potential patterns and correlations among the variables.

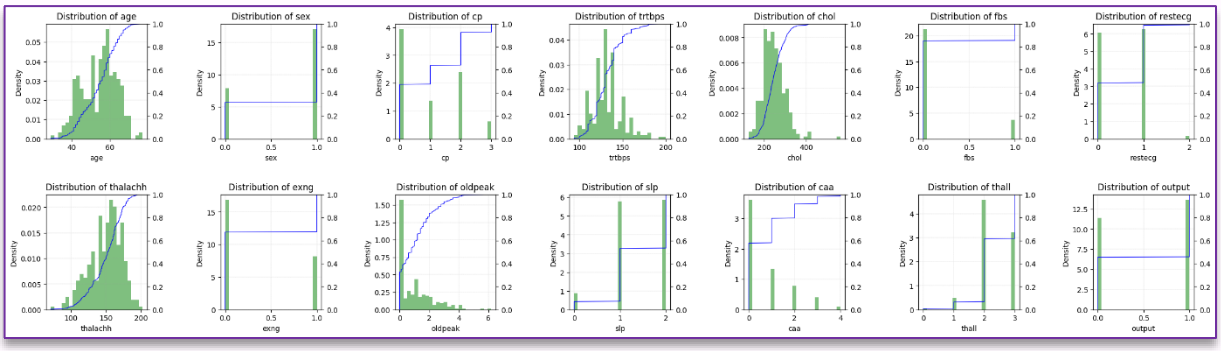
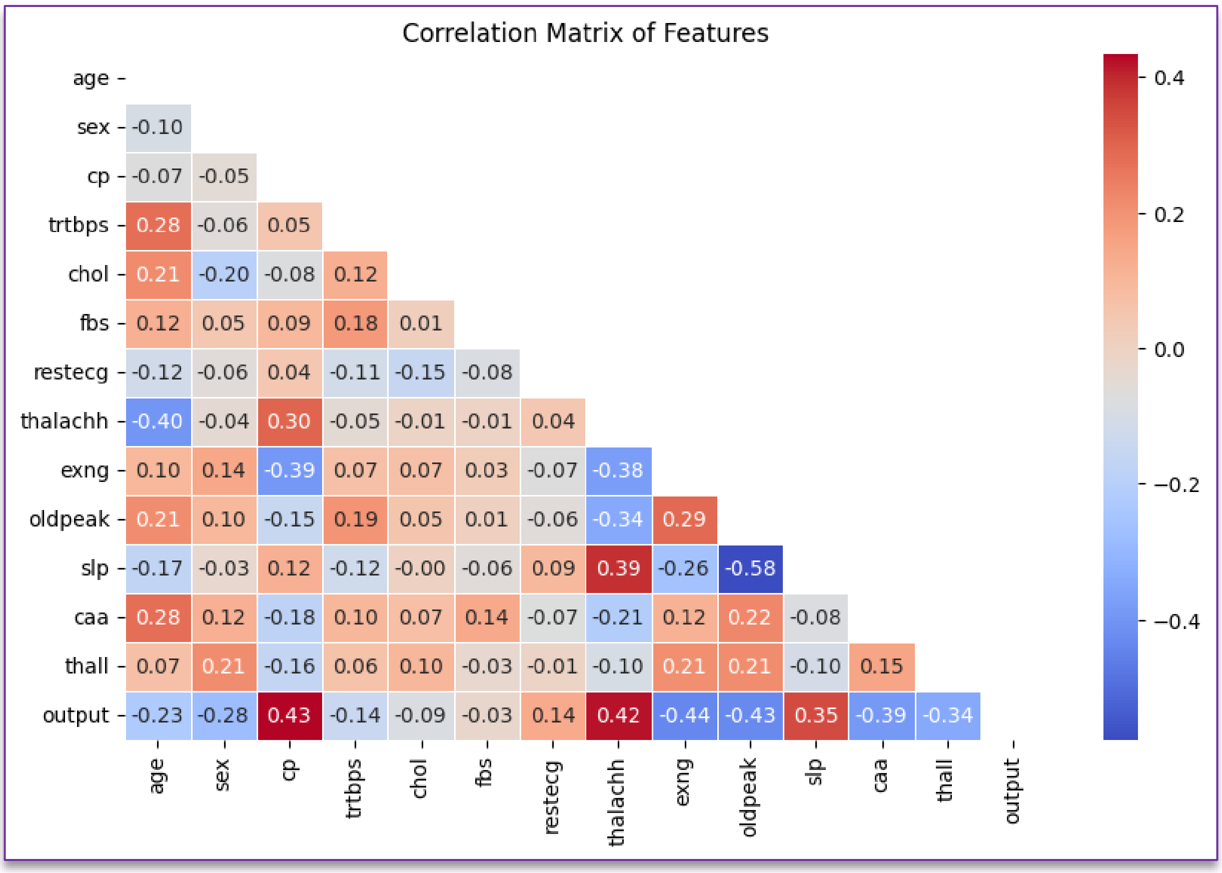


Figure 2: Distribution of all columns across the dataset



	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
age														
sex	-0.10													
cp	-0.07	-0.05												
trtbps	0.28	-0.06	0.05											
chol	0.21	-0.20	-0.08	0.12										
fbs	0.12	0.05	0.09	0.18	0.01									
restecg	-0.12	-0.06	0.04	-0.11	-0.15	-0.08								
thalachh	-0.40	-0.04	0.30	-0.05	-0.01	-0.01	0.04							
exng	0.10	0.14	-0.39	0.07	0.07	0.03	-0.07	-0.38						
oldpeak	0.21	0.10	-0.15	0.19	0.05	0.01	-0.06	-0.34	0.29					
slp	-0.17	-0.03	0.12	-0.12	-0.00	-0.06	0.09	0.39	-0.26	-0.58				
caa	0.28	0.12	-0.18	0.10	0.07	0.14	-0.07	-0.21	0.12	0.22	-0.08			
thall	0.07	0.21	-0.16	0.06	0.10	-0.03	-0.01	-0.10	0.21	0.21	-0.10	0.15		
output	-0.23	-0.28	0.43	-0.14	-0.09	-0.03	0.14	0.42	-0.44	-0.43	0.35	-0.39	-0.34	

Figure 3: Correlation Matrix of Features in Heart Attacks Dataset

The visualizations presented offer valuable insights into the dataset, enhancing our comprehension of the data's distribution and structure. This understanding serves as a foundation for the development and refinement of the methodology employed in this project.

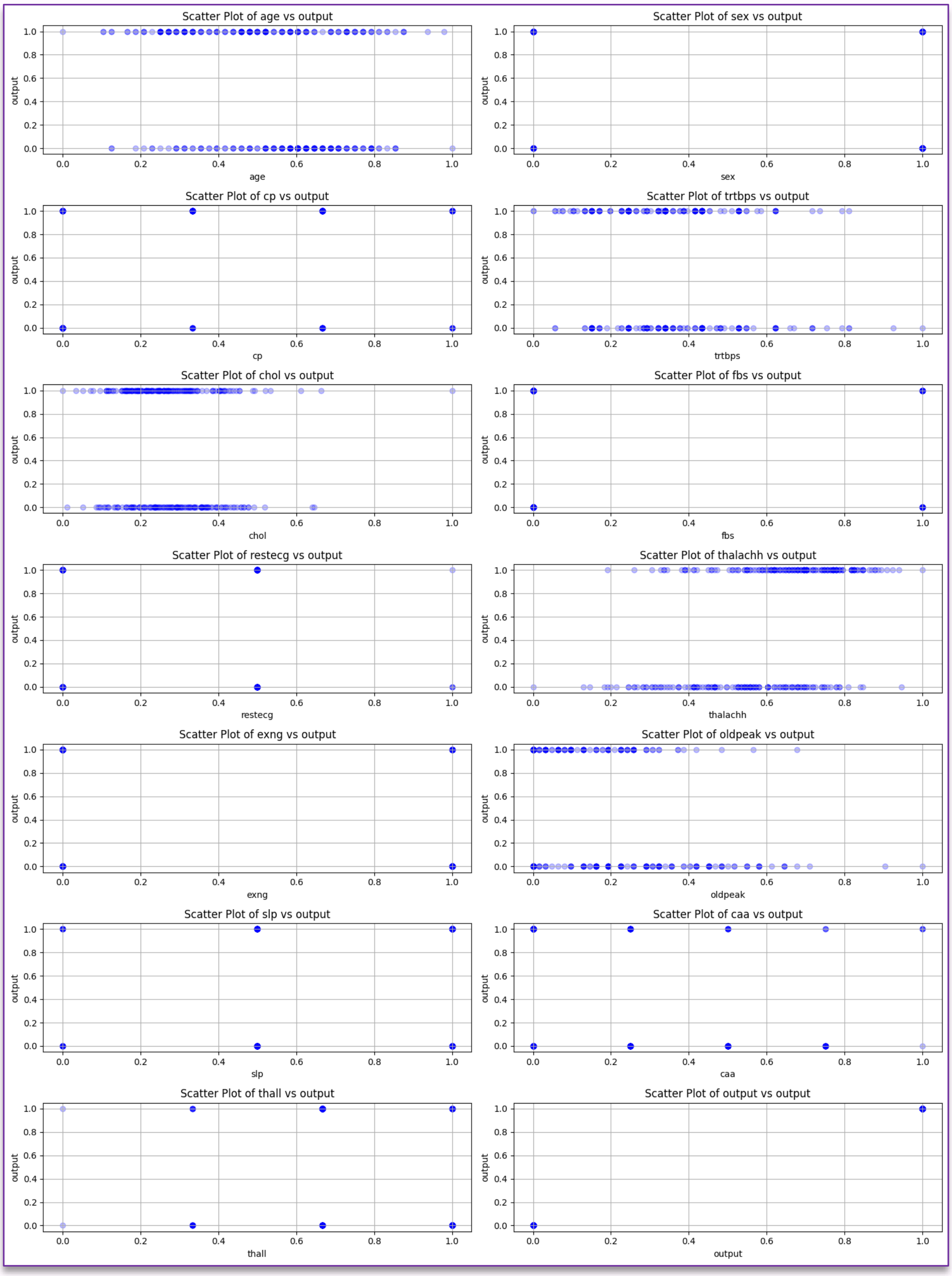


Figure 4: Scatter plots for every column with the 'output' column

3. Methodology:

A foundational machine learning approach was employed in this project, comprising data preprocessing, model selection, cross-validation, hyperparameter tuning, model evaluation on the test set, and acquisition of pertinent metrics tailored to the task at hand, thus facilitating subsequent discussion of the outcomes.

3.1. Data Preprocessing:

In this section, an analysis was conducted on how the data could be prepared for evaluating classifier models. Firstly, the presence of missing values was checked, and categorical values which needs to be encoded was checked. Fortunately, it was found that the dataset had already been cleaned in this regard, as it can be seen in Figure 1.

Subsequently, scaling of values was performed. Following this, an examination of the data before evaluation was decided upon. Distributions for all columns were constructed, as illustrated in Figure 2.

Then a correlation matrix was plotted, as shown in Figure 3, and scatter plots of every attribute against the target attribute was plotted, as shown in Figure 4, to gain a better understanding of the data. After this analysis, columns with the least correlation to the target column 'output' were removed.

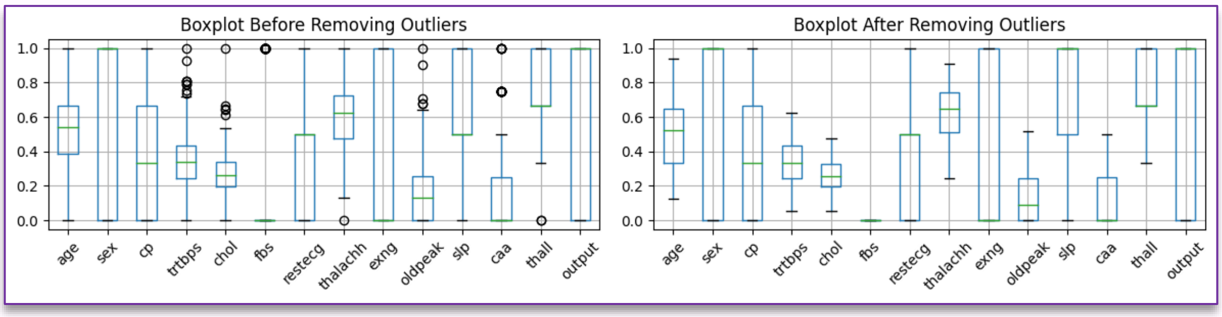


Figure 5: Boxplots of all columns before and after removing outliers

Additionally, outliers were looked for and removed as shown in Figure 5, leading to the splitting of the dataset into training and testing sets in balance to preparation for subsequent model training and evaluation.

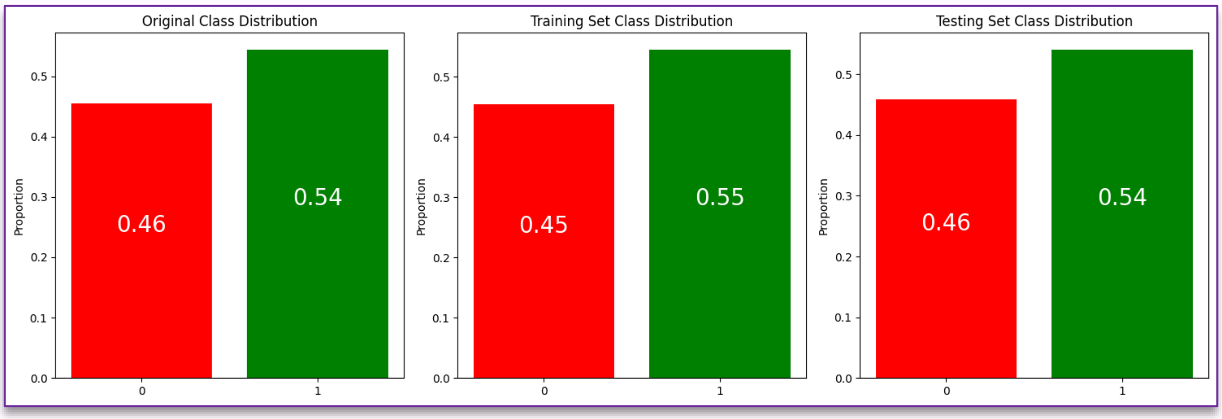


Figure 6: Barplots that showing proportions of target values to show the data is splitted in balance to the train and test sets

3.2. Model Selection and Cross Validation:

A range of classifiers was chosen to explore their applicability to the heart disease prediction task, with emphasis on models tailored for binary classification problems. Here are brief explanations for selecting each model:

- ***Logistic Regression***: A commonly used linear model for binary classification tasks, known for its simplicity and interpretability.
- ***Decision Tree***: Provides a clear decision-making process by partitioning the feature space into segments, making it particularly useful for understanding feature importance.
- ***Random Forest***: An ensemble learning method that combines multiple decision trees, offering improved generalization performance and robustness against overfitting.
- ***Support Vector Classifier***: Effective for binary classification tasks, especially in scenarios with non-linear decision boundaries, owing to its ability to map data into high-dimensional feature spaces.
- ***K-Nearest Neighbor Classifier***: This algorithm is known for its simplicity and effectiveness in classifying data points based on the majority class of their nearest neighbors. It is particularly suitable for tasks characterized by complex decision boundaries, non-linear relationships, and datasets that are not excessively large. Its intuitive approach makes it a valuable choice, especially when interpretability and computational efficiency are prioritized.

After selecting the models to be utilized, the next step involved evaluating and enhancing their performances through hyperparameter tuning using cross-validation techniques. Two methods were employed for this purpose:

- ***K-Fold Cross-Validation:*** This technique partitions the dataset into k subsets of equal size, allowing each subset to serve as both a training and validation set. The model is trained k times, each time using a different subset as the validation set and the remaining data for training. This method provides a robust estimation of the model's performance and helps mitigate overfitting.

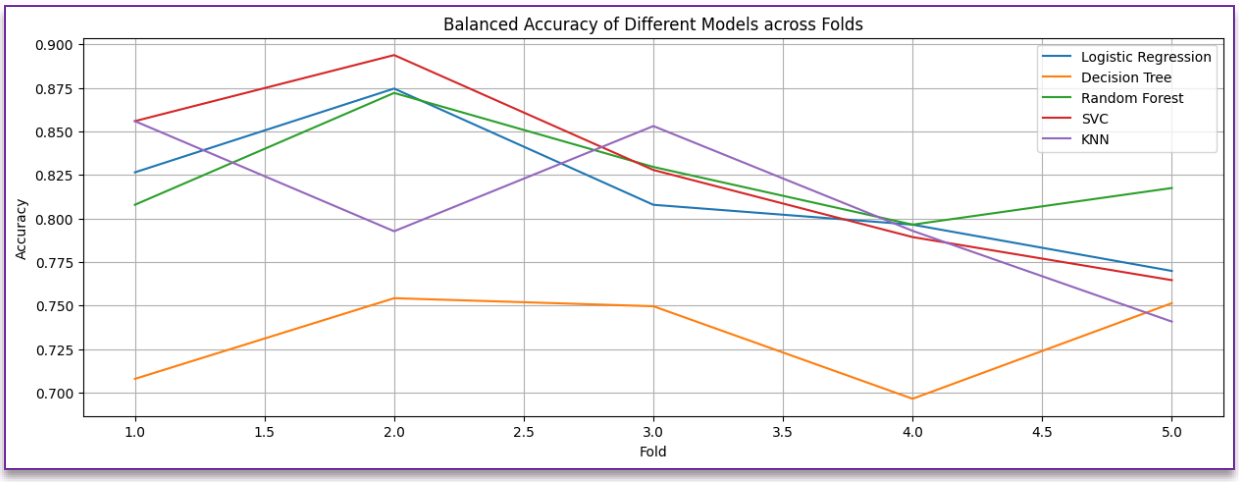
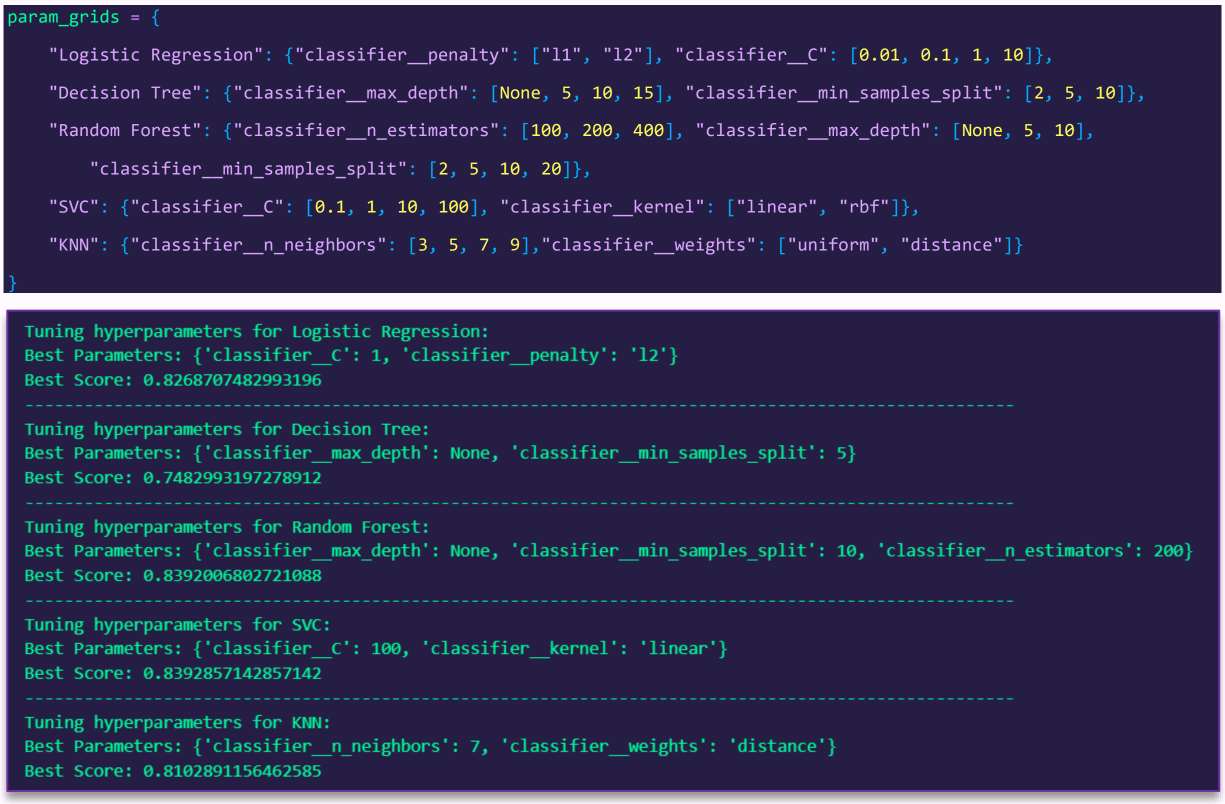


Figure 7: Lineplot to show balanced accuracy scores of all classifiers in 5-Fold Cross-Validation

- ***GridSearchCV:*** This is a method for hyperparameter tuning that systematically searches through a predefined grid of hyperparameters to find the optimal combination that yields the best performance for the model. It exhaustively evaluates all possible combinations of hyperparameters using cross-validation, making it an efficient approach for optimizing model performance.

There are the parameter grids and result of GridSearchCV with best parameters below:

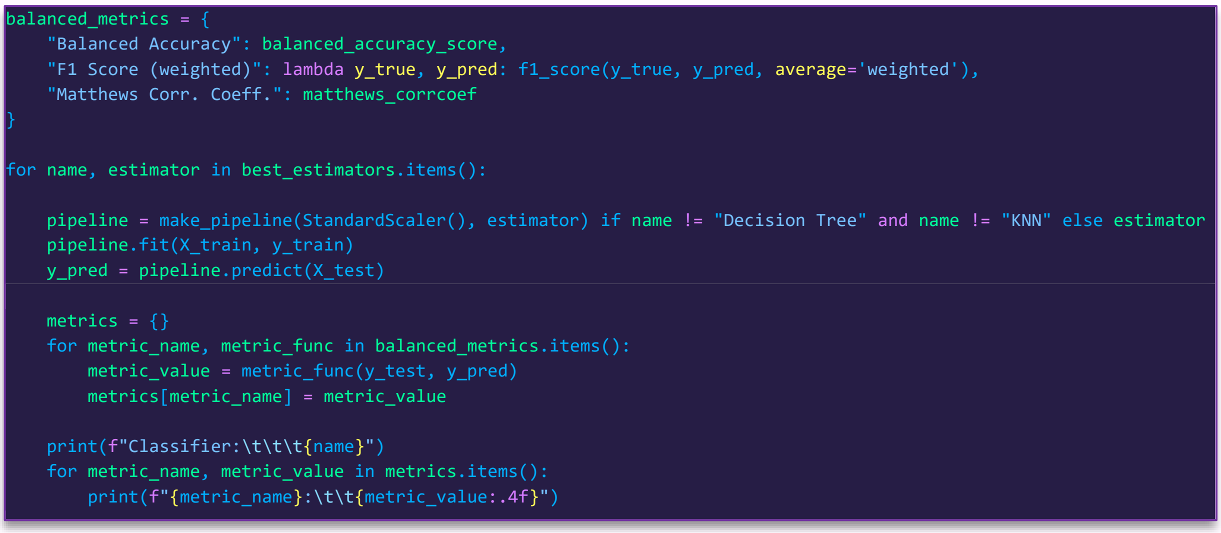


```
param_grids = {
    "Logistic Regression": {"classifier__penalty": ["l1", "l2"], "classifier__C": [0.01, 0.1, 1, 10]},
    "Decision Tree": {"classifier__max_depth": [None, 5, 10, 15], "classifier__min_samples_split": [2, 5, 10]},
    "Random Forest": {"classifier__n_estimators": [100, 200, 400], "classifier__max_depth": [None, 5, 10],
        "classifier__min_samples_split": [2, 5, 10, 20]},
    "SVC": {"classifier__C": [0.1, 1, 10, 100], "classifier__kernel": ["linear", "rbf"]},
    "KNN": {"classifier__n_neighbors": [3, 5, 7, 9],"classifier__weights": ["uniform", "distance"]}
}
```

```
Tuning hyperparameters for Logistic Regression:
Best Parameters: {'classifier__C': 1, 'classifier__penalty': 'l2'}
Best Score: 0.8268707482993196
------------------------------------------------------------------------------------------------
Tuning hyperparameters for Decision Tree:
Best Parameters: {'classifier__max_depth': None, 'classifier__min_samples_split': 5}
Best Score: 0.7482993197278912
------------------------------------------------------------------------------------------------
Tuning hyperparameters for Random Forest:
Best Parameters: {'classifier__max_depth': None, 'classifier__min_samples_split': 10, 'classifier__n_estimators': 200}
Best Score: 0.8392006802721088
------------------------------------------------------------------------------------------------
Tuning hyperparameters for SVC:
Best Parameters: {'classifier__C': 100, 'classifier__kernel': 'linear'}
Best Score: 0.8392857142857142
------------------------------------------------------------------------------------------------
Tuning hyperparameters for KNN:
Best Parameters: {'classifier__n_neighbors': 7, 'classifier__weights': 'distance'}
Best Score: 0.8102891156462585
```

Figure 8: Coding part of defining param_grids and results of GridSearchCV method

3.3. Model Evaluation:



```
balanced_metrics = {
    "Balanced Accuracy": balanced_accuracy_score,
    "F1 Score (weighted)": lambda y_true, y_pred: f1_score(y_true, y_pred, average='weighted'),
    "Matthews Corr. Coeff.": matthews_corrcoef
}

for name, estimator in best_estimators.items():

    pipeline = make_pipeline(StandardScaler(), estimator) if name != "Decision Tree" and name != "KNN" else estimator
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    metrics = {}
    for metric_name, metric_func in balanced_metrics.items():
        metric_value = metric_func(y_test, y_pred)
        metrics[metric_name] = metric_value

    print(f"Classifier:\t\t\t{name}")
    for metric_name, metric_value in metrics.items():
        print(f"{metric_name}:\t\t{metric_value:.4f}")
```

Figure 9: Coding part for implementation of evaluating models and choosing balanced metrics

As illustrated in Figure 9, the tuned models obtained from the 'best_estimators' dictionary were utilized. Subsequently, a pipeline was constructed for standard scaling using the StandardScaler, which transforms the data to ensure that each feature has a mean of 0 and a standard deviation of 1. This standardization is particularly crucial for models sensitive to the scale of input features, such as the K-Nearest Neighbor Classifier and Decision Tree.

For evaluating the classifiers on the test set, the following three metrics were selected:

- ***Balanced Accuracy Score***: This metric evaluates the accuracy of a classifier in a balanced manner, considering the distribution of classes in the dataset. It is especially useful when dealing with imbalanced datasets.

- ***F1-Score***: The F1-Score is the harmonic mean of precision and recall and provides a balance between these two metrics. It is particularly suitable for binary classification tasks, where there is an imbalance between the classes.

- ***Matthew's Correlation Coefficient***: Matthew's Correlation Coefficient (MCC) is a measure of the quality of binary classifications, considering both true and false positives and negatives. It ranges from -1 to 1, where 1 indicates perfect predictions, 0 represents random predictions, and -1 indicates complete disagreement between predictions and observations.

These metrics collectively provide a comprehensive assessment of the classifiers' performance on the test set, taking into account various aspects such as accuracy, precision, recall, and the balance between true positive and false positive rates.

4. Development:

4.1. Requirements:

- ***Hardware Requirements***: The analysis and model training processes were executed on standard computing hardware, which may have imposed limitations on the complexity and scale of data processing and model training. Particularly noteworthy are the constraints encountered during hyperparameter tuning, which demands substantial computational resources. Despite these limitations, efforts were made to optimize the performance within the available hardware constraints.

- ***Software Requirements***: Python emerged as the primary programming language for this project due to its versatility and extensive ecosystem of libraries. Key libraries utilized include pandas for data manipulation tasks, scikit-learn for machine learning algorithms, and matplotlib for data visualization. Leveraging these software components facilitated efficient data handling, model training, and result visualization, contributing to the overall effectiveness of the analysis.

4.2. Implementation:

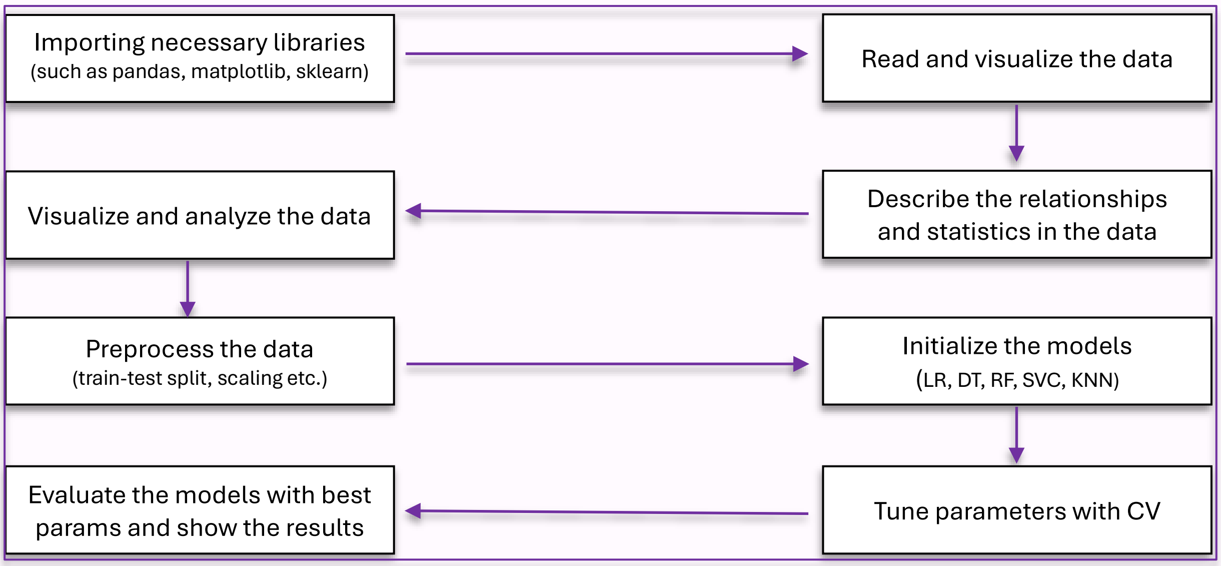

Importing necessary libraries (such as pandas, matplotlib, sklearn)
Read and visualize the data
Describe the relationships and statistics in the data
Visualize and analyze the data
Preprocess the data (train-test split, scaling etc.)
Initialize the models (LR, DT, RF, SVC, KNN)
Tune parameters with CV
Evaluate the models with best params and show the results

Figure 10: Flowchart of Implementation Process

- ***Importing necessary libraries***: Essential libraries such as pandas, matplotlib, seaborn, numpy, and scikit-learn were imported to facilitate data manipulation, visualization, and machine learning tasks.

- ***Read and visualize the data***: The dataset was loaded into memory using pandas, and a preliminary visualization was conducted to provide an initial overview of the data.

- ***Describe the relationships and statistics in the data***: Statistical summaries and relationships within the data were explored to gain insights into its characteristics and distribution.

- ***Visualize and analyze the data***: Various visualization techniques, including histograms, correlation matrices, and scatter plots, were employed to analyze and understand the data better visually.

- ***Preprocess the data***: Data preprocessing steps, such as outlier removal, feature scaling, and splitting into balanced train and test sets, were performed to prepare the data for model training.

- ***Initialize the models***: The classifier models, including Logistic Regression, Decision Tree, Random Forest, Support Vector Classifier, and K-Nearest Neighbor Classifier, were initialized with default parameters.

- ***Tune parameters with CV***: Hyperparameter tuning was conducted using the 5-Fold GridSearchCV method to find the optimal parameters for each model.

- ***Evaluate the models with best params and show the results***: The tuned models were evaluated on the test set using balanced metrics such as balanced accuracy score, F1-score, and Matthew's correlation coefficient. Confusion matrices were also presented to further analyze the performance of each model.

5. Catalog:

5.1. Time Spent:

Analysis:	1 hour
Design:	4 hours
Implementation:	6 hours
Testing:	1 hour
Reporting:	3 days

5.2. Programmer Catalog:

- ***Modularity:*** The code is organized into modular functions, making it easy to extract and reuse specific functionalities for other projects or datasets.
- ***Configurability:*** Parameters and settings are often exposed as variables or arguments, allowing for easy customization to adapt the code to different requirements or datasets.
- ***Standard Libraries***: Standard Python libraries such as pandas, scikit-learn, and matplotlib are used extensively, ensuring compatibility and familiarity for other programmers.

All the functions are in first cell of the .ipynb file, so it makes it easy to access, here are the functions:

1. def load_data(file_path)
2. def visualize_data_hist(df)
3. def scale_data(df, scaler)
4. def plot_correlation_matrix(df)
5. def visualize_data_scatter(df, target)
6. def remove_outliers(df)
7. def remove_least_correlated(df, target)
8. def traintest_split(df, target)
9. def plot_balanced_split(y, y_train, y_test)
10. def K_Fold_CV(X_train, y_train, classifiers)
11. def grid_search_cv(X_train, y_train, classifiers, param_grids)
12. def evaluate_models(X_train, y_train, X_test, y_test, balanced_metrics, best_estimators)

Then here are the explanations of parameters of functions:

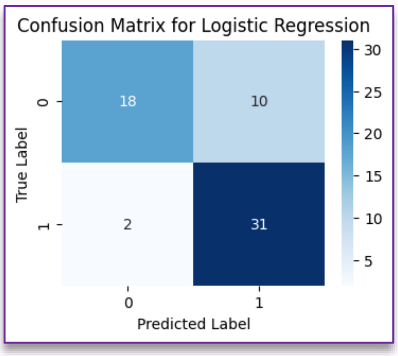
file_path:	Path of the .csv file
df:	DataFrame object returning from 1st function
scaler:	Scaler which you want to use; MinMaxScaler(), StandardScaler() etc.
target:	Target column's name
X_train, y_train, X_test, y_test:	Splitted datasets which is output of function 8
classifiers:	Initialized classifiers with default parameters in dictionary form
param_grids:	Parameter grids for classifiers in dictionary form
best_estimators:	Tuned classifiers which is output of function 11, in dictionary form
balanced_metrics:	Selected metrics to evaluation, in dictionary form

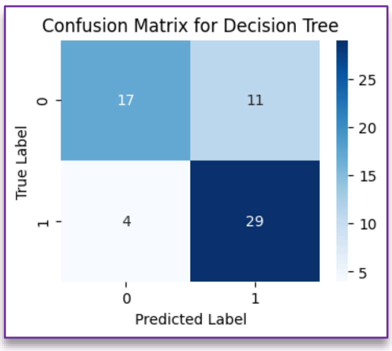
5.3. User Catalog:

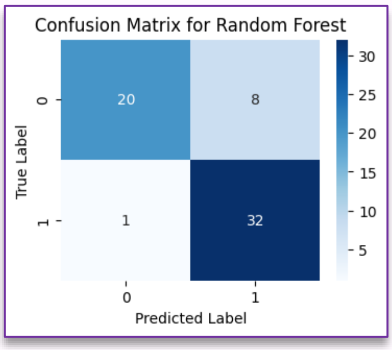
- ***Environment Setup***: Install the requisite Python libraries, including pandas, scikit-learn, and matplotlib. You can install these libraries using pip, a package manager for Python.
- ***Data Preparation***: Ensure data is in a clean CSV format, with appropriately typed features ready for analysis. You can use any spreadsheet software such as Microsoft Excel or Google Sheets to clean and prepare your data before loading it into the program.
- ***Code Execution***: Replace the 'file_path' parameter in the load_data() with the path to your .csv file. This path should point to the location of your dataset on your local machine or network.
- Run the code blocks sequentially to perform data analysis, visualization, and preprocessing. You can do this by executing the code cells in a Python environment such as Jupyter or in a Python script editor.
- Model training and evaluation processes are automated, requiring no modifications unless specific changes to hyperparameters or model inclusion are desired. Simply execute the code blocks containing model training and evaluation functions to obtain results.
- ***Interpreting Output***: The code outputs a comprehensive report detailing data exploratory findings and model performance metrics, aiding users in making informed decisions based on the machine learning analysis. You can review the printed output or saved reports generated by the program to interpret the results. The report includes visualizations such as histograms, correlation matrices, and confusion matrices, as well as numerical metrics such as balanced accuracy, F1-score etc.

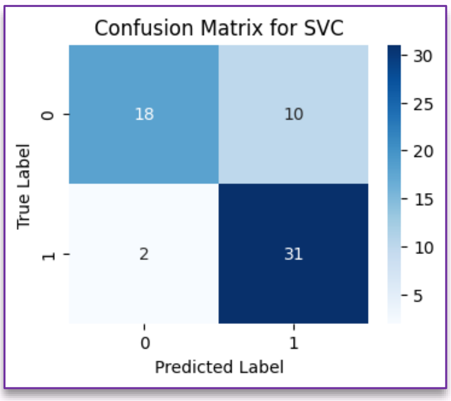
6. Results:

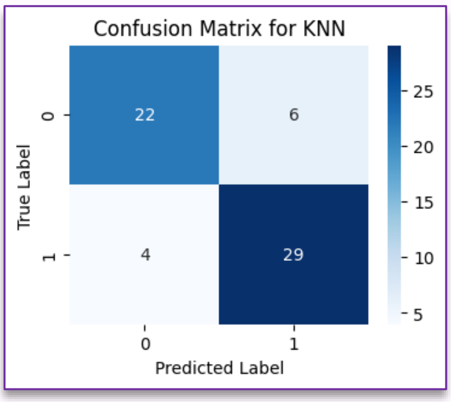
Classifier:	LR	DT	RF	SVC	KNN
Balanced Accuracy:	0.7911	0.7430	0.8420	0.7911	0.8323
F1 Score (weighted):	0.7975	0.7483	0.8490	0.7975	0.8354
Matthews Corr. Coeff.:	0.6181	0.5096	0.7174	0.6181	0.6696











When evaluating classifier models, the choice of metric depends on the specific task requirements. Balanced accuracy score is useful for datasets with class imbalance, while F1-score balances precision and recall, and Matthews Correlation Coefficient considers true positives, true negatives, false positives, and false negatives. Therefore, using a combination of these metrics provides a comprehensive evaluation of classifier performance, guiding the selection of the most suitable model for the task.

- ***Logistic Regression***: The Logistic Regression classifier achieved a balanced accuracy of 0.7911, indicating a good overall performance in classifying both positive and negative instances. The F1 score and Matthews Correlation Coefficient are also relatively high, indicating a good balance between precision and recall and a strong correlation between predicted and observed classifications. The confusion matrix reveals that the model correctly classified 18 true negatives and 31 true positives, with 10 false positives and 2 false negatives.

- ***Decision Tree***: The Decision Tree classifier achieved a slightly lower balanced accuracy compared to Logistic Regression, indicating a moderate performance in classifying instances. The F1 score and Matthews Correlation Coefficient are also lower, suggesting a slightly weaker balance between precision and recall and a weaker correlation between predicted and observed classifications. The confusion matrix shows that the model correctly classified 17 true negatives and 29 true positives, with 11 false positives and 4 false negatives.

- ***Random Forest***: The Random Forest classifier achieved the highest balanced accuracy among all classifiers, indicating the best overall performance in classifying instances. The F1 score and Matthews Correlation Coefficient are also the highest, suggesting a strong balance between precision and recall and a strong correlation between predicted and observed classifications. The confusion matrix reveals that the model correctly classified 20 true negatives and 32 true positives, with 8 false positives and 1 false negative.

- ***Support Vector Classifier (SVC)***: The SVC classifier achieved a balanced accuracy similar to Logistic Regression, indicating a good overall performance in classifying instances. The F1 score and Matthews Correlation Coefficient are also similar, suggesting a comparable balance between precision and recall and a similar correlation between predicted and observed classifications. The confusion matrix shows results similar to Logistic Regression, with 18 true negatives, 31 true positives, 10 false positives, and 2 false negatives.

- ***K-Nearest Neighbor Classifier (KNN)***: The KNN classifier achieved a balanced accuracy slightly lower than Random Forest but higher than Logistic Regression and SVC. The F1 score and Matthews Correlation Coefficient are also relatively high, indicating a good balance between precision and recall and a strong correlation between predicted and observed classifications. The confusion matrix reveals that the model correctly classified 22 true negatives and 29 true positives, with 6 false positives and 4 false negatives.

Overall, Random Forest appears to be the best-performing classifier based on the provided metrics, achieving the highest balanced accuracy, F1 score, and Matthews Correlation Coefficient. It also exhibits the most balanced performance across true positives, true negatives, false positives, and false negatives in the confusion matrix.