AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING I-CHEP
COMPUTER ENGINEERING AND SOFTWARE SYSTEMS PROGRAM
2024-2025

Heart Failure Classification Models Report

Introduction to Machine Learning CSE381

PREPARED BY:

AbdulRahman Hesham Kamel - 21P0153
Ahmed Nezar Ahmed - 21P0025
Kirollos Ehab Magdy - 21P0006

COURSE COORDINATORS:

Prof. Mahmoud Khalil
Eng. Engy Ahmed Hassan

Contents

1 Introduction **1**
- 1.1 Overview of the Project . . . 1
- 1.2 Objectives and Scope . . . 2
  - 1.2.1 Objectives . . . 2
  - 1.2.2 Scope . . . 3

2 Dataset Description **4**
- 2.1 Source and Features . . . 4
  - 2.1.1 Source . . . 4
  - 2.1.2 Features . . . 4
  - 2.1.3 Features Data Types Numerical Features Statistical Description . . . 6
- 2.2 Target Variable . . . 7
  - 2.2.1 Details of the Target Variable . . . 7
  - 2.2.2 Importance of the Target Variable . . . 7
  - 2.2.3 Preprocessing for the Target Variable . . . 7
  - 2.2.4 Scenario-Specific Observations . . . 8
  - 2.2.5 Role in the Project . . . 8

3 Exploratory Data Analysis (EDA) **9**
- 3.1 Checking for duplicate or missing values . . . 9
- 3.2 Visualizations . . . 9
  - 3.2.1 KDE (Kernel Density Estimation) Plots . . . 9
  - 3.2.2 PIE Charts . . . 12
  - 3.2.3 Bar Plots . . . 14
  - 3.2.4 Box Plots . . . 17
  - 3.2.5 Scatter Plots . . . 20
- 3.3 Correlation of Numerical Features . . . 28

4 Data Cleaning and Preprocessing **29**
- 4.1 Scenarios . . . 29
  - 4.1.1 Detailed Scenarios Description . . . 29
  - 4.1.2 Overall Purpose of Scenarios . . . 31
  - 4.1.3 Dataset Characteristics in Scenarios . . . 31
- 4.2 Handling Missing Values . . . 33
  - 4.2.1 Observations . . . 33
  - 4.2.2 Impact on Scenarios . . . 33
- 4.3 Outlier Detection and Treatment . . . 33
  - 4.3.1 Observations . . . 33
  - 4.3.2 Impact on Scenarios . . . 33

4.4 Duplicate Record Removal . . . 34
4.4.1 Observations . . . 34
4.4.2 Impact on Scenarios . . . 34
4.5 Data Splitting (Training, Validation, Testing) . . . 34
4.5.1 Observations . . . 34
4.5.2 Impact on Scenarios . . . 34
4.6 General Observations . . . 35

5 Model Implementation **36**
5.1 Description of Models . . . 36
5.1.1 Naive Bayes . . . 36
5.1.2 SVM (Support Vector Machine) . . . 37
5.1.3 KNN (K-Nearest Neighbors) . . . 38
5.1.4 Decision Trees . . . 39
5.2 Training Procedures and Hyperparameter Tuning . . . 40
5.2.1 Preprocessing Scenarios . . . 40
5.2.2 Model Selection . . . 41
5.2.3 Hyperparameter Tuning . . . 41
5.2.4 Evaluation Metrics . . . 42
5.2.5 Visualization of Results . . . 42
5.2.6 Best Model Selection and Insights . . . 43
5.2.7 Model Persistence and Reproducibility . . . 43
5.3 Performance Metrics . . . 44
5.3.1 Accuracy . . . 44
5.3.2 Precision . . . 44
5.3.3 Recall . . . 44
5.3.4 F1-Score . . . 44
5.3.5 Confusion Matrix . . . 44
5.3.6 Class-Wise Metrics . . . 45
5.3.7 Comparative Visualization . . . 45
5.3.8 Insights Gained . . . 45
5.4 Evaluation Metrics on Best Models . . . 46
5.4.1 Grid Search . . . 46
5.4.2 Random search . . . 54
5.4.3 Optuna . . . 62
5.5 Comparative Analysis of Models . . . 70
5.5.1 Naive Bayes . . . 70
5.5.2 Support Vector Machines (SVM) . . . 71
5.5.3 K-Nearest Neighbors (KNN) . . . 71
5.5.4 Decision Trees . . . 72

6 Code Implementation **74**
6.1 Imports and Necessary Libraries . . . 74
6.2 Data Exploration . . . 75
6.3 Data Visualization . . . 77
6.4 Data Preprocessing . . . 79
6.5 Data Preprocessing Scenarios . . . 81
6.6 Features PCA . . . 91
6.7 Modeling . . . 92
6.8 Modeling Util Classes . . . 94
6.9 Running HPO Scenarios . . . 96
6.10 HPO . . . 99
6.10.1 Naive Bayes . . . 99
6.10.2 SVM . . . 102
6.10.3 Decision Tree . . . 105
6.10.4 KNN . . . 108
6.11 Comparing Models Utils . . . 111
6.12 Cluster Analysis . . . 114

7 Dendrogram Analysis **116**
7.1 Hierarchical Clustering and Proximity Measures . . . 116
7.1.1 Hierarchical Clustering . . . 116
7.1.2 Dimensionality Reduction . . . 116
7.1.3 Visualization . . . 117
7.1.4 Cluster Validation . . . 117
7.1.5 Process Workflow . . . 117
7.1.6 Key Insights . . . 118
7.2 Visualization and Observations . . . 119
7.2.1 Scenario 1_N . . . 119
7.2.2 Scenario 1_S . . . 128
7.2.3 Scenario 2_N . . . 137
7.2.4 Scenario 2_S . . . 146
7.2.5 Scenario 3_N . . . 155
7.2.6 Scenario 3_S . . . 164
7.2.7 Scenario 4_N . . . 173
7.2.8 Scenario 4_S . . . 182
7.2.9 Scenario 5_N . . . 191
7.2.10 Scenario 5_S . . . 200
7.2.11 Scenario 6_N . . . 209
7.2.12 Scenario 6_S . . . 218
7.2.13 Scenario 7_N . . . 227

7.2.14 Scenario 7_S 236
7.2.15 Scenario 8_N 245
7.2.16 Scenario 8_S 254
7.3 Alignment with PCA and Classifier Results 263
7.3.1 Accuracy with PCA 2 dimensions 263

8 Results and Discussion **271**
8.1 Summary of Model Performances 271
8.2 Challenges Faced and Solutions 272
8.3 Key Insights 274

9 Conclusion **276**

1 Introduction

1.1 Overview of the Project

This project focuses on leveraging machine learning techniques to predict heart failure risk using the **Heart Failure Prediction Dataset**. The dataset includes clinical and demographic attributes such as age, cholesterol levels, resting blood pressure, and maximum heart rate achieved, alongside a target variable indicating the presence of heart disease.

The project's main objectives include:

1. **Data Exploration and Visualization:**
   Conduct a thorough exploration of the dataset to identify patterns and trends. Utilize **Principal Component Analysis (PCA)** to reduce dimensionality and visualize the data in 2D space.

2. **Data Cleaning and Preprocessing:**
   Prepare the dataset for machine learning by addressing missing values, outliers, duplicates, and splitting the data into training, validation, and testing sets.

3. **Machine Learning Model Development:**
   Train and evaluate multiple classifiers, including:

   - **Naive Bayes**
   - **Support Vector Machines (SVM)**
   - **K-Nearest Neighbors (KNN)**
   - **Decision Trees**

   Explore hyperparameter tuning using techniques like grid search or random search to optimize model performance.

4. **Performance Evaluation:**
   Assess the models using metrics such as **precision**, **recall**, **F1-score**, and the **confusion matrix**, comparing their effectiveness in predicting heart failure.

5. **Dendrogram Analysis:**
   Construct a dendrogram to visualize hierarchical clustering relationships within the data, offering additional insights into patient groupings based on their attributes.

6. **Deliverables:**
   The project deliverables include a comprehensive **Jupyter Notebook** with code and output, as well as a detailed **report** documenting the process, results, and observations.

1.2 Objectives and Scope

1.2.1 Objectives

1. **Data Understanding and Analysis:**
   Explore the Heart Failure Prediction Dataset to uncover meaningful patterns and relationships between clinical and demographic attributes.

2. **Data Cleaning and Preparation:**
   Perform necessary preprocessing steps, including handling missing values, removing outliers, and splitting the dataset into training, validation, and testing sets, to ensure data quality.

3. **Model Training and Optimization:**
   Implement and train multiple machine learning models, including:

   - Naive Bayes
   - Support Vector Machines (SVM)
   - K-Nearest Neighbors (KNN)
   - Decision Trees

   Optimize hyperparameters using techniques like grid search or random search to improve model performance.

4. **Performance Evaluation:**
   Evaluate the models using appropriate metrics such as **precision**, **recall**, **F1-score**, and **confusion matrix**. Compare results to identify the best-performing model.

5. **Hierarchical Clustering Analysis:**
   Construct and analyze a dendrogram to visualize hierarchical clustering of data points, providing insights into natural groupings in the dataset.

6. **Integration of Results and Reporting:**
   Synthesize findings from data exploration, model performance, and clustering analysis into a comprehensive report that includes code snippets, visualizations, and detailed observations.

1.2.2 Scope

- **Dataset:**
  The project is limited to the Heart Failure Prediction Dataset, which contains a defined set of clinical and demographic attributes.
- **Machine Learning Models:**
  Focus on classical machine learning algorithms such as Naive Bayes, SVM, KNN, and Decision Trees. Deep learning or ensemble methods are beyond the scope of this project.
- **Evaluation Metrics:**
  Precision, recall, F1-score, and confusion matrices will be used to assess performance. Advanced evaluation techniques are not emphasized.
- **Dimensionality Reduction:**
  Utilize PCA for visualization and interpretation, restricting analysis to a 2D representation.
- **Clustering:**
  Hierarchical clustering is performed to explore patient groupings, with proximity measures such as dissimilarity or similarity.
- **Tools and Deliverables:**
  The project uses Python-based tools like **Jupyter Notebooks** for implementation. Deliverables include a documented notebook and a formal report detailing all findings.

2 Dataset Description

2.1 Source and Features

2.1.1 Source

The dataset used in this project is the **Heart Failure Prediction Dataset**, which includes clinical and demographic data collected from patients. The dataset is designed to predict the risk of heart failure, a leading cause of global mortality. It provides a valuable opportunity to apply machine learning techniques to address a critical healthcare challenge.

2.1.2 Features

The dataset comprises the following features:

1. **Age:** The age of the individual in years. (Numeric)
2. **Sex:** The biological sex of the individual, typically coded as 0 for female and 1 for male. (Category)
3. **ChestPainType:** The type of chest pain experienced by the individual, categorized into types such as:
   - Typical Angina
   - Atypical Angina
   - Non-Anginal Pain
   - Asymptomatic

   (Category)
4. **RestingBP:** Resting blood pressure (measured in mmHg) taken while the person is at rest. (Numeric)
5. **Cholesterol:** The cholesterol level in the blood, measured in mg/dL, including both LDL (bad cholesterol) and HDL (good cholesterol). (Numeric)

6. **FastingBS:** Fasting blood sugar level, measured after fasting for 8+ hours, typically coded as 0 if <120 mg/dL and 1 if $\geq 120 mg/dL.(Category)$

7. **RestingECG:** Results of the resting electrocardiogram, which records the electrical activity of the heart. Categories often include: (Category)

    - Normal
    - Showing ST-T wave abnormalities (e.g., ischemia)
    - Indicating probable or definite left ventricular hypertrophy

8. **MaxHR:** Maximum heart rate achieved during physical activity, measured in beats per minute (bpm). (Numeric)

9. **ExerciseAngina:** Presence of exercise-induced angina (chest pain due to reduced blood flow to the heart), typically coded as 0 for No and 1 for Yes. (Category)

10. **Oldpeak:** The ST depression value on the ECG, representing changes in the heart's electrical activity during or after exercise. It indicates ischemia severity. (Numeric)

11. **ST_Slope:** The slope of the ST segment during the ECG, categorized as: (Category)

    - Upsloping
    - Flat
    - Downsloping

12. **HeartDisease:** Indicator of the presence of heart disease, typically coded as 0 for No (absence) and 1 for Yes (presence). (Target/Category)

    These features serve as input variables for the machine learning models, helping to identify patterns and relationships that predict heart disease.
    The code implementation in the project further details each scenario, including specific preprocessing steps, justifications, and model evaluations. This systematic approach ensures thorough testing and reliable findings.

2.1.3 Features Data Types Numerical Features Statistical Description

Feature Name	Data Type
Age	int
Sex	String
ChestPainType	String
RestingBP	int
Cholesterol	int
FastingBS	int
RestingECG	String
MaxHR	int
ExerciseAngina	String
Oldpeak	float
ST_Slope	String
HeartDisease	int

Table 1: Dataset Features and Their Data Types

Statistic	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918	918	918	918	918	918	918
mean	53.51	132.40	198.80	0.23	136.81	0.89	0.55
std	9.43	18.51	109.38	0.42	25.46	1.07	0.50
min	28.00	0.00	0.00	0.00	60.00	-2.60	0.00
25%	47.00	120.00	173.25	0.00	120.00	0.00	0.00
50%	54.00	130.00	223.00	0.00	138.00	0.60	1.00
75%	60.00	140.00	267.00	0.00	156.00	1.50	1.00
max	77.00	200.00	603.00	1.00	202.00	6.20	1.00

Table 2: Summary Statistics for Dataset Features

2.2 Target Variable

The **target variable** in this project is a key feature that represents the presence or absence of heart disease in patients. It is a binary categorical variable used to define the outcome of the machine learning classification task.

2.2.1 Details of the Target Variable

- **Name**: The exact name may vary in the dataset (e.g., 'HeartDisease', 'Outcome', 'Target').
- **Values**:
  - '0': Indicates that the patient does **not** have heart disease.
  - '1': Indicates that the patient has been diagnosed with heart disease.

2.2.2 Importance of the Target Variable

1. **Supervised Learning**: The target variable is used to train and evaluate machine learning models, making it the central focus of this classification problem.
2. **Class Distribution**:
   - The distribution of the target variable (e.g., percentage of '0' vs. '1') provides insights into whether the dataset is balanced.
   - An imbalanced dataset (e.g., one class significantly outweighing the other) may require additional techniques such as resampling, class weighting, or advanced metrics for evaluation.
3. **Metrics for Evaluation**: The performance of the classifiers is assessed using metrics that account for the target variable, such as:
   - **Precision**: How many predicted positives are correct.
   - **Recall**: How many actual positives are correctly predicted.
   - **F1-Score**: A harmonic mean of precision and recall.
   - **Confusion Matrix**: Provides insights into true positives, false positives, true negatives, and false negatives.

2.2.3 Preprocessing for the Target Variable

- The target variable is encoded as a binary numeric variable ('0' and '1'), which is compatible with most machine learning algorithms.
- No further transformations are necessary, as the target is already in a usable format.

2.2.4 Scenario-Specific Observations

- In different scenarios, preprocessing strategies like handling missing values, removing outliers, or feature engineering do not alter the target variable directly.
- Class distribution in the target variable is monitored across scenarios to ensure that any preprocessing step does not skew the representation of ‘0‘ and ‘1‘.

2.2.5 Role in the Project

The target variable is guiding every step from data exploration and preprocessing to model training, evaluation, and interpretation. The ultimate goal of this project is to maximize the accurate prediction of this variable, enabling early detection of heart disease.

3 Exploratory Data Analysis (EDA)

3.1 Checking for duplicate or missing values

After inspection, it has been confirmed that there are no missing or duplicate values. Each feature in the dataset contains complete records, and all rows are unique.

3.2 Visualizations

3.2.1 KDE (Kernel Density Estimation) Plots

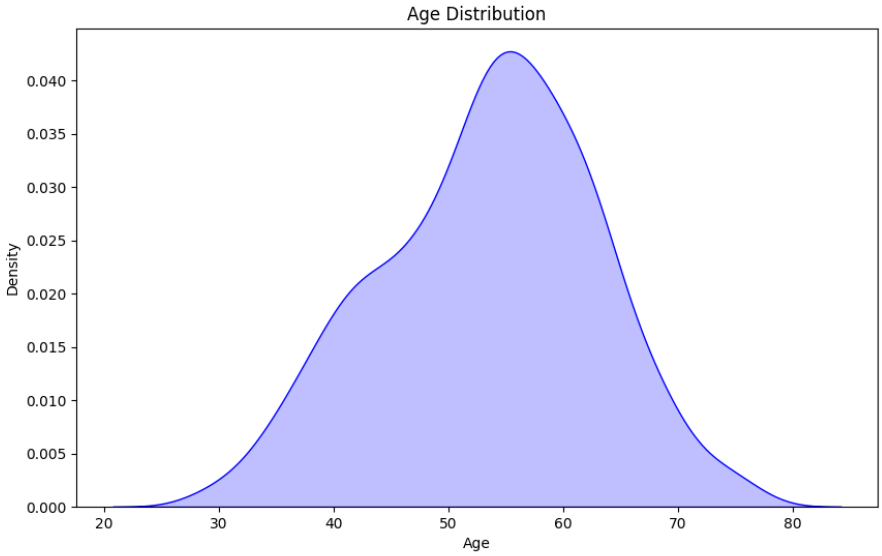


Figure 1: The KDE plot for Age shows the distribution of ages in the dataset.

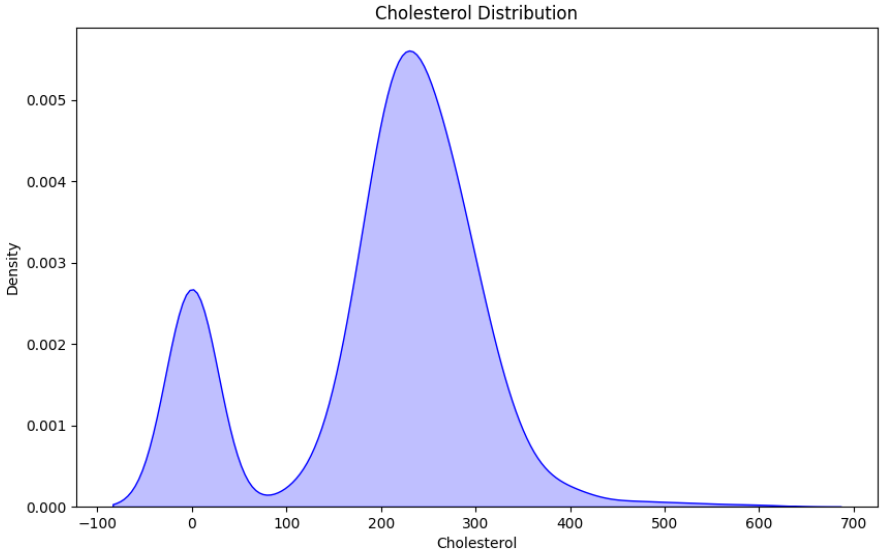


Figure 2: The KDE plot for Cholesterol highlights the distribution of cholesterol levels.

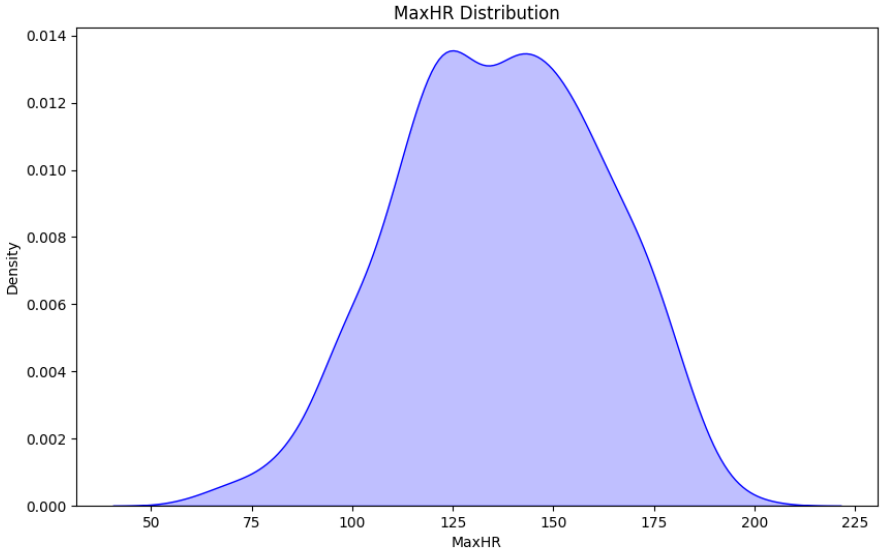


Figure 3: The KDE plot for MaxHR illustrates the distribution of maximum heart rates.

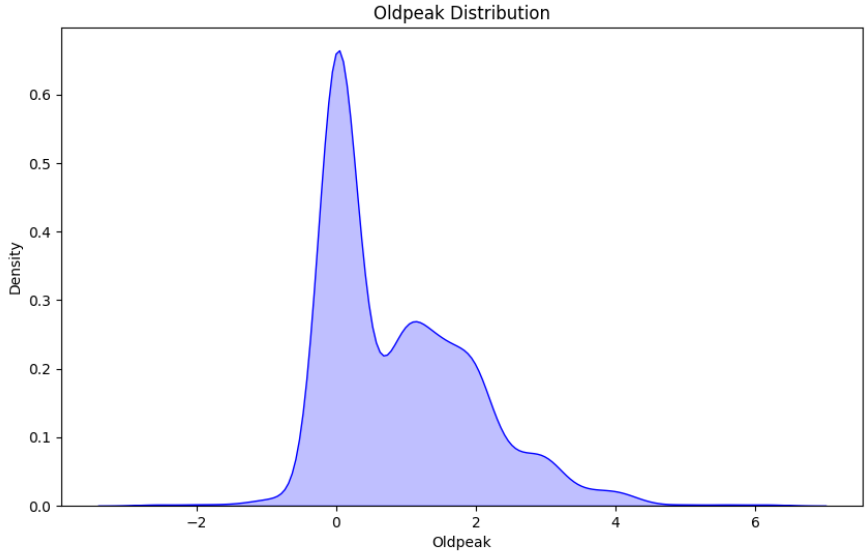


Figure 4: The KDE plot for Oldpeak demonstrates the distribution of the depression induced by exercise.

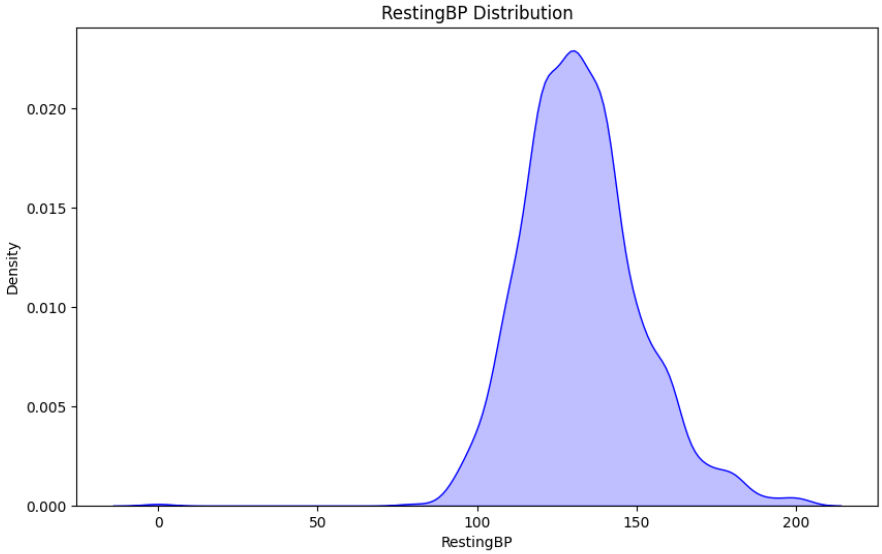


Figure 5: The KDE plot for RestingBP shows the distribution of resting blood pressure values.

3.2.2 PIE Charts

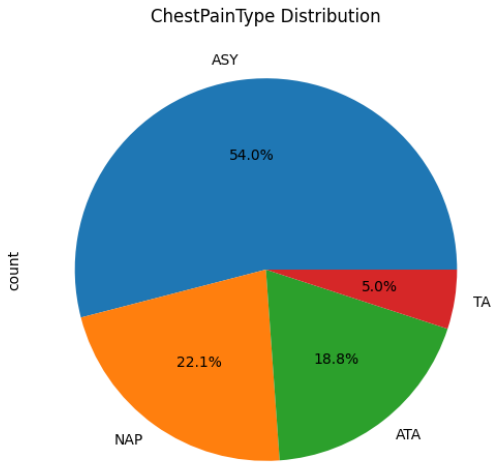


Figure 6: The pie chart shows the distribution of Chest Pain Types.

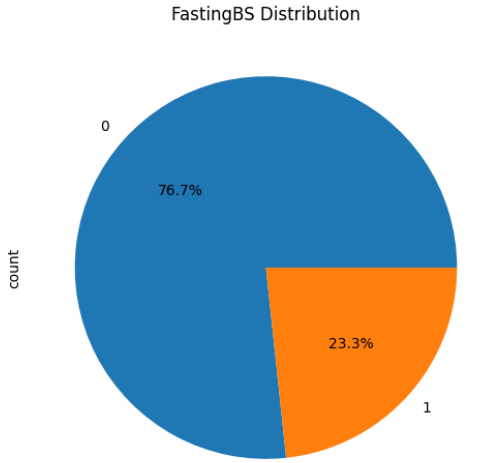


Figure 7: The pie chart for FastingBS shows the distribution of fasting blood sugar levels.

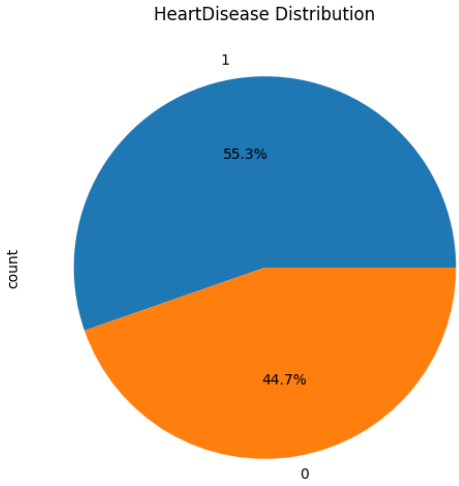


Figure 8: The pie chart shows the distribution of Heart Disease occurrences in the dataset.

3.2.3 Bar Plots

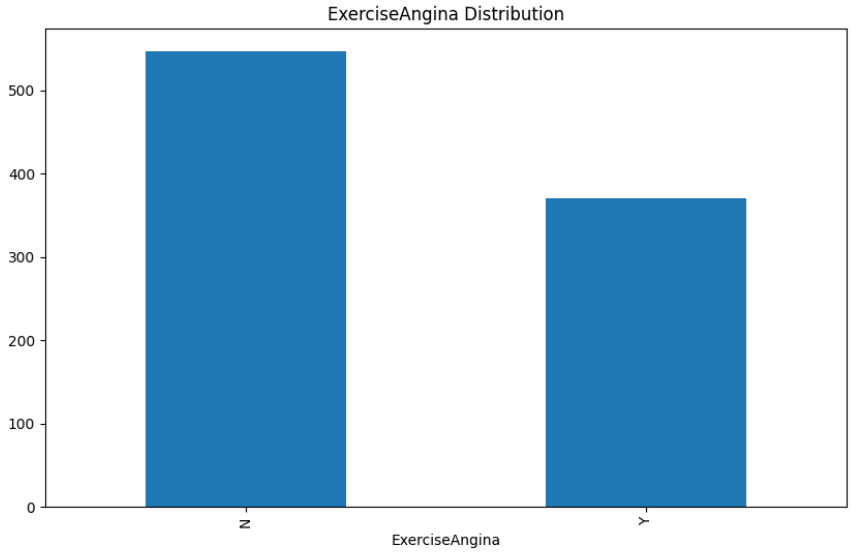


Figure 9: The bar plot shows the distribution of Exercise Angina occurrences in the dataset.

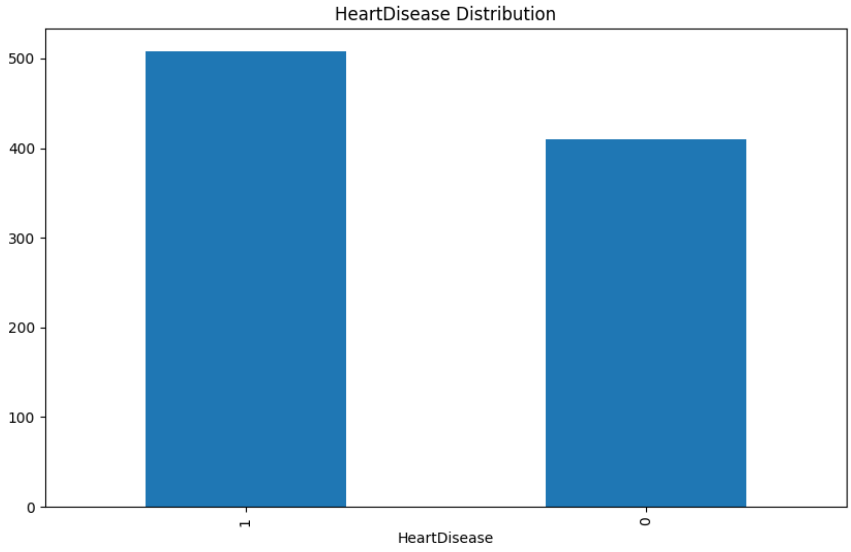


Figure 10: The bar plot displays the count of Heart Disease cases.

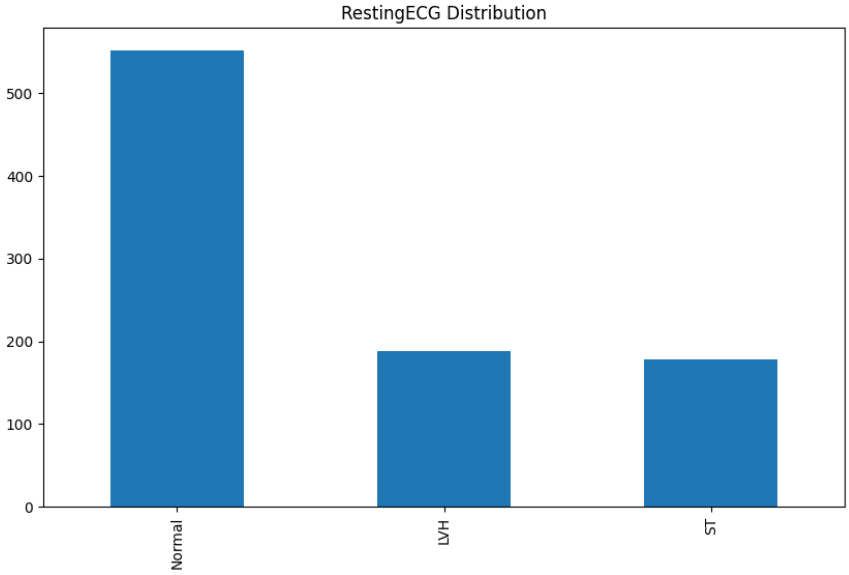


Figure 11: The bar plot visualizes the different RestingECG types in the dataset.

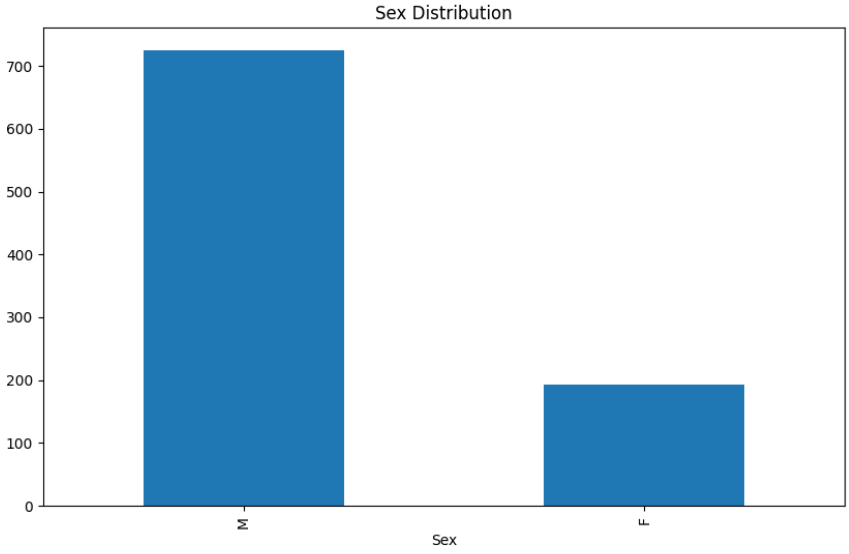


Figure 12: The bar plot shows the distribution of male and female subjects in the dataset.

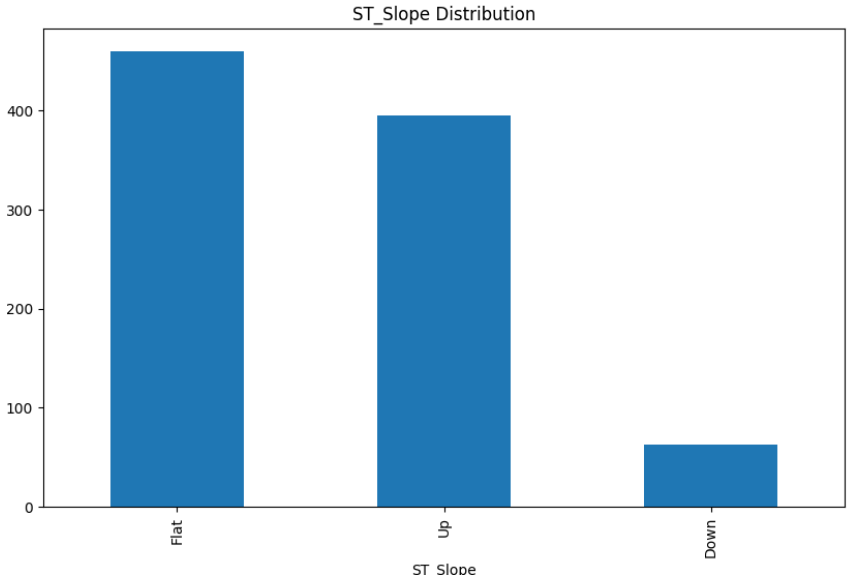


Figure 13: The bar plot for ST Slope visualizes the different slope values in the dataset.

3.2.4 Box Plots

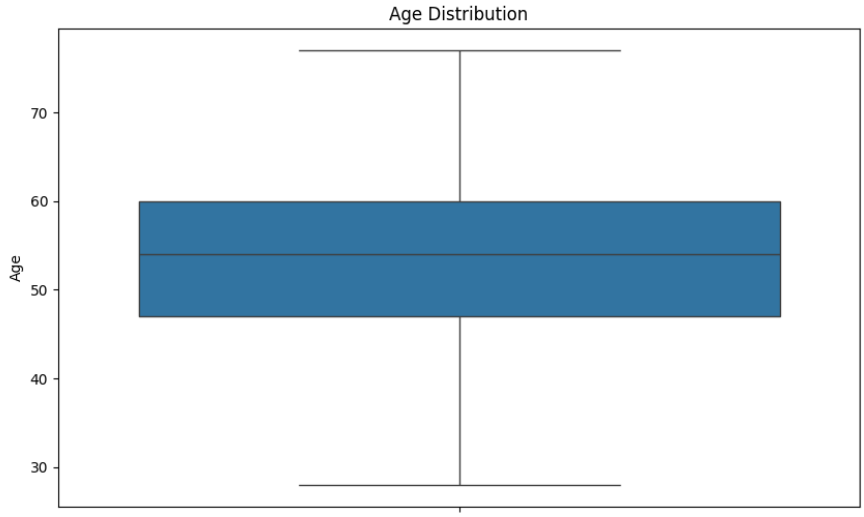


Figure 14: The box plot for Age shows the spread and outliers of age distribution.

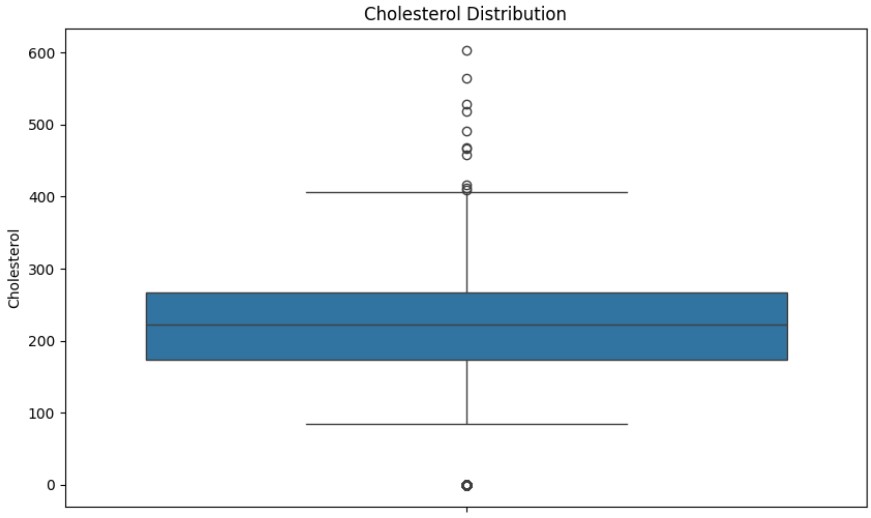


Figure 15: The box plot for Cholesterol illustrates the spread and outliers in cholesterol levels.

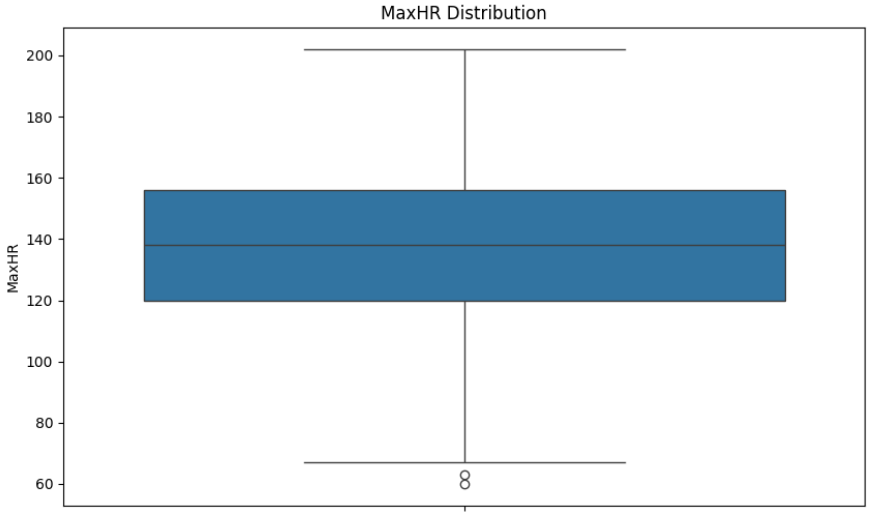


Figure 16: The box plot for MaxHR shows the range and outliers of maximum heart rate values.

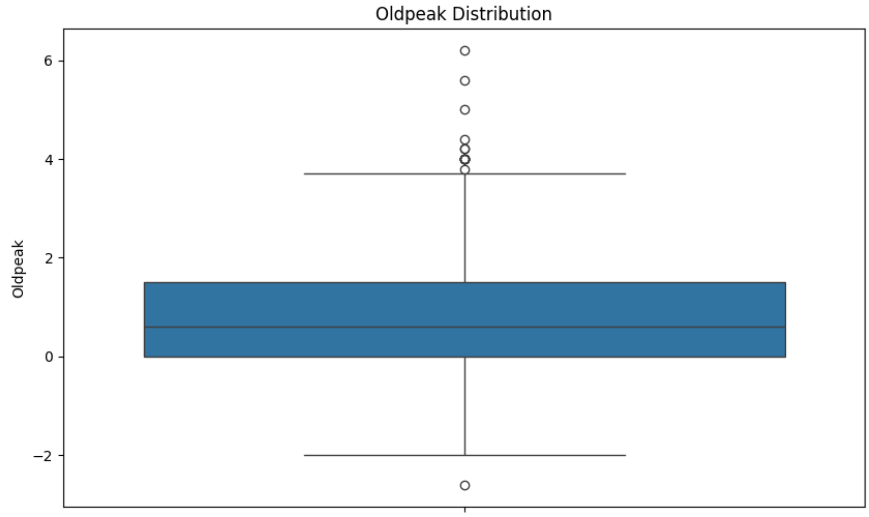


Figure 17: The box plot for Oldpeak illustrates the distribution and outliers of depression induced by exercise.

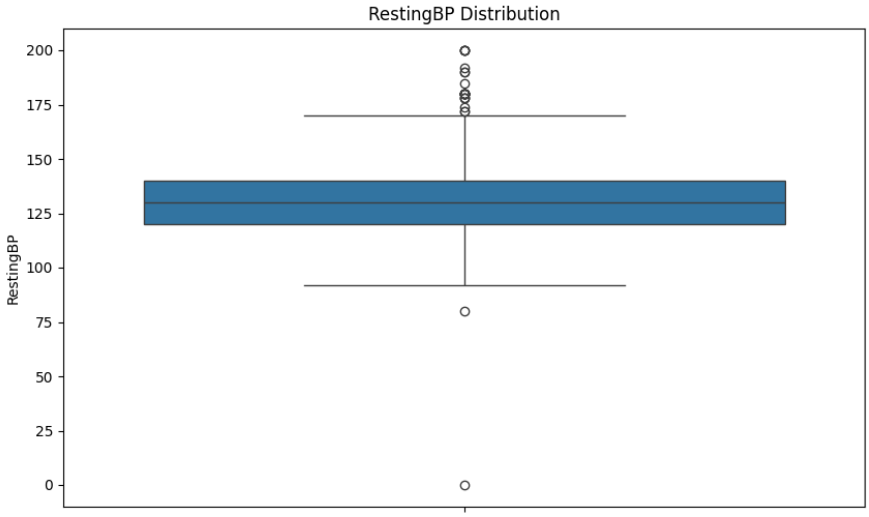


Figure 18: The box plot for RestingBP highlights the distribution and outliers of resting blood pressure.

3.2.5 Scatter Plots

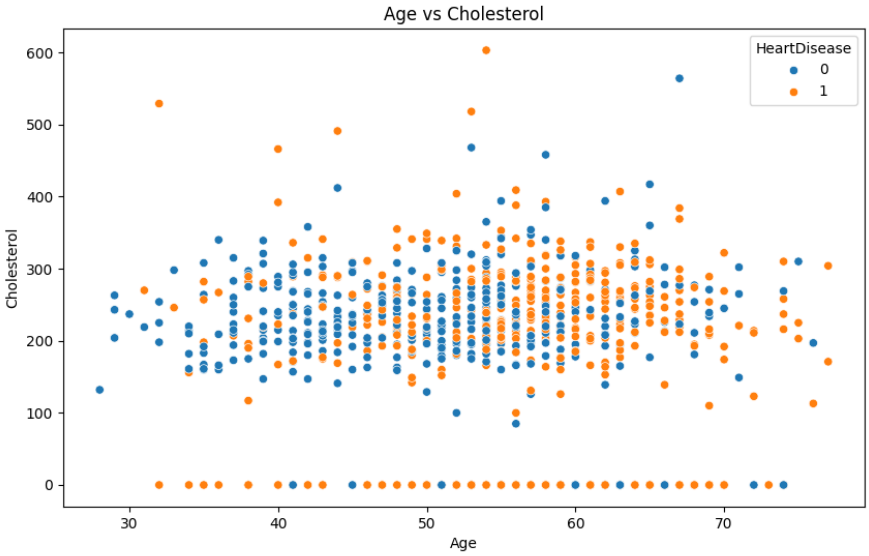


Figure 19: Scatter plot showing the relationship between Age and Cholesterol levels.

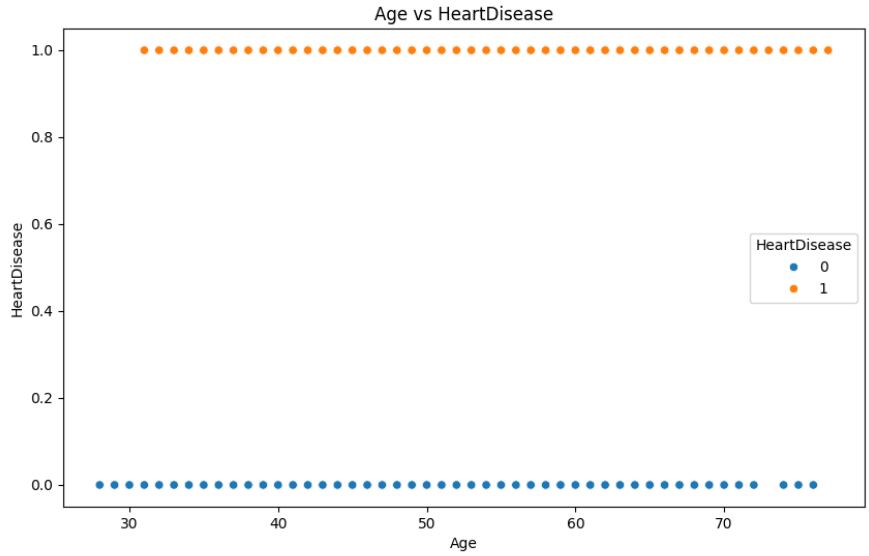


Figure 20: Scatter plot depicting the correlation between Age and Heart Disease occurrences.

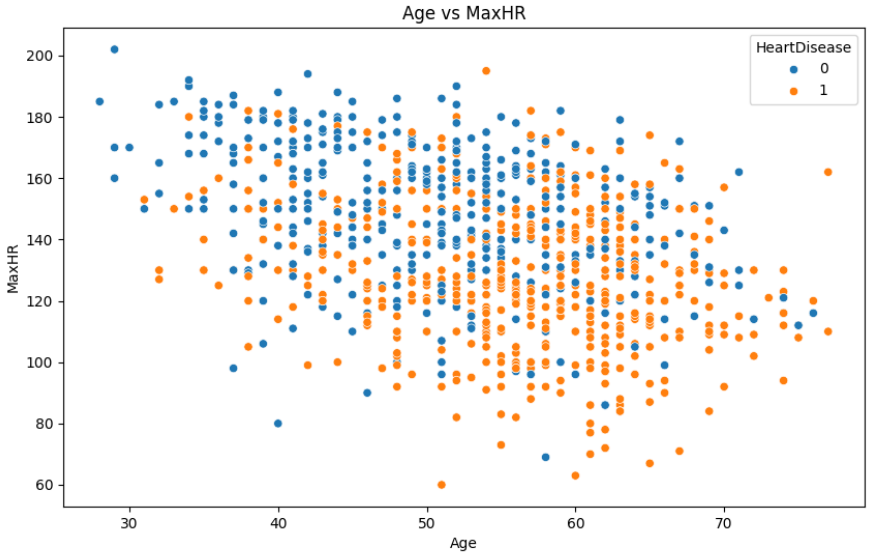


Figure 21: Scatter plot visualizing the relationship between Age and MaxHR.

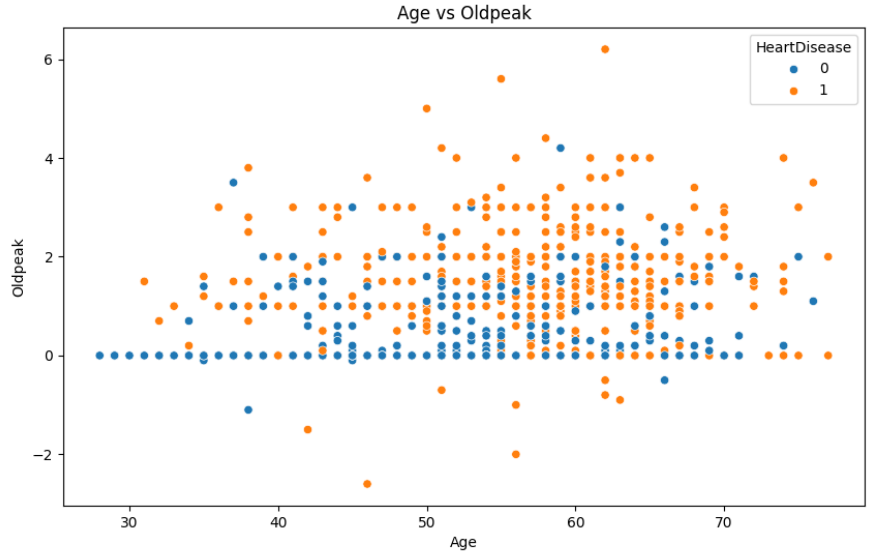


Figure 22: Scatter plot showing the relationship between Age and Oldpeak values.

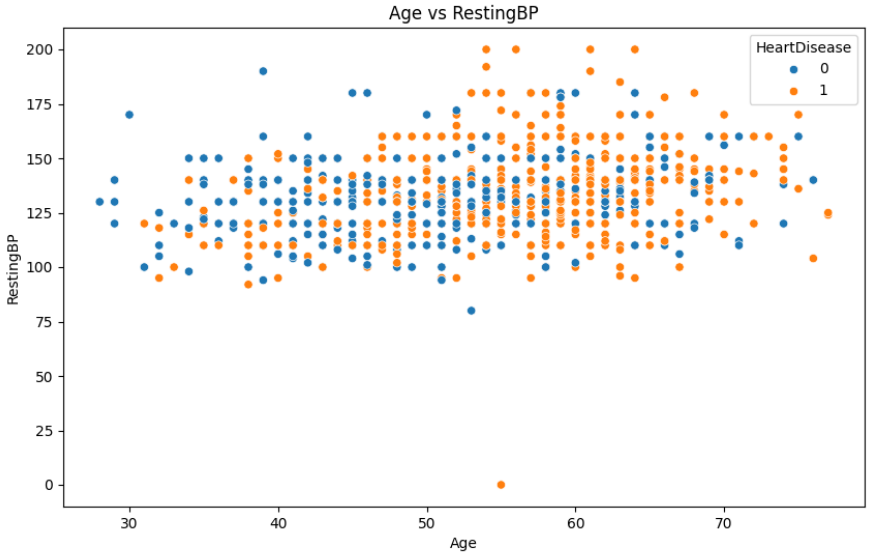


Figure 23: Scatter plot illustrating the relationship between Age and RestingBP.

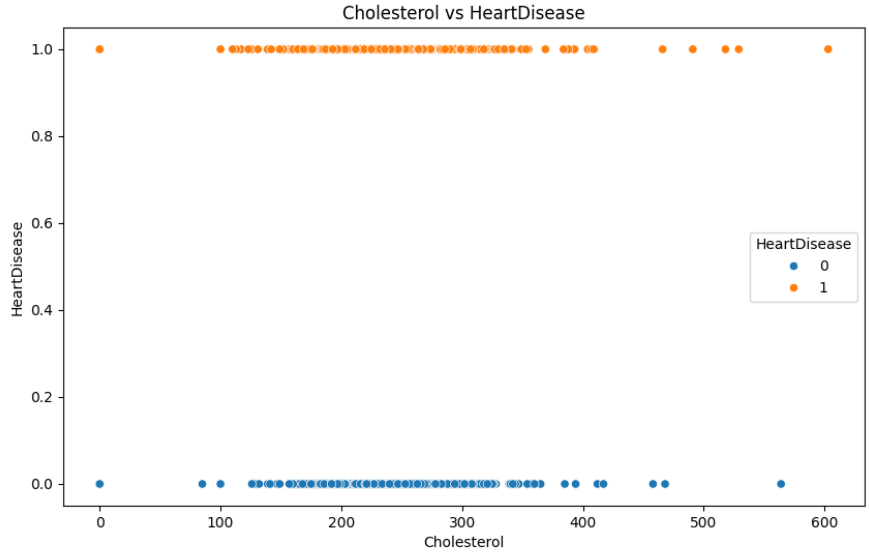


Figure 24: Scatter plot showing the correlation between Cholesterol levels and Heart Disease.

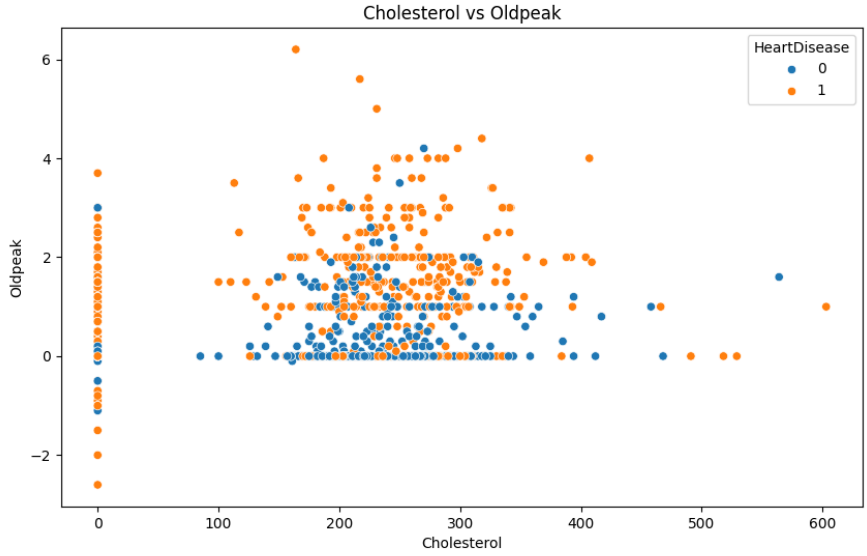


Figure 25: Scatter plot depicting the relationship between Cholesterol levels and Oldpeak values.

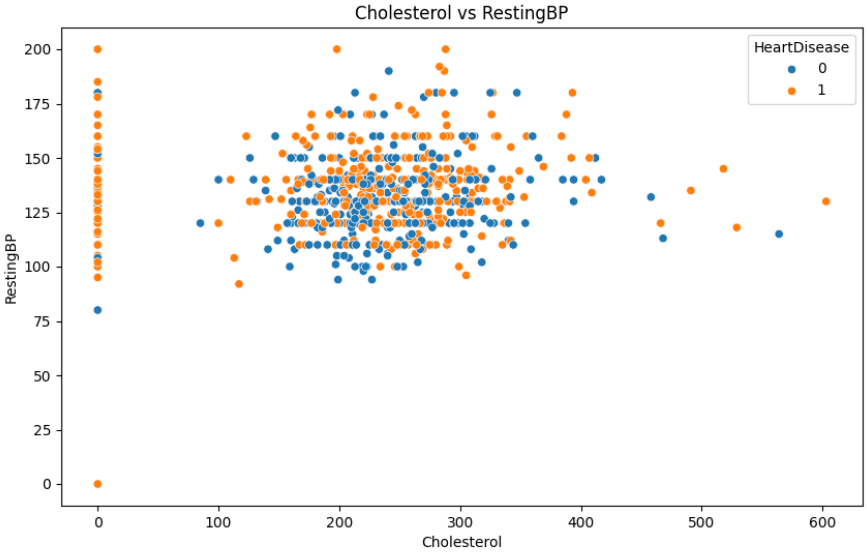


Figure 26: Scatter plot visualizing the relationship between Cholesterol levels and RestingBP.

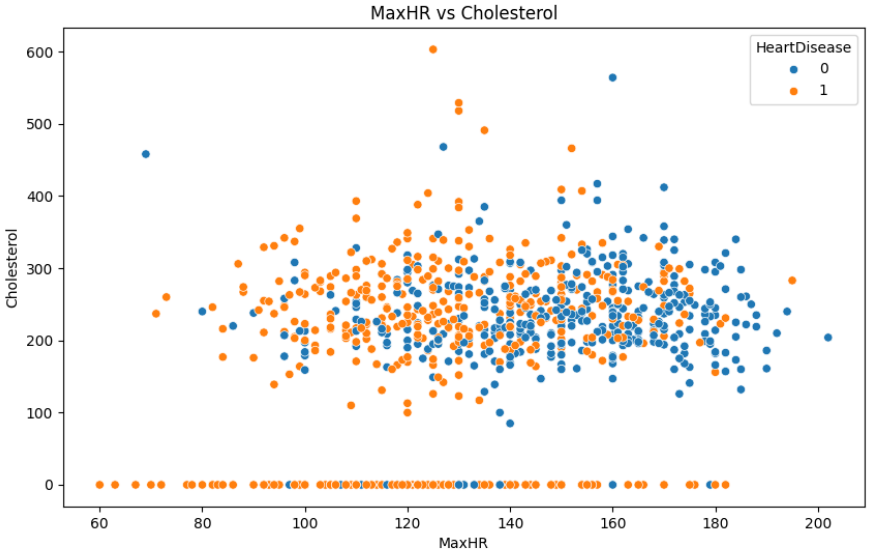


Figure 27: Scatter plot showing the relationship between MaxHR and Cholesterol levels.

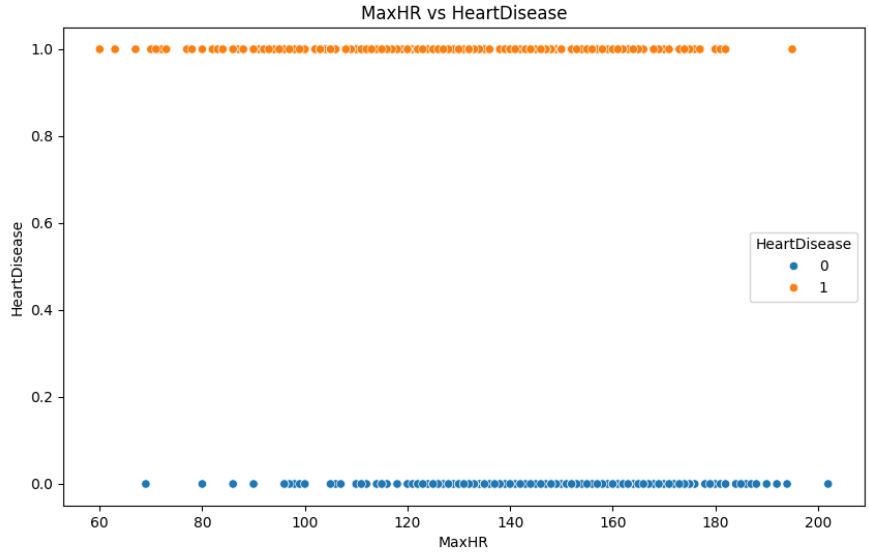


Figure 28: Scatter plot depicting the relationship between MaxHR and Heart Disease.

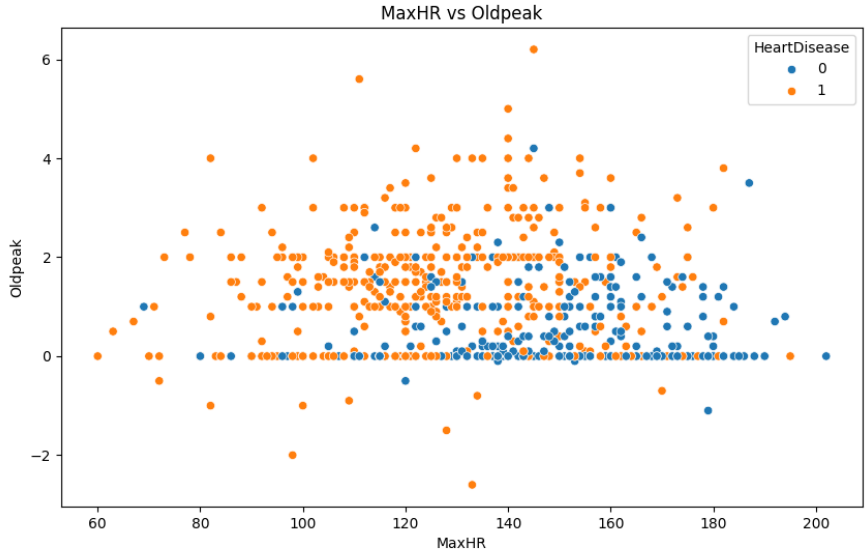


Figure 29: Scatter plot illustrating the relationship between MaxHR and Oldpeak values.

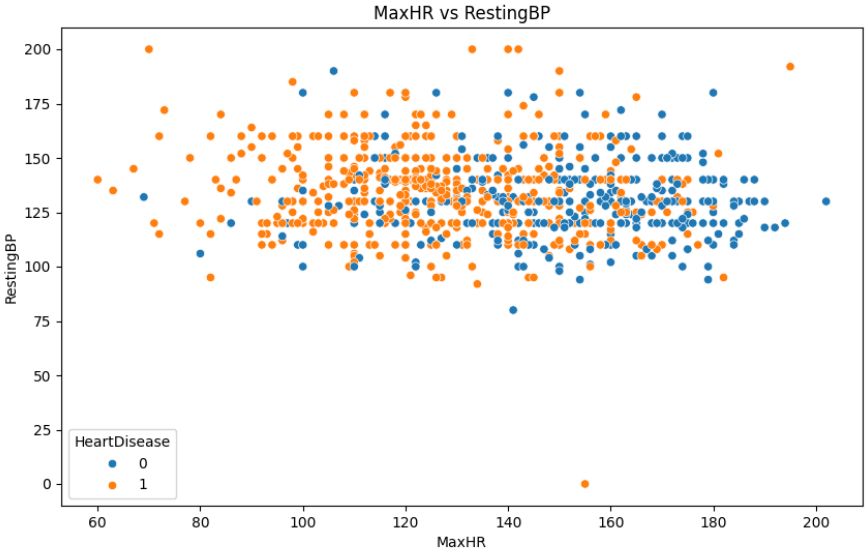


Figure 30: Scatter plot visualizing the relationship between MaxHR and RestingBP.

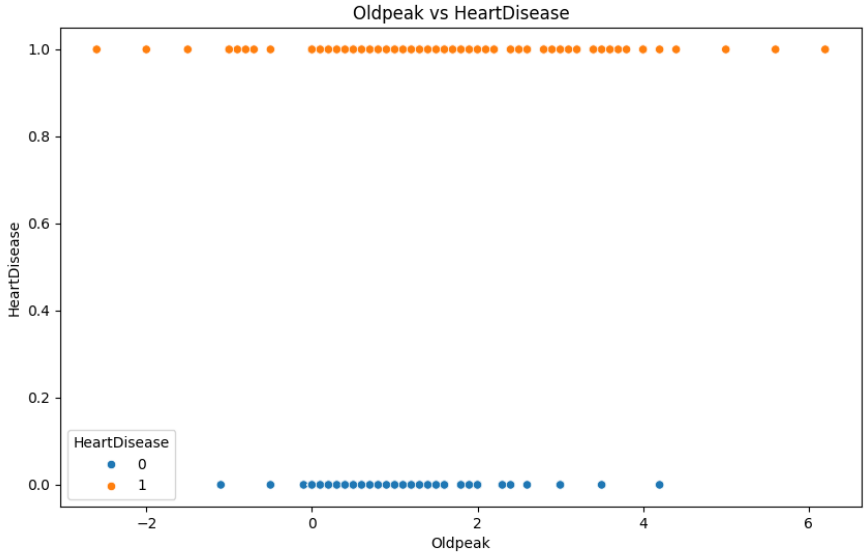


Figure 31: Scatter plot showing the correlation between Oldpeak and Heart Disease.

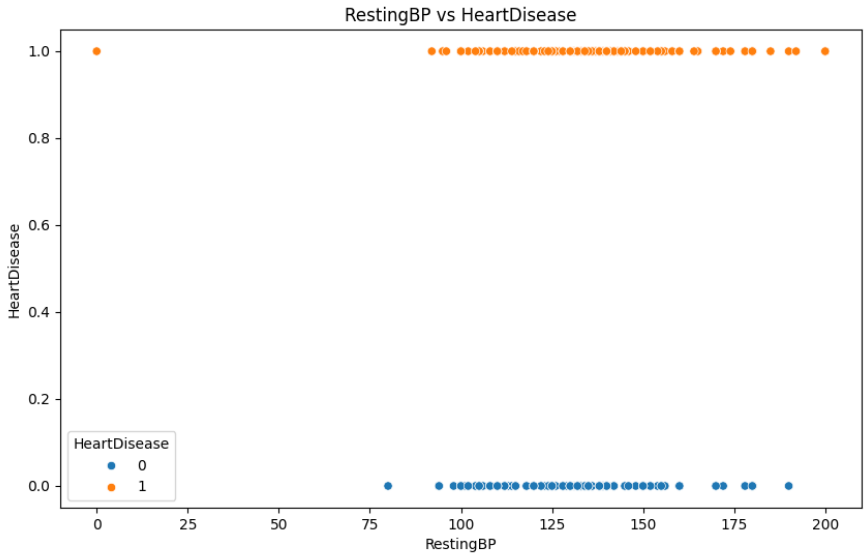


Figure 32: Scatter plot visualizing the relationship between RestingBP and Heart Disease.

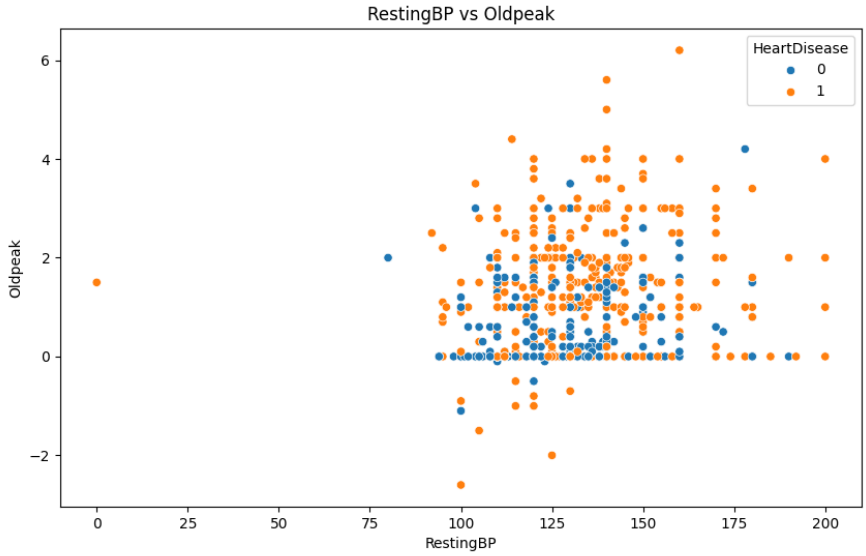


Figure 33: Scatter plot illustrating the relationship between RestingBP and Oldpeak.

3.3 Correlation of Numerical Features

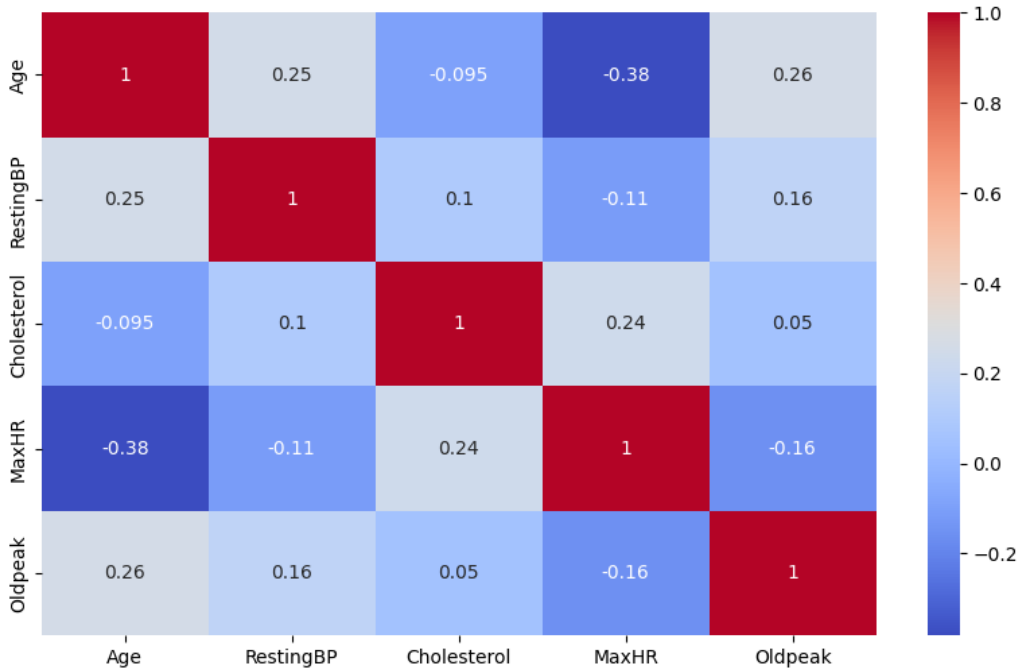


Figure 34: Correlation of Numerical Features

From the Matrix, we can see there is no strong correlation between any numerical features.

4 Data Cleaning and Preprocessing

4.1 Scenarios

To explore the dataset more effectively and evaluate the performance of various models under different conditions, we created **multiple scenarios**. Each scenario corresponds to a unique subset or manipulation of the data:

- **Scenarios 1 to 8:** Represent different configurations of the dataset, such as adjusted feature sets, modified data distributions, or specific filtering conditions.
- **N and S Variants:** Each scenario is divided into two variations:
  - **N:** Normalized version of the dataset.
  - **S:** Scaled version of the dataset.

4.1.1 Detailed Scenarios Description

Below is a comprehensive breakdown of each scenario and its corresponding goals:

Scenario 1: Baseline Dataset

- **N (Normalized):** All features are normalized to values between 0 and 1, ensuring consistency across features.
- **S (Scaled):** Features are standardized to have a mean of 0 and a standard deviation of 1.

Purpose: Establish a baseline performance using basic normalization and scaling techniques.

Scenario 2: Feature Selection

- **N (Normalized):** Irrelevant features are removed based on exploratory data analysis, with normalization applied to the remaining features.
- **S (Scaled):** Features are reduced and scaled to ensure effectiveness with algorithms like SVM.

Purpose: Analyze how feature reduction affects model accuracy and efficiency.

Scenario 3: Missing Value Handling

- **N (Normalized):** Missing values are imputed using statistical methods (e.g., mean or median) and normalized.
- **S (Scaled):** Similar imputations are followed by scaling.

Purpose: Assess the impact of different missing value treatments.

Scenario 4: Outlier Removal

- **N (Normalized):** Outliers are removed based on Z-scores or interquartile ranges and normalized.
- **S (Scaled):** Outlier removal is followed by scaling.

Purpose: Investigate the effect of outliers on model performance.

Scenario 5: Feature Engineering

- **N (Normalized):** Additional features are created and normalized.
- **S (Scaled):** Engineered features are scaled for compatibility with distance-based models.

Purpose: Explore whether feature engineering enhances predictive power.

Scenario 6: Duplicate Removal

- **N (Normalized):** Duplicates are removed, and normalization is applied.
- **S (Scaled):** Duplicates are removed, followed by scaling.

Purpose: Test the impact of removing redundant data.

Scenario 7: Data Splitting

- **N (Normalized):** Data is split into training (80%), validation (10%), and testing (10%) sets, with normalization applied.
- **S (Scaled):** The same split is performed, with scaling instead of normalization.

Purpose: Compare the influence of data splitting strategies under normalized and scaled conditions.

Scenario 8: Advanced Feature Engineering

- **N (Normalized):** Complex features are created and normalized.
- **S (Scaled):** Advanced features are scaled for model evaluation.

Purpose: Evaluate the benefits of advanced feature engineering techniques.

4.1.2 Overall Purpose of Scenarios

These scenarios were designed to:

- Test different preprocessing strategies (e.g., normalization, scaling, feature selection).
- Assess model robustness under varying dataset conditions.
- Provide comprehensive insights into the effects of data transformations on machine learning performance.

4.1.3 Dataset Characteristics in Scenarios

The following table provides an overview of the dataset characteristics for each scenario (1 to 8), along with their corresponding normalized (N) and scaled (S) versions. The key metrics include the number of rows, columns, missing values, duplicate rows, numeric columns, and categorical columns, which collectively describe the structure and quality of the data across various preprocessing stages.

- **Scenario 1 to Scenario 4:**
  - **Encoding:** Label encoding was applied to all categorical columns.
  - **Scaling:**
    - Scenario 1 and Scenario 2: Normalized using Min-Max Scaling.
    - Scenario 3 and Scenario 4: Standard Scaling.
  - **Error Handling:**
    - Scenario 1: No outlier removal or error handling.
    - Scenario 2: Blunders (e.g., invalid values) were fixed without removing outliers.
    - Scenario 3: Outliers were removed using the IQR method; blunders were then fixed.
    - Scenario 4: Outliers were replaced with mean values; blunders were then fixed.

- **Scenario 5 to Scenario 8:**
  - **Encoding:**
    - One-Hot encoding was applied to the `ChestPainType` column.
    - Label encoding was applied to the remaining categorical columns.
  - **Scaling:**
    - Scenario 5 and Scenario 6: Normalized using Min-Max Scaling.
    - Scenario 7 and Scenario 8: Standard Scaling.
  - **Error Handling:**
    - Scenario 5: No outlier removal or error handling.
    - Scenario 6: Blunders (e.g., invalid values) were fixed without removing outliers.
    - Scenario 7: Outliers were removed using the IQR method; blunders were then fixed.
    - Scenario 8: Outliers were replaced with mean values; blunders were then fixed.

These scenarios were designed to evaluate the impact of different encoding strategies, scaling methods, and error handling techniques on model performance.

Table 3: Dataset Characteristics Across Different Scenarios and Variants (N: Normalized, S: Scaled)

Scenario	Rows	Columns	Numeric Columns
1_N	918	12	12
1_S	918	12	12
2_N	918	12	12
2_S	918	12	12
3_N	702	12	12
3_S	702	12	12
4_N	918	12	12
4_S	918	12	12
5_N	918	15	15
5_S	918	15	15
6_N	918	15	15
6_S	918	15	15
7_N	702	15	15
7_S	702	15	15
8_N	918	15	15
8_S	918	15	15

Table 4: Dataset Characteristics Across Different Scenarios and Versions (N: Normalized, S: Scaled)

4.2 Handling Missing Values

4.2.1 Observations

- Across all scenarios, there are no missing values in any dataset (both N and S variants).
- This consistency suggests that missing value handling has been performed prior to these analyses, ensuring that models can be trained without interruptions or biases due to incomplete data.
- Scenario 3 specifically focused on imputing missing values using statistical methods, such as the mean or median for numerical attributes. This step was critical to ensure completeness and consistency in the dataset.

4.2.2 Impact on Scenarios

- For Scenario 3, the imputation process ensures uniformity, allowing the normalized and scaled datasets to remain consistent with the structure of other scenarios.
- As missing values were handled in early stages, downstream processes like feature engineering and scaling could proceed without additional adjustments.

4.3 Outlier Detection and Treatment

4.3.1 Observations

- Outliers were addressed specifically in Scenario 4, where statistical techniques like Z-scores or interquartile ranges (IQR) were applied to detect and remove anomalies.
- After outlier removal, the number of rows in the dataset remained consistent across normalized and scaled versions, ensuring comparability.

4.3.2 Impact on Scenarios

- The removal of outliers enhances the reliability of machine learning models by reducing the influence of extreme values.
- Features impacted by outliers, such as age or cholesterol levels, are now more representative of typical patient values, improving model accuracy and generalization.

4.4 Duplicate Record Removal

4.4.1 Observations

- There are no duplicate rows across any of the datasets in all scenarios (N and S), as identified during the analysis.
- Scenario 6 explicitly focuses on duplicate record removal, ensuring the datasets used for training are free from redundancy.

4.4.2 Impact on Scenarios

- The absence of duplicates ensures that models do not overfit due to repetitive data patterns.
- Removing duplicates also enhances computational efficiency, particularly for scenarios involving large datasets or complex transformations.

4.5 Data Splitting (Training, Validation, Testing)

4.5.1 Observations

- Scenario 7 introduces an 80-10-10 split for training, validation, and testing sets, ensuring a balanced and representative dataset for each stage of the machine learning pipeline.
- This splitting strategy is applied to both normalized and scaled versions of the dataset, enabling fair comparisons across preprocessing techniques.

4.5.2 Impact on Scenarios

- Proper splitting avoids data leakage and ensures unbiased model evaluation.
- The balanced split across training, validation, and testing ensures robust performance metrics, allowing accurate comparisons of classifier performance in subsequent stages.

4.6 General Observations

- **Rows and Columns:**
  - The number of rows varies between scenarios, with 702 rows in Scenario 3 due to missing value handling and a decrease in Scenario 7 due to data splitting.
  - The number of columns increases from 12 to 15 starting in Scenario 5, reflecting the addition of engineered features.
- **Missing Values:**
  - All datasets across the scenarios have zero missing values, indicating comprehensive preprocessing during early stages.
- **Duplicate Rows:**
  - No duplicate rows are present in any scenario, ensuring data integrity and representativeness.
- **Numeric and Categorical Columns:**
  - The number of numeric columns remains at 12 for Scenarios 1 to 4, increasing to 15 from Scenario 5 onwards, indicating feature engineering.
  - No categorical columns are present in any dataset, ensuring all features are numeric and suitable for machine learning models.

5 Model Implementation

5.1 Description of Models

This section outlines the machine learning models implemented in the project. Each model was chosen based on its ability to handle the dataset and solve the classification problem effectively. The descriptions below detail the algorithms, their characteristics, and the hyperparameters optimized during the study.

5.1.1 Naive Bayes

Naive Bayes is a probabilistic classifier that applies **Bayes' Theorem** with the assumption of feature independence. It is lightweight and effective for large datasets.

Key Features:
- Ideal for categorical and numerical data.
- Performs well with high-dimensional datasets.

Optimization Results:
The configuration achieved using either **Grid**, **Random**, **Optuna Optimization** where there accuracies are the same with value `85.86956521739131%` on same scenario `Scenario 1_N`.

Hyperparameters Tuned:

1. In **Grid** the parameters tuned are `var_smoothing: 1e-09, priors: null`.
2. In **Random** the parameters tuned are `priors: null, var_smoothing: 1e-09`.
3. In **Optuna Optimization** the parameters tuned are `var_smoothing: 3.7269867644642197e-06`.

5.1.2 SVM (Support Vector Machine)

SVM is a robust classifier that identifies a hyperplane to separate classes effectively, often used for binary or multi-class classification problems.

Key Features:
- Works well with high-dimensional datasets.
- Offers flexibility through kernel functions (e.g., linear, RBF, polynomial).

Optimization Results:
The configuration achieved using either **Grid**, **Random**, **Optuna Optimization** where there accuracies are `89.13043478260869%`, `89.13043478260869%`, `90.21739130434783%` on same scenario `Scenario 1_N` respectively.

Hyperparameters Tuned:

1. In **Grid** the parameters tuned are `kernel: "poly", gamma: 1, C: 0.1`.
2. In **Random** the parameters tuned are `kernel: "poly", gamma: 1, C: 0.1`.
3. In **Optuna Optimization** the parameters tuned are `C: 6.055144656433842, kernel: "rbf", gamma: 0.3386296212305996`.

5.1.3 KNN (K-Nearest Neighbors)

KNN is an instance-based learning algorithm that classifies data based on the nearest training samples.

Key Features:
- Non-parametric and simple to implement.
- Sensitive to the value of `k` and distance metrics.

Optimization Results:
The configuration achieved using either **Grid**, **Random**, **Optuna Optimization** where there accuracies are `89.13043478260869%`, `89.13043478260869%`, `90.21739130434783%` on scenarios `Scenario 8_S`, `Scenario 8_S`, `Scenario 1_S` respectively.

Hyperparameters Tuned:

1. In **Grid** the parameters tuned are `metric: "manhattan", n_neighbors: 9, weights: "distance"`.
2. In **Random** the parameters tuned are `weights: "distance", n_neighbors: 9, metric: "manhattan"`.
3. In **Optuna Optimization** the parameters tuned are `n_neighbors: 10, weights: "distance", metric: "manhattan"`.

5.1.4 Decision Trees

Decision Trees use a hierarchical structure to split data based on feature values, making them interpretable and flexible.

Key Features:
- Handles both numerical and categorical data.
- Prone to overfitting without constraints like pruning or depth limits.

Optimization Results:
The configuration achieved using either **Grid**, **Random**, **Optuna Optimization** where there accuracies are `83.69565217391305%`, `82.60869565217391%`, `83.69565217391305%` on scenarios `Scenario 4_S`, `Scenario 8_S`, `Scenario 5_S` respectively.

Hyperparameters Tuned:

1. In **Grid** the parameters tuned are `criterion: "log_loss", max_depth: 30, min_samples_leaf: 1, min_samples_split: 2`.
2. In **Random** the parameters tuned are `min_samples_split: 2, min_samples_leaf: 1, max_depth: 30, criterion: "log_loss"`.
3. In **Optuna Optimization** the parameters tuned are `criterion: "log_loss", max_depth: 6, min_samples_split: 6, min_samples_leaf: 3`.

5.2 Training Procedures and Hyperparameter Tuning

The training procedures and hyperparameter tuning process were methodically designed and implemented to optimize the performance of the selected machine learning models. Below is a detailed description of the steps and methodologies employed:

5.2.1 Preprocessing Scenarios

To ensure robust model training, a variety of preprocessing techniques were implemented across different scenarios:

- **Encoding Methods**:
  - *Label Encoding*: Used for ordinal or categorical features, such as `Sex` and `ExerciseAngina`, to assign unique integers to categories.
  - *One-Hot Encoding*: Applied to non-ordinal features like `ChestPainType` to create binary columns for each category.
- **Error Handling**:
  - Identified blunders in the dataset, such as negative values for `Oldpeak` or zero values in `Cholesterol` and `RestingBP`, which were replaced with meaningful values (e.g., mean).
- **Outlier Detection and Treatment**:
  - Outliers were detected using the IQR method. In some scenarios, they were removed entirely, while in others, they were replaced with the mean.
- **Scaling**:
  - Two scaling methods, MinMax Scaling and Standard Scaling, were applied to standardize the feature ranges and improve algorithm compatibility.

Each scenario was saved as a unique dataset to enable a fair comparison of different preprocessing techniques.

5.2.2 Model Selection

Four machine learning algorithms were tested, each with distinct characteristics:

- **Naive Bayes (GaussianNB)**:
  - A probabilistic model suitable for features with Gaussian distributions.
- **Support Vector Machines (SVM)**:
  - A powerful algorithm for both linear and non-linear classification using kernel methods.
- **K-Nearest Neighbors (KNN)**:
  - A simple, distance-based algorithm that classifies based on the majority class of neighbors.
- **Decision Trees**:
  - A flexible, interpretable model that splits data hierarchically based on feature values.

5.2.3 Hyperparameter Tuning

To achieve optimal model performance, three hyperparameter tuning methods were utilized:

- **Grid Search**:
  - Exhaustively searched the parameter space to find the best combination of hyperparameters.
  - Effective for small parameter spaces but computationally expensive for larger configurations.
- **Random Search**:
  - Randomly sampled hyperparameters from predefined distributions, striking a balance between exploration and efficiency.
- **Bayesian Optimization (Optuna)**:
  - Used a probabilistic model to guide the search for hyperparameters, focusing on areas of the parameter space likely to yield better performance.

Each tuning method was applied to the selected algorithms, and the best-performing hyperparameters were stored along with the results.

5.2.4 Evaluation Metrics

To assess the performance of the models, several metrics were calculated:

- **Accuracy**:
  - The proportion of correctly classified instances.
- **Precision, Recall, and F1-Score**:
  - Evaluated for multi-class classification tasks to measure the trade-offs between false positives and false negatives.
- **Confusion Matrix**:
  - Provided a detailed breakdown of predictions versus actual labels, aiding in error analysis.

5.2.5 Visualization of Results

- **Accuracy Comparisons**:
  - Bar charts and line plots were used to compare the accuracies of different models across scenarios and optimization methods.
- **PCA Visualization**:
  - Principal Component Analysis was employed to visualize data in 2D and 3D, highlighting feature separability and distribution patterns.
- **Confusion Matrix Heatmaps**:
  - Illustrated the classification performance of the models with color-coded matrices for clarity.

5.2.6 Best Model Selection and Insights

- **Selection Criteria**:
  - Models were ranked based on their accuracy scores across validation and test sets, and the best model from each optimization method was selected.
- **Overall Insights**:
  - The tuning process revealed that preprocessing significantly impacted model performance, particularly the handling of outliers and scaling.
  - Optuna proved to be the most efficient method for hyperparameter tuning, often outperforming Grid Search and Random Search in terms of computational efficiency and accuracy.

5.2.7 Model Persistence and Reproducibility

- The best models were saved as `.pkl` files along with their tuning details in JSON format.
- Each model's associated preprocessing scenario and hyperparameter configuration were documented for future reproducibility.

This structured approach to training and hyperparameter tuning not only maximized the model performance but also provided valuable insights into the impact of preprocessing and optimization strategies. This methodology serves as a comprehensive framework for tackling similar machine learning tasks effectively.

5.3 Performance Metrics

To evaluate the performance of the models across different preprocessing scenarios and optimization methods, the following metrics were calculated for both validation and test datasets:

5.3.1 Accuracy

Definition: The proportion of correctly predicted instances among the total instances.
Implementation: `accuracy_score` from `sklearn.metrics` was used to calculate this metric.

5.3.2 Precision

Definition: The ratio of correctly predicted positive observations to the total predicted positives.
Purpose: Useful for minimizing false positives, especially when certain predictions have high stakes.
Implementation: Computed using `precision_score` with a weighted average to handle class imbalance.

5.3.3 Recall

Definition: The ratio of correctly predicted positive observations to all the actual positives.
Purpose: Critical for minimizing false negatives, especially in applications like healthcare or fraud detection.
Implementation: Calculated using `recall_score` with a weighted average.

5.3.4 F1-Score

Definition: The harmonic mean of precision and recall, providing a balanced measure.
Purpose: Useful for scenarios where false positives and false negatives have similar consequences.
Implementation: Derived using `f1_score`.

5.3.5 Confusion Matrix

Definition: A detailed representation of true positives, true negatives, false positives, and false negatives.
Purpose: Provides insights into specific types of misclassifications.
Visualization: Heatmaps of confusion matrices were generated using `sns.heatmap` from `seaborn`.

5.3.6 Class-Wise Metrics

Breakdown:

- Precision, Recall, and F1-Score were calculated for each class separately to evaluate per-class performance.
- Implemented using `classification_report` from `sklearn.metrics`.

5.3.7 Comparative Visualization

- **Accuracy Comparison**: Accuracy scores were plotted for all models and optimization methods, allowing easy identification of the best-performing configurations.
- **Confusion Matrix Heatmaps**: Heatmaps provided a visual interpretation of classification results, highlighting areas of confusion.
- **Class-Wise Metric Charts**: Bar charts were used to display precision, recall, and F1-scores for individual classes.

5.3.8 Insights Gained

- **Validation vs. Test Performance**: The metrics were evaluated on both validation and test sets to ensure the models generalize well.
- **Preprocessing Impact**: Scenarios with robust preprocessing showed reduced misclassification and improved metrics across the board.
- **Optimization Impact**: Models tuned using Bayesian Optimization (Optuna) consistently achieved better performance compared to Random Search and Grid Search.

The metrics provided a comprehensive evaluation of model performance, enabling the identification of the most effective preprocessing techniques and hyperparameter tuning methods for this dataset. Visual comparisons further enriched the analysis, allowing for clear and actionable insights.

5.4 Evaluation Metrics on Best Models

5.4.1 Grid Search

- Naive Bayes (Scenario_1_N):

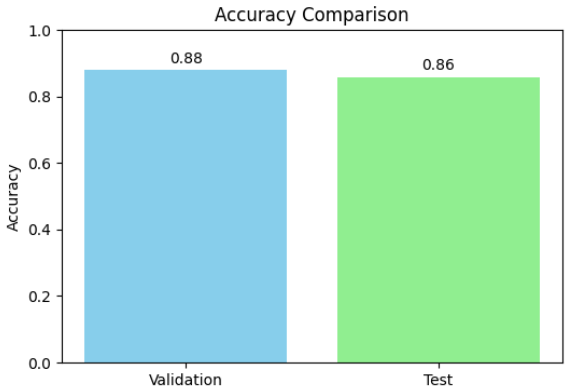


Figure 35: Accuracy comparison for Naive Bayes using Grid Search.

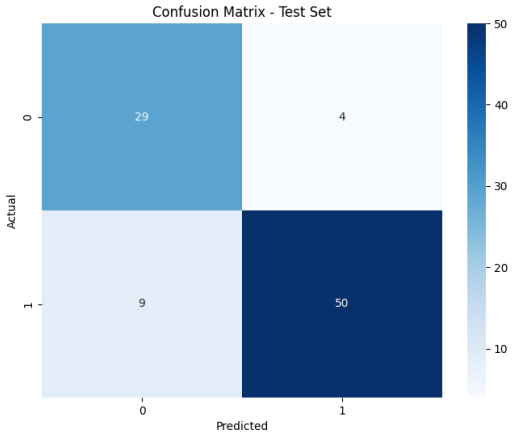


Figure 36: Confusion matrix for Naive Bayes using Grid Search.

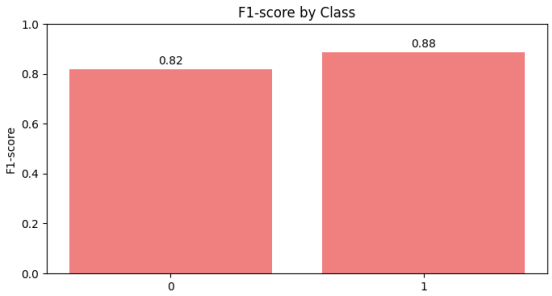


Figure 37: F1-score by class for Naive Bayes using Grid Search.

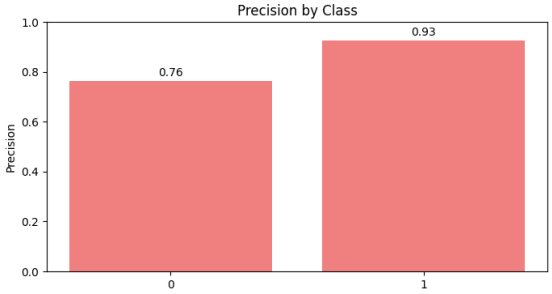


Figure 38: Precision by class for Naive Bayes using Grid Search.

Figure 39: Recall by class for Naive Bayes using Grid Search.

– SVM (Scenario_1_N):

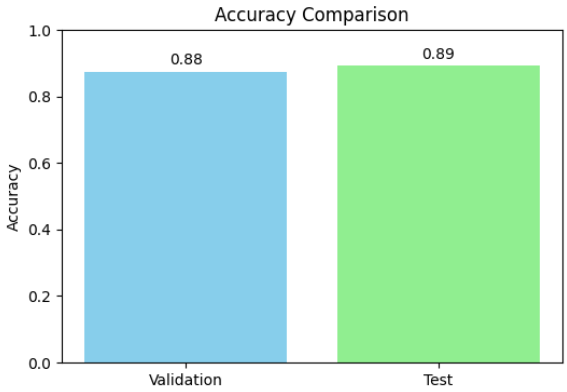


Figure 40: Accuracy comparison for SVM using Grid Search.

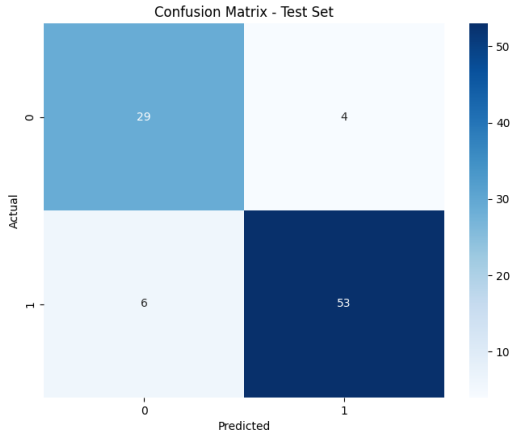


Figure 41: Confusion matrix for SVM using Grid Search.

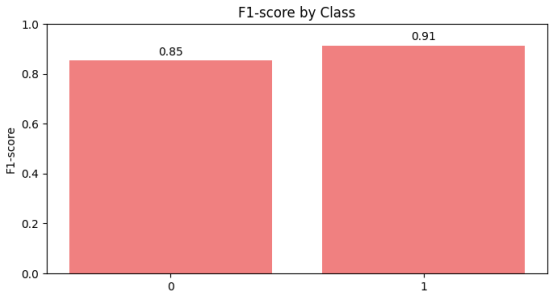


Figure 42: F1-score by class for SVM using Grid Search.

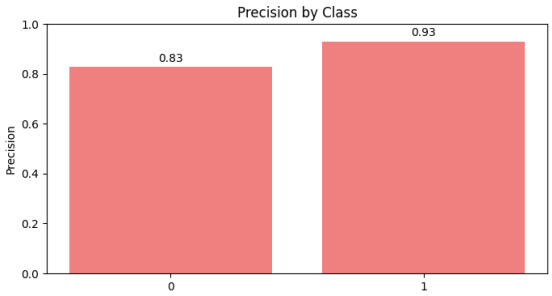


Figure 43: Precision by class for SVM using Grid Search.

Figure 44: Recall by class for SVM using Grid Search.

– KNN (Scenario_8_S):

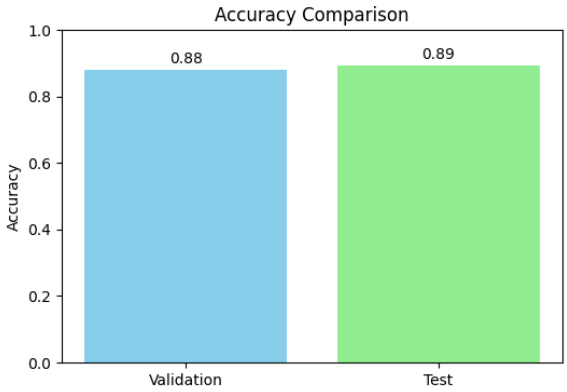


Figure 45: Accuracy comparison for KNN using Grid Search.

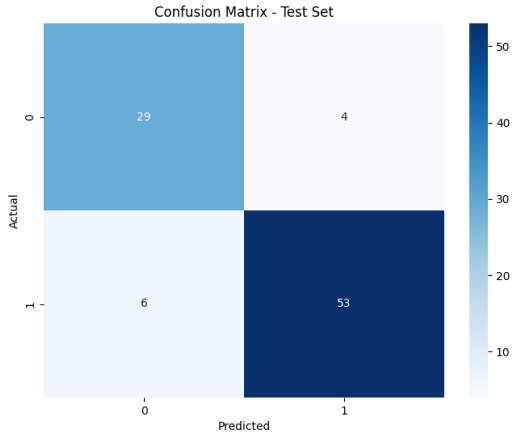


Figure 46: Confusion matrix for KNN using Grid Search.

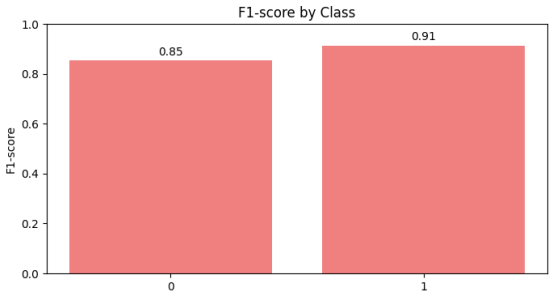


Figure 47: F1-score by class for KNN using Grid Search.

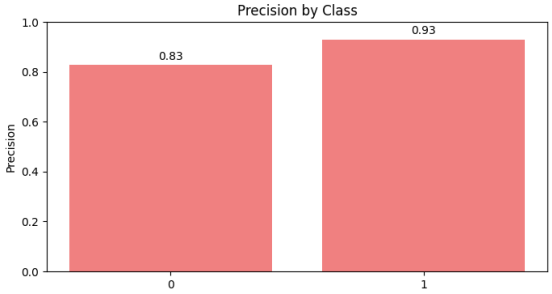


Figure 48: Precision by class for KNN using Grid Search.

Figure 49: Recall by class for KNN using Grid Search.

– Decision Tree (Scenario_4_S):

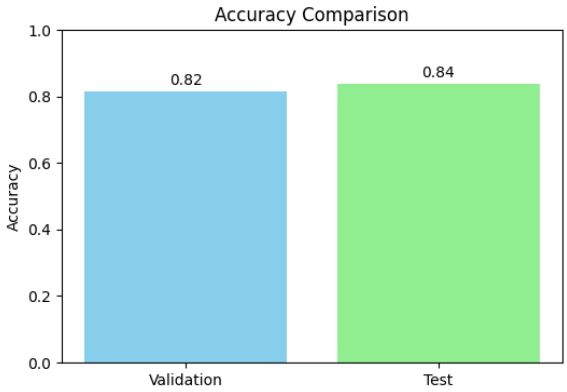


Figure 50: Accuracy comparison for Decision Tree using Grid Search.

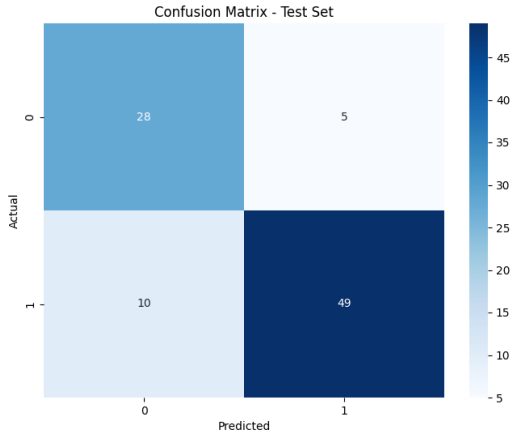


Figure 51: Confusion matrix for Decision Tree using Grid Search.

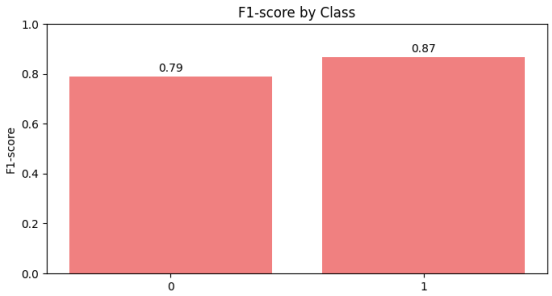


Figure 52: F1-score by class for Decision Tree using Grid Search.

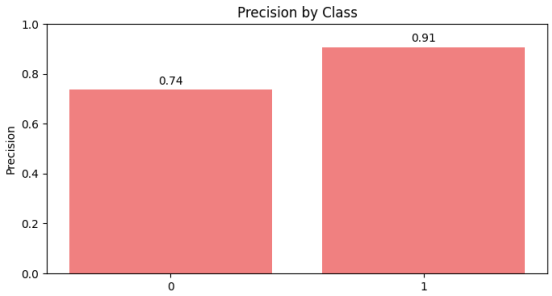


Figure 53: Precision by class for Decision Tree using Grid Search.

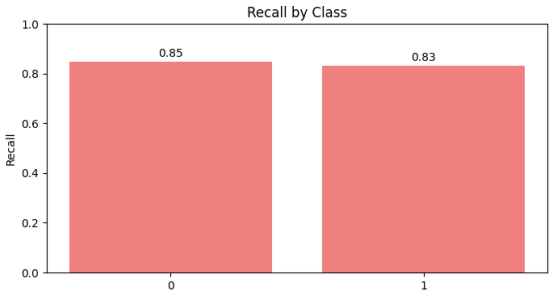


Figure 54: Recall by class for Decision Tree using Grid Search.

5.4.2 Random search

– Naive Bayes (Scenario_1_N):

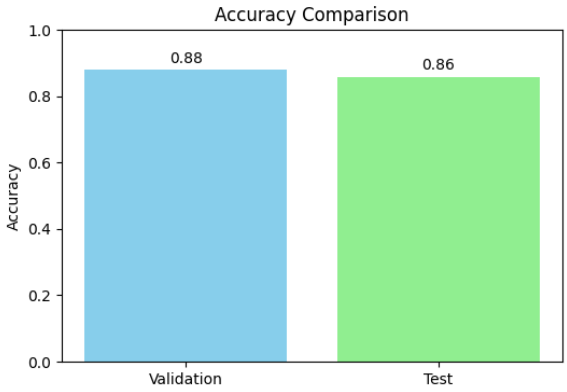


Figure 55: Accuracy comparison for Naive Bayes using Random Search.

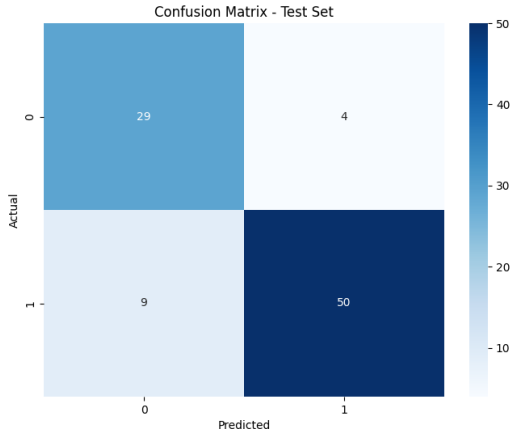


Figure 56: Confusion matrix for Naive Bayes using Random Search.

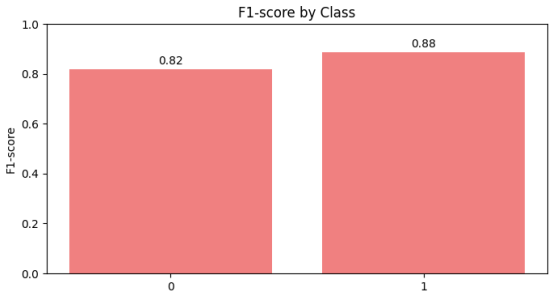


Figure 57: F1-score by class for Naive Bayes using Random Search.

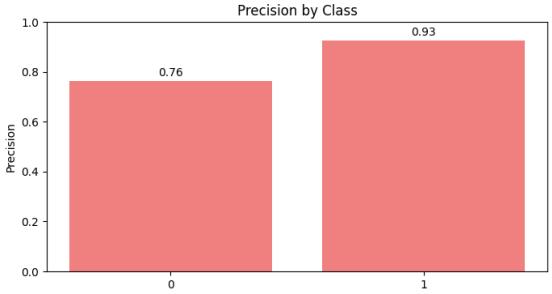


Figure 58: Precision by class for Naive Bayes using Random Search.

Figure 59: Recall by class for Naive Bayes using Random Search.

– SVM (Scenario_1_N):

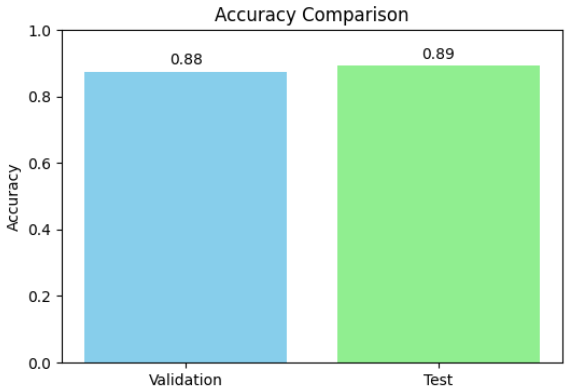


Figure 60: Accuracy comparison for SVM using Random Search.

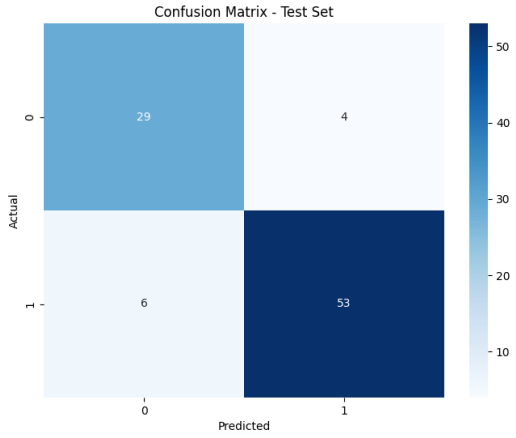


Figure 61: Confusion matrix for SVM using Random Search.

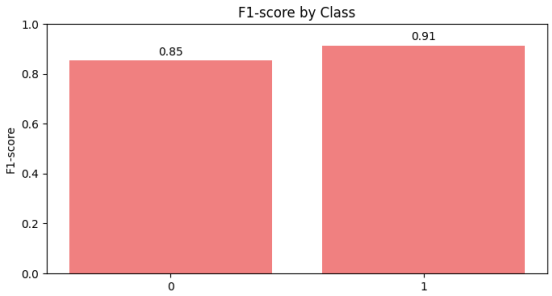


Figure 62: F1-score by class for SVM using Random Search.

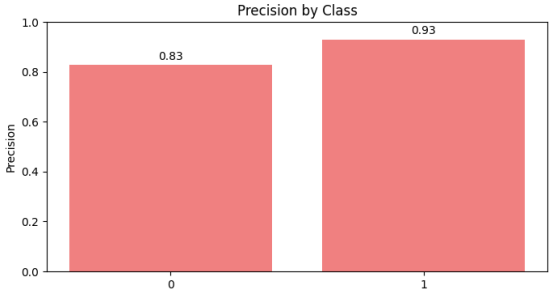


Figure 63: Precision by class for SVM using Random Search.

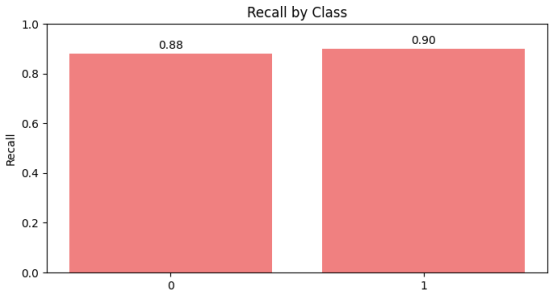


Figure 64: Recall by class for SVM using Random Search.

– KNN (Scenario_8_S):

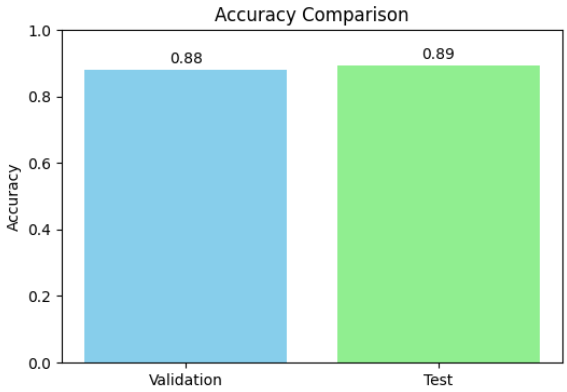


Figure 65: Accuracy comparison for KNN using Random Search.

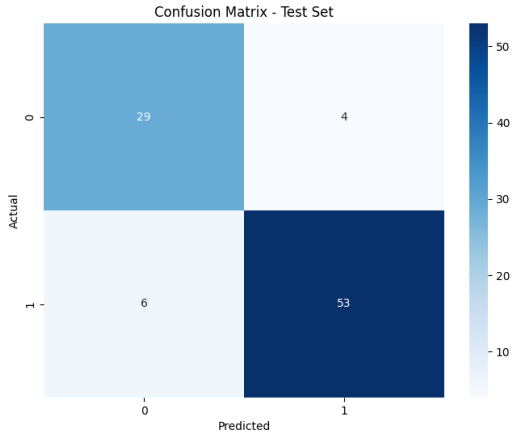


Figure 66: Confusion matrix for KNN using Random Search.

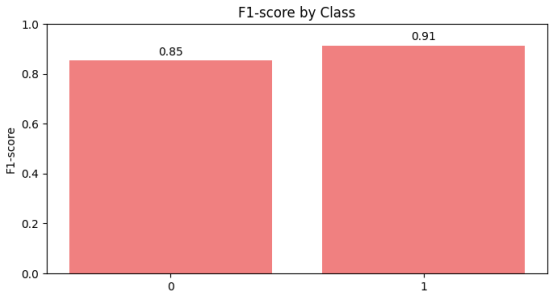


Figure 67: F1-score by class for KNN using Random Search.

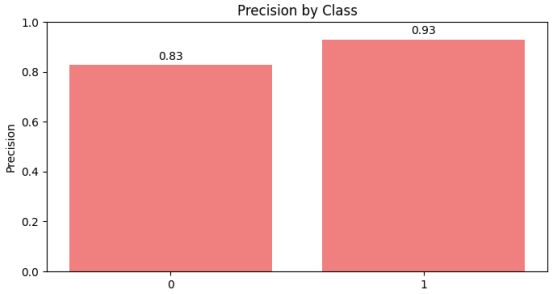


Figure 68: Precision by class for KNN using Random Search.

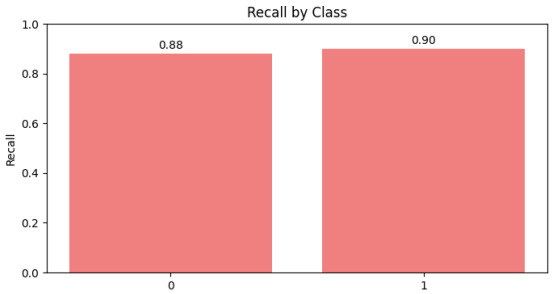


Figure 69: Recall by class for KNN using Random Search.

– Decision Tree (Scenario_8_S):

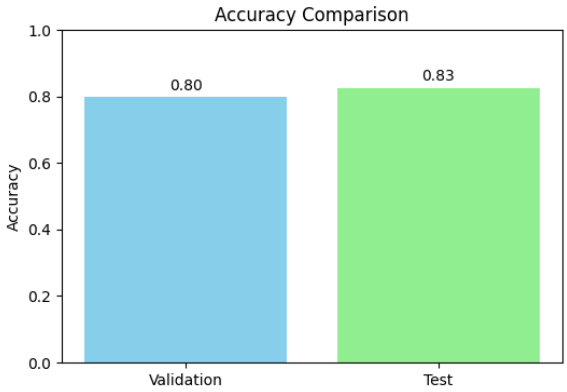


Figure 70: Accuracy comparison for Decision Tree using Random Search.

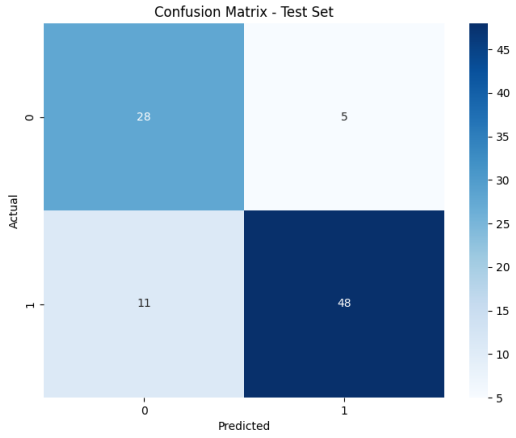


Figure 71: Confusion matrix for Decision Tree using Random Search.

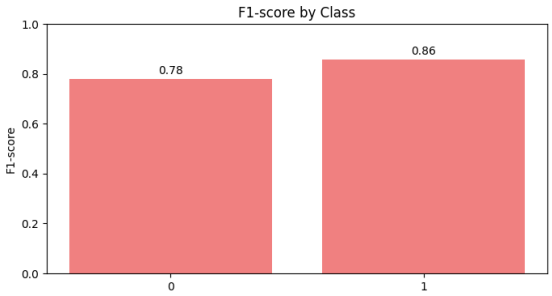


Figure 72: F1-score by class for Decision Tree using Random Search.

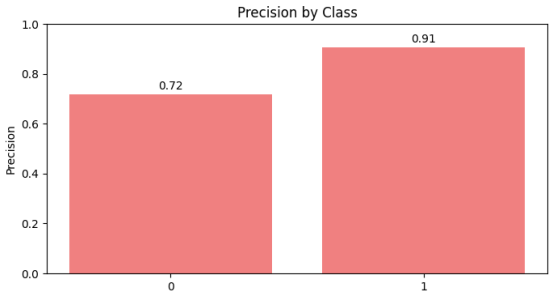


Figure 73: Precision by class for Decision Tree using Random Search.

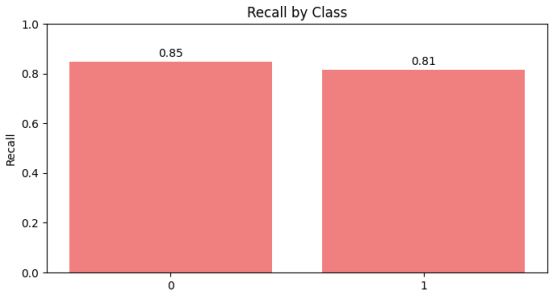


Figure 74: Recall by class for Decision Tree using Random Search.

5.4.3 Optuna

– Naive Bayes (Scenario_1_N):

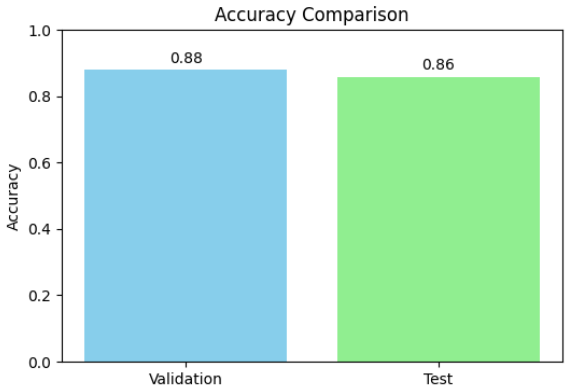


Figure 75: Accuracy comparison for Naive Bayes using Optuna.

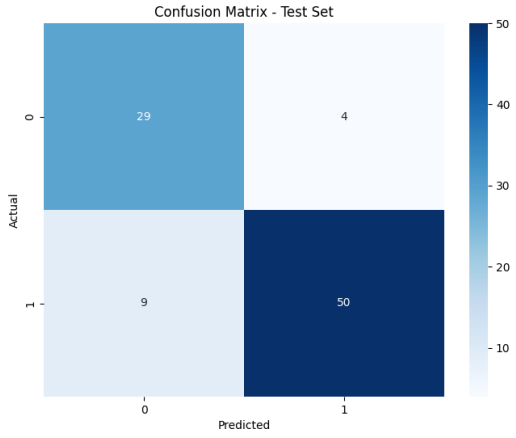


Figure 76: Confusion matrix for Naive Bayes using Optuna.

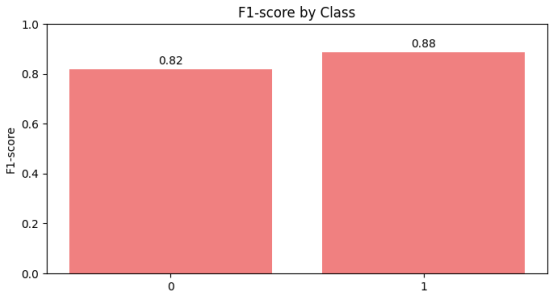


Figure 77: F1-score by class for Naive Bayes using Optuna.

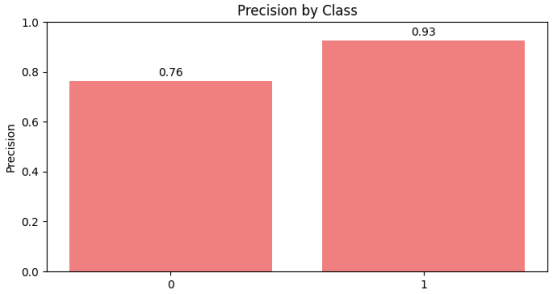


Figure 78: Precision by class for Naive Bayes using Optuna.

Figure 79: Recall by class for Naive Bayes using Optuna.

– SVM (Scenario_1_N):

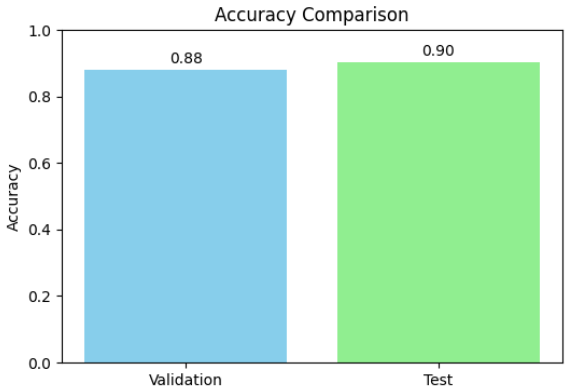


Figure 80: Accuracy comparison for SVM using Optuna.

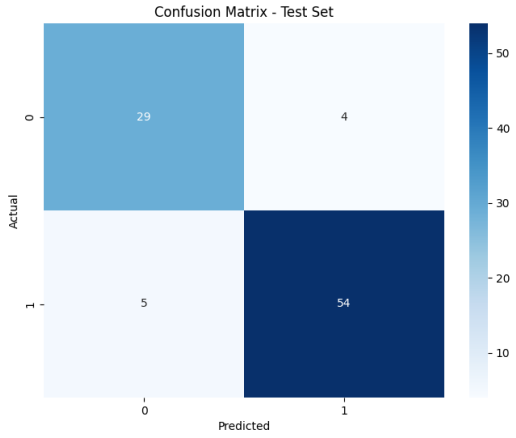


Figure 81: Confusion matrix for SVM using Optuna.

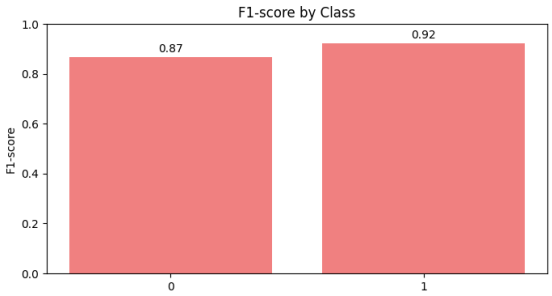


Figure 82: F1-score by class for SVM using Optuna.

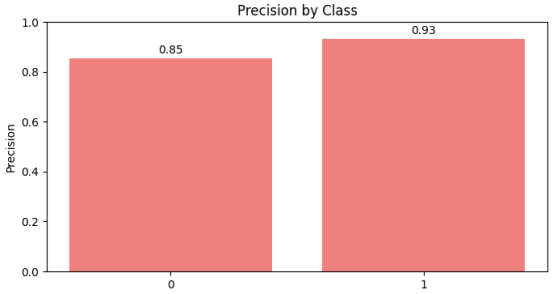


Figure 83: Precision by class for SVM using Optuna.

Figure 84: Recall by class for SVM using Optuna.

– KNN (Scenario_5_S):

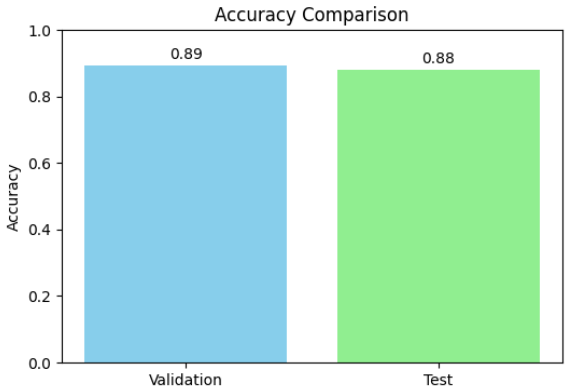


Figure 85: Accuracy comparison for KNN in Scenario 5 (Optimized)

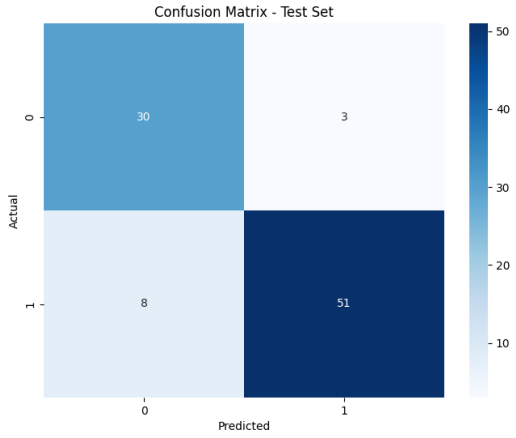


Figure 86: Confusion matrix for KNN in Scenario 5 (Optimized)

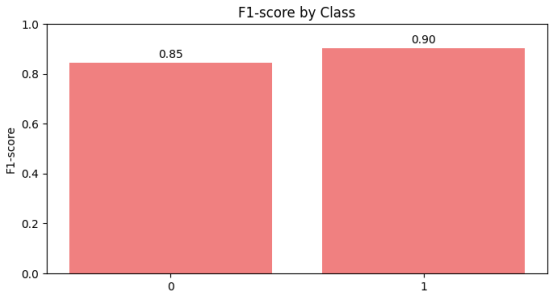


Figure 87: F1-score by class for KNN in Scenario 5 (Optimized)

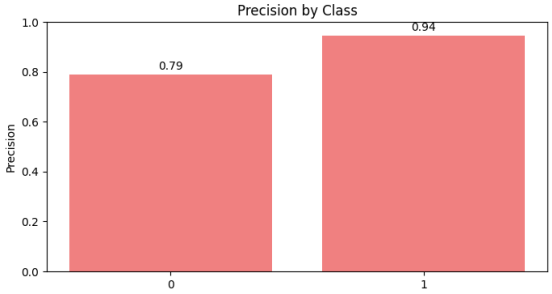


Figure 88: Precision by class for KNN in Scenario 5 (Optimized)

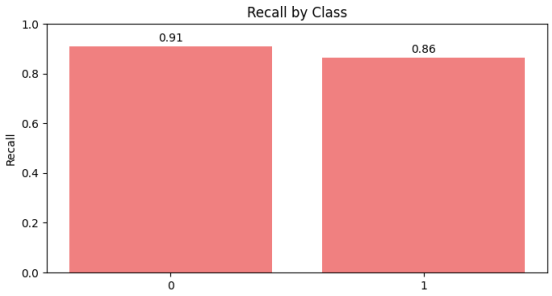


Figure 89: Recall by class for KNN in Scenario 5 (Optimized)

– Decision Tree (Scenario_1_N):

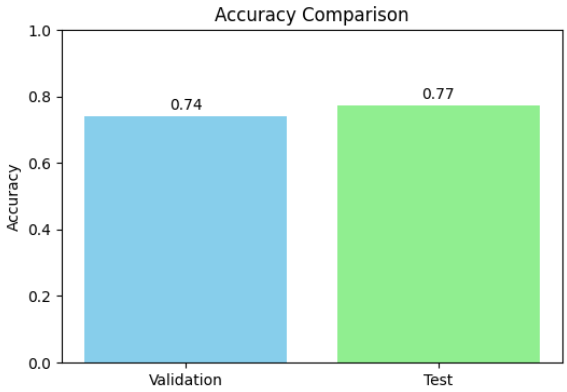


Figure 90: Accuracy comparison for Decision Tree in Scenario 1 (Optimized)

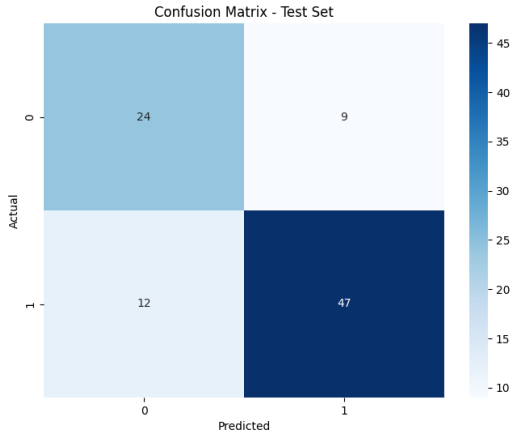


Figure 91: Confusion matrix for Decision Tree in Scenario 1 (Optimized)

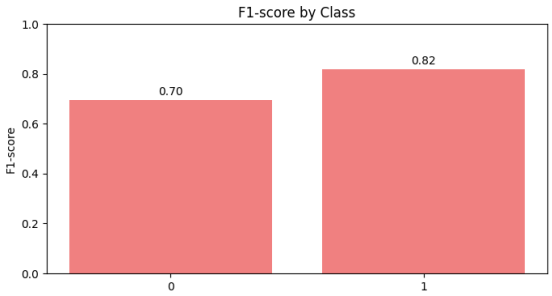


Figure 92: F1-score by class for Decision Tree in Scenario 1 (Optimized)

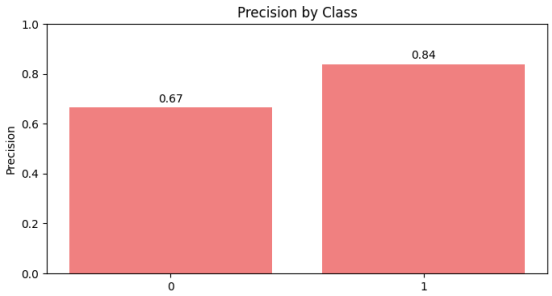


Figure 93: Precision by class for Decision Tree in Scenario 1 (Optimized)

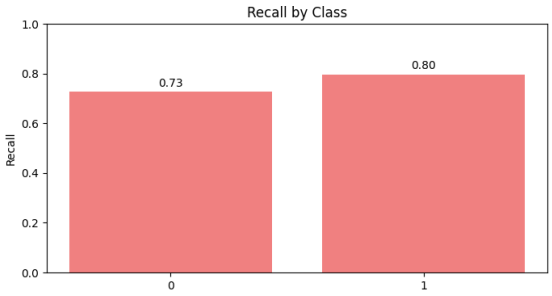


Figure 94: Recall by class for Decision Tree in Scenario 1 (Optimized)

5.5 Comparative Analysis of Models

The following section provides a detailed comparative analysis of the models used in the study, highlighting their strengths, limitations, and specific performance metrics.

5.5.1 Naive Bayes

Description: A probabilistic model based on Bayes' theorem, assuming conditional independence among features given the target class.

Advantages:

- Computationally efficient and fast to train.
- Effective with small datasets or when features are Gaussian-distributed.

Limitations:

- Performance degrades with correlated features.
- Not effective for complex decision boundaries.

Performance Highlights:

- Reliable performance with minimal preprocessing.
- Insensitive to scaling, making it robust for raw data.
- Bayesian Optimization (Optuna) improved performance by fine-tuning `var_smoothing`.

5.5.2 Support Vector Machines (SVM)

Description: Constructs hyperplanes in high-dimensional spaces for classification and supports both linear and non-linear kernels.

Advantages:

- Effective for high-dimensional data.
- Handles non-linear relationships efficiently with kernel tricks.

Limitations:

- Computationally expensive, especially with large datasets.
- Requires careful tuning of `C` and `gamma` hyperparameters.

Performance Highlights:

- High accuracy with standardized preprocessing.
- Robust performance after Grid Search and Optuna tuning.
- Sensitive to poorly scaled data due to reliance on distances.

5.5.3 K-Nearest Neighbors (KNN)

Description: A non-parametric, distance-based algorithm that classifies data based on the majority class of its `k` nearest neighbors.

Advantages:

- Simple to implement and understand.
- No explicit training phase; relies on inference-time distance computation.

Limitations:

- Sensitive to the choice of `k` and feature scaling.
- Computationally intensive for large datasets.

Performance Highlights:

- Best results with MinMax Scaling.
- Effective in scenarios with local data patterns.
- Tuning `n_neighbors` and distance metrics improved accuracy.

5.5.4 Decision Trees

Description: A tree-structured model that splits data into subsets based on feature values to form hierarchical decisions.

Advantages:

- Intuitive and interpretable.
- Handles both numerical and categorical data with minimal preprocessing.

Limitations:

- Prone to overfitting with deep trees.
- Sensitive to slight changes in data.

Performance Highlights:

- Competitive accuracy with minimal preprocessing.
- Robust performance across error-handling scenarios.
- Reduced overfitting with optimized `max_depth` and `min_samples_split`.

Summary of Model Performance

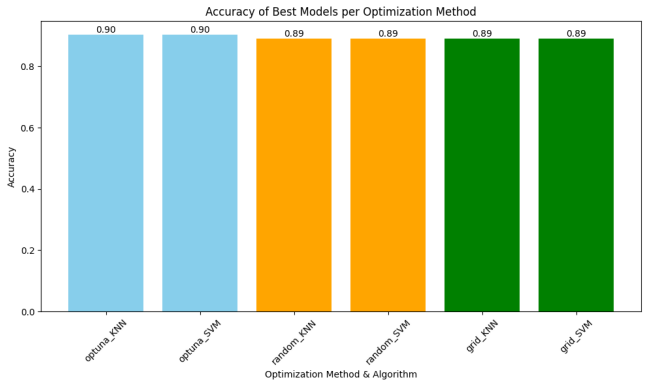


Figure 95: Summary of Best Models Per Method

Model	Strengths	Weaknesses	Best Use Cases
Naive Bayes	Fast; robust to scaling	Assumes feature independence	Quick prototyping, baselines
SVM	Effective for high-dimensional data	Sensitive to preprocessing	Well-separated classes, non-linear relations
KNN	Simple, non-parametric	Sensitive to k, scaling	Local patterns, small datasets
Decision Trees	Interpretable; mixed data types	Overfitting prone	Explainable models, missing data

Table 5: Summary of Model Performance

Insights:

- **Preprocessing Matters:** SVM and KNN showed sensitivity to scaling and data cleaning.
- **Tuning Techniques:** Bayesian Optimization consistently yielded the best results across models.
- **Robust Models:** Naive Bayes and Decision Trees performed reliably regardless of preprocessing variations.

6 Code Implementation

6.1 Imports and Necessary Libraries

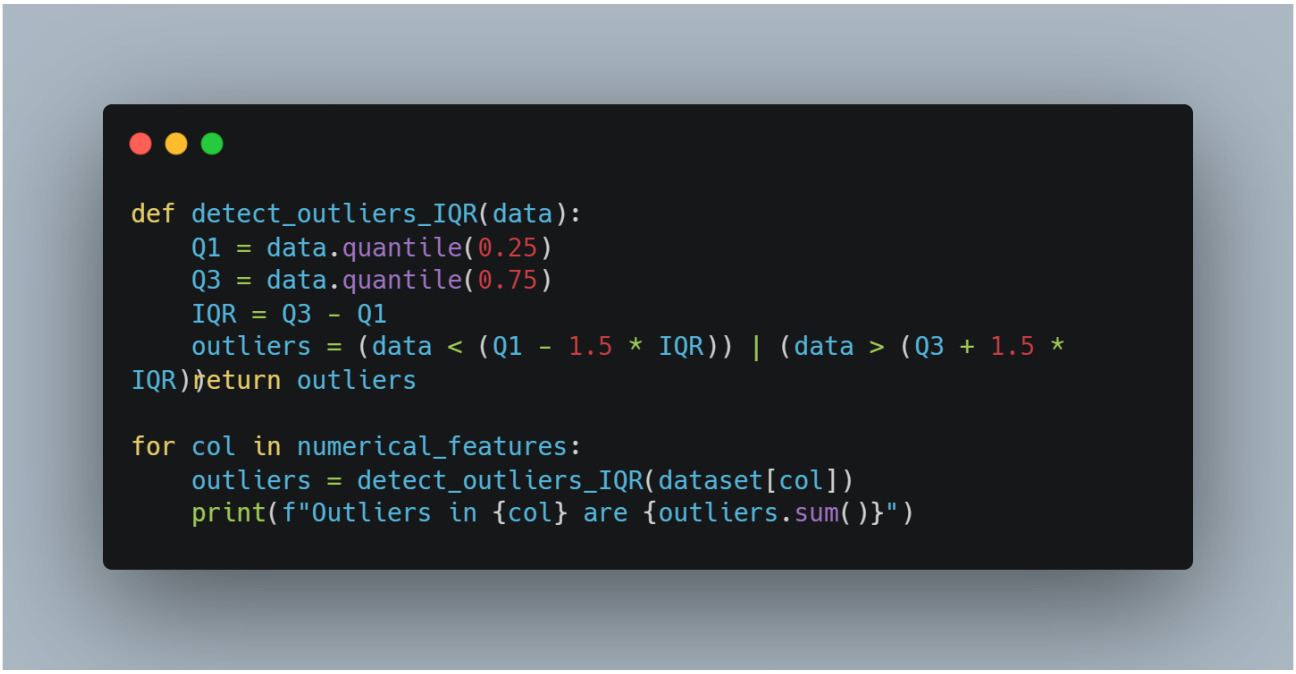
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import optuna
import pickle as pkl
import json
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV,
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    accuracy_score,
    classification_report
)
```

Figure 96: Necassary Library Imports

6.2 Data Exploration

```python
dataset = pd.read_csv("data/heart.csv")
print("Number of missing values in data", dataset.isnull().sum().sum()) # No Missing
print("Number of duplicates in data", dataset.duplicated().sum()) # No Duplicates
print(dataset["ChestPainType"].value_counts())
print("---")
print(dataset["RestingBP"].describe()) # 0 does it make sense????
print("---")
print(dataset["Age"].describe())
print("---")
print(dataset['Cholesterol'].describe())  # Can Cholesterol be 0???? actually No
print("---")
print(dataset['FastingBS'].value_counts())
print("---")
print(dataset['MaxHR'].describe())
print("---")
print(dataset['ExerciseAngina'].value_counts())
print("---")
print(dataset['Oldpeak'].describe()) # Negative values should be refused
print("---")
print(dataset["HeartDisease"].value_counts())
print("---")
numerical_features = dataset.select_dtypes(include=['int64', 'float64']).columns
numerical_features = numerical_features.drop(labels=['HeartDisease', "FastingBS"])
```

Figure 97: Exploring Nature of Dataset



```python
def detect_outliers_IQR(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    outliers = (data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 *
IQR))return outliers

for col in numerical_features:
    outliers = detect_outliers_IQR(dataset[col])
    print(f"Outliers in {col} are {outliers.sum()}")
```

Figure 98: Outlier Detection Using IQR

6.3 Data Visualization

```python
def kde_plot(df, feature):
    plt.figure(figsize=(10, 6))
    plt.title(f"{feature} Distribution")
    sns.kdeplot(df[feature], color="b", fill=True)
    plt.show()

def pie_plot(df, feature):
    plt.figure(figsize=(10, 6))
    plt.title(f"{feature} Distribution")
    df[feature].value_counts().plot.pie(autopct="%1.1f%%")
    plt.show()

def bar_plot(df, feature):
    plt.figure(figsize=(10, 6))
    plt.title(f"{feature} Distribution")
    df[feature].value_counts().plot.bar()
    plt.show()

def box_plot(df, feature):
    plt.figure(figsize=(10, 6))
    plt.title(f"{feature} Distribution")
    sns.boxplot(df[feature])
    plt.show()

def scatter_plot(df, feature1, feature2, hue):
    plt.figure(figsize=(10, 6))
    plt.title(f"{feature1} vs {feature2}")
    sns.scatterplot(x=feature1, y=feature2, data=df,
hue=hue)show()
```

Figure 99: Data Visualization Functions

```
kde_plot(df, "Age")
kde_plot(df, "RestingBP")
kde_plot(df, "Cholesterol")
kde_plot(df, "MaxHR")
kde_plot(df, "Oldpeak")
pie_plot(df, "ChestPainType")
pie_plot(df, "HeartDisease")
pie_plot(df, "FastingBS")
bar_plot(df, "Sex")
bar_plot(df, "RestingECG")
bar_plot(df, "ExerciseAngina")
bar_plot(df, "ST_Slope")
bar_plot(df, "HeartDisease")
box_plot(df, "Age")
box_plot(df, "RestingBP")
box_plot(df, "Cholesterol")
box_plot(df, "MaxHR")
box_plot(df, "Oldpeak")
scatter_plot(df, "Age", "MaxHR", "HeartDisease")
scatter_plot(df, "Age", "Cholesterol", "HeartDisease")
scatter_plot(df, "Age", "RestingBP", "HeartDisease")
scatter_plot(df, "Age", "Oldpeak", "HeartDisease")
scatter_plot(df, "Age", "HeartDisease", "HeartDisease")
scatter_plot(df, "MaxHR", "Cholesterol", "HeartDisease")
scatter_plot(df, "MaxHR", "RestingBP", "HeartDisease")
scatter_plot(df, "MaxHR", "Oldpeak", "HeartDisease")
scatter_plot(df, "MaxHR", "HeartDisease", "HeartDisease")
scatter_plot(df, "Cholesterol", "RestingBP", "HeartDisease")
scatter_plot(df, "Cholesterol", "Oldpeak", "HeartDisease")
scatter_plot(df, "Cholesterol", "HeartDisease",
"HeartDisease")
scatter_plot(df, "RestingBP", "Oldpeak", "HeartDisease")
scatter_plot(df, "RestingBP", "HeartDisease", "HeartDisease")
scatter_plot(df, "Oldpeak", "HeartDisease", "HeartDisease")
```

Figure 100: Calling The Visualization Functions

6.4 Data Preprocessing

```
data_dir = "Scenarios"
os.makedirs(data_dir, exist_ok=True)
df = pd.read_csv('data/heart.csv')
category_columns = df.select_dtypes(include=['object']).columns
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns
numerical_features = numerical_features.drop(labels=['HeartDisease',
"FastingBS"])
```

Figure 101: Preprocessing Utils

```
def label_encode(df, columns):
    le = LabelEncoder()
    for column in columns:
        df[column] = le.fit_transform(df[column])
    return df

def apply_scaler(df, scaler):
    columns = df.columns
    columns = columns.drop('HeartDisease')
    df_new = pd.DataFrame(scaler.fit_transform(df.drop(columns='HeartDisease')),
columns=new_columns)
    df_new = pd.concat([df_new, df[['HeartDisease']]], axis=1)
    return df_new

def detect_outliers_IQR(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    outliers = (data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))
    return outliers
```

Figure 102: Preprocessing Utils Part 2

```python
def is_blunder(df):
    columns = []
    if df["Cholesterol"].min() <= 0:
        print("Blunder in Cholesterol")
        columns.append("Cholesterol")
    if df["Oldpeak"].min() < 0:
        print("Blunder in Oldpeak")
        columns.append("Oldpeak")
    if df["RestingBP"].min() <= 0:
        print("Blunder in RestingBP")
        columns.append("RestingBP")

    return columns

def fix_blunder(df, columns):
    for column in columns:
        if column == "Cholesterol" or column == "RestingBP":
            df[column] = df[column].replace(0, df[column].mean())
        elif column == "Oldpeak":
            df[column] = df[column].apply(lambda x: np.abs(x) if x < 0 else
x)  # Validate if blunder is fixed
    columns = is_blunder(df)
    if len(columns) == 0:
        print("Blunders fixed")
    return df
```

Figure 103: Handling blunder functions

6.5 Data Preprocessing Scenarios

```python
df1 = df.copy()

df1 = label_encode(df1, category_columns)
df1 = apply_scaler(df1, MinMaxScaler())

df1.to_csv(os.path.join(data_dir, "dataset_scenario_1_N.csv"),
index=False)
```

Figure 104: Scenario-1N Data Generation

```python
df2 = df.copy()

df2 = label_encode(df2, category_columns)
df2 = apply_scaler(df2, StandardScaler())

df2.to_csv(os.path.join(data_dir, "dataset_scenario_1_S.csv"),
index=False)
```

Figure 105: Scenario-1S Data Generation

```python
df3 = df.copy()

blunder_columns = is_blunder(df3)
df3 = fix_blunder(df3, blunder_columns)

df3 = label_encode(df3, category_columns)
df3 = apply_scaler(df3, MinMaxScaler())

df3.to_csv(os.path.join(data_dir, "dataset_scenario_2_N.csv"),
index=False)
df3
```

Figure 106: Scenario-2N Data Generation

```python
df4 = df.copy()

blunder_columns = is_blunder(df4)
df4 = fix_blunder(df4, blunder_columns)

df4 = label_encode(df4, category_columns)
df4 = apply_scaler(df4, StandardScaler())
df4.to_csv(os.path.join(data_dir, "dataset_scenario_2_S.csv"),
index=False)
df4
```

Figure 107: Scenario-2S Data Generation

```python
df5 = df.copy()

outliers = {}

for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df5[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

outlier_mask = pd.Series(True, index=df5.index)
for col in numerical_features:
    outlier_mask &= ~outliers[col].reindex(df5.index, fill_value=False)

print(f"Total outliers removed: {outlier_mask.sum()}")

df5 = df5[outlier_mask]

df5.reset_index(drop=True, inplace=True)

blunder_columns = is_blunder(df5)

df5 = fix_blunder(df5, blunder_columns) # if no blunders then it will return the same
dataframe
df5 = label_encode(df5, category_columns)
df5 = apply_scaler(df5, MinMaxScaler())
df5.to_csv(os.path.join(data_dir, "dataset_scenario_3_N.csv"), index=False)

df5
```

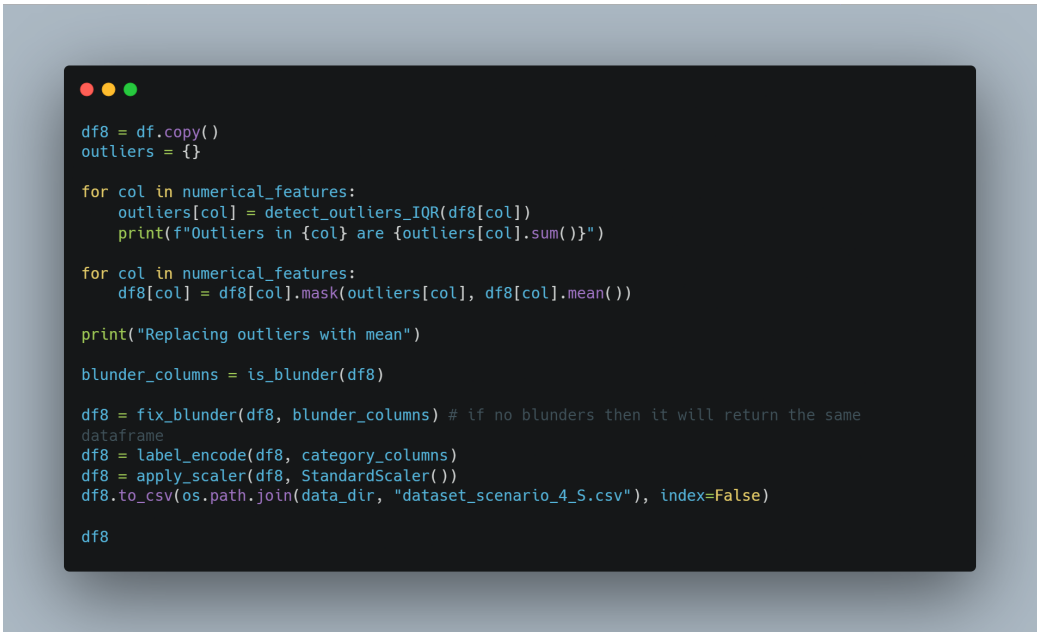
Figure 108: Scenario-3N Data Generation

```python
df6 = df.copy()

outliers = {}

for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df6[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

outlier_mask = pd.Series(True, index=df6.index)
for col in numerical_features:
    outlier_mask &= ~outliers[col].reindex(df6.index, fill_value=False)

print(f"Total outliers removed: {outlier_mask.sum()}")

df6 = df6[outlier_mask]

df6.reset_index(drop=True, inplace=True)

blunder_columns = is_blunder(df6)

df6 = fix_blunder(df6, blunder_columns) # if no blunders then it will return the same
dataframe
df6 = label_encode(df6, category_columns)
df6 = apply_scaler(df6, StandardScaler())
df6.to_csv(os.path.join(data_dir, "dataset_scenario_3_S.csv"), index=False)

df6
```

Figure 109: Scenario-3S Data Generation

```python
df7 = df.copy()
outliers = {}

for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df7[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

for col in numerical_features:
    df7[col] = df7[col].mask(outliers[col], df7[col].mean())

print("Replacing outliers with mean")

blunder_columns = is_blunder(df7)

df7 = fix_blunder(df7, blunder_columns) # if no blunders then it will return the same
dataframe
df7 = label_encode(df7, category_columns)
df7 = apply_scaler(df7, MinMaxScaler())
df7.to_csv(os.path.join(data_dir, "dataset_scenario_4_N.csv"), index=False)

df7
```

Figure 110: Scenario-4N Data Generation



```
df8 = df.copy()
outliers = {}

for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df8[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

for col in numerical_features:
    df8[col] = df8[col].mask(outliers[col], df8[col].mean())

print("Replacing outliers with mean")

blunder_columns = is_blunder(df8)

df8 = fix_blunder(df8, blunder_columns) # if no blunders then it will return the same
dataframe
df8 = label_encode(df8, category_columns)
df8 = apply_scaler(df8, StandardScaler())
df8.to_csv(os.path.join(data_dir, "dataset_scenario_4_S.csv"), index=False)

df8
```

Figure 111: Scenario-4S Data Generation

```
df9 = df.copy()

df9 = label_encode(df9, category_columns.drop(labels='ChestPainType'))
df9 = pd.get_dummies(df9, columns=['ChestPainType'], dtype=int)

df9 = apply_scaler(df9, MinMaxScaler())
df9.to_csv(os.path.join(data_dir, "dataset_scenario_5_N.csv"),
index=False)
df9
```

Figure 112: Scenario-5N Data Generation

```
df10 = df.copy()

df10 = label_encode(df10, category_columns.drop(labels='ChestPainType'))
df10 = pd.get_dummies(df10, columns=['ChestPainType'], dtype=int)

df10 = apply_scaler(df10, StandardScaler())
df10.to_csv(os.path.join(data_dir, "dataset_scenario_5_S.csv"),
index=False)
df10
```

Figure 113: Scenario-5S Data Generation

```
df11 = df.copy()

df11 = label_encode(df11, category_columns.drop(labels='ChestPainType'))
df11 = pd.get_dummies(df11, columns=['ChestPainType'], dtype=int)

blunder_columns = is_blunder(df11)
df11 = fix_blunder(df11, blunder_columns)

df11 = apply_scaler(df11, MinMaxScaler())
df11.to_csv(os.path.join(data_dir, "dataset_scenario_6_N.csv"),
index=False)
df11
```

Figure 114: Scenario-6N Data Generation

```python
df12 = df.copy()

df12 = label_encode(df12, category_columns.drop(labels='ChestPainType'))
df12 = pd.get_dummies(df12, columns=['ChestPainType'], dtype=int)

blunder_columns = is_blunder(df12)
df12 = fix_blunder(df12, blunder_columns)

df12 = apply_scaler(df12, StandardScaler())
df12.to_csv(os.path.join(data_dir, "dataset_scenario_6_S.csv"),
index=False)
df12
```

Figure 115: Scenario-6S Data Generation

```python
df13 = df.copy()

outliers = {}
for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df13[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

outlier_mask = pd.Series(True, index=df13.index)
for col in numerical_features:
    outlier_mask &= ~outliers[col].reindex(df13.index, fill_value=False)

df13 = df13[outlier_mask]
df13.reset_index(drop=True, inplace=True)

blunder_columns = is_blunder(df13)
df13 = fix_blunder(df13, blunder_columns)

df13 = label_encode(df13, category_columns.drop(labels='ChestPainType'))
df13 = pd.get_dummies(df13, columns=['ChestPainType'], dtype=int)

df13 = apply_scaler(df13, MinMaxScaler())
df13.to_csv(os.path.join(data_dir, "dataset_scenario_7_N.csv"),
index=False)
df13
```

Figure 116: Scenario-7N Data Generation

```python
df14 = df.copy()

outliers = {}
for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df14[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

outlier_mask = pd.Series(True, index=df14.index)
for col in numerical_features:
    outlier_mask &= ~outliers[col].reindex(df14.index, fill_value=False)

print("Removed Outliers:", df14.shape[0] - sum(outlier_mask))

df14 = df14[outlier_mask]
df14.reset_index(drop=True, inplace=True)

blunder_columns = is_blunder(df14)
df14 = fix_blunder(df14, blunder_columns)

df14 = label_encode(df14, category_columns.drop(labels='ChestPainType'))
df14 = pd.get_dummies(df14, columns=['ChestPainType'], dtype=int)

df14 = apply_scaler(df14, StandardScaler())
df14.to_csv(os.path.join(data_dir, "dataset_scenario_7_S.csv"),
index=False)
df14
```

Figure 117: Scenario-7S Data Generation

```
df15 = df.copy()

outliers = {}
for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df15[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

for col in numerical_features:
    df15[col] = df15[col].mask(outliers[col], df15[col].mean())

blunder_columns = is_blunder(df15)
df15 = fix_blunder(df15, blunder_columns)

df15 = label_encode(df15, category_columns.drop(labels='ChestPainType'))
df15 = pd.get_dummies(df15, columns=['ChestPainType'], dtype=int)

df15 = apply_scaler(df15, MinMaxScaler())
df15.to_csv(os.path.join(data_dir, "dataset_scenario_8_N.csv"),
index=False)
df15
```

Figure 118: Scenario-8N Data Generation

```
df16 = df.copy()

outliers = {}
for col in numerical_features:
    outliers[col] = detect_outliers_IQR(df16[col])
    print(f"Outliers in {col} are {outliers[col].sum()}")

for col in numerical_features:
    df16[col] = df16[col].mask(outliers[col], df16[col].mean())

print("Replace Outliers with mean:", df16.shape[0] - sum(outlier_mask))

blunder_columns = is_blunder(df16)
df16 = fix_blunder(df16, blunder_columns)

df16 = label_encode(df16, category_columns.drop(labels='ChestPainType'))
df16 = pd.get_dummies(df16, columns=['ChestPainType'], dtype=int)

df16 = apply_scaler(df16, StandardScaler())
df16.to_csv(os.path.join(data_dir, "dataset_scenario_8_S.csv"),
index=False)
df16
```

Figure 119: Scenario-8S Data Generation

6.6 Features PCA

```python
def plot_pca_scenarios(scenarios, three_d=False):
    for scenario in scenarios:
        df = pd.read_csv(os.path.join(data_dir, scenario))
        X = df.drop(columns='HeartDisease')
        y = df['HeartDisease']
        cmap = sns.color_palette("viridis", as_cmap=True)
        norm = plt.Normalize(vmin=np.min(y), vmax=np.max(y))
        unique_values = np.unique(y)
        colors = [cmap(norm(value)) for value in unique_values]
        pca = PCA(n_components=2)
        X_pca = pca.fit_transform(X)
        fig, axes = plt.subplots(1, 2, figsize=(15, 6))
        sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y, palette=colors,
ax=axes[0])
        axes[0].set_title(f"PCA 2D for {scenario}")

        if three_d:
            ax_3d = fig.add_subplot(122, projection='3d')
            ax_3d.scatter(X_pca[:, 0], X_pca[:, 1], y, c=y, cmap=cmap)
            ax_3d.set_title(f"PCA 3D for {scenario}")
        else:
            fig.delaxes(axes[1])

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

Figure 120: PCA Plotter functions

```python
list_of_scenarios = os.listdir(data_dir)
plot_pca_scenarios(list_of_scenarios,
three_d=True)
```

Figure 121: PCA Plotter functions calling

6.7 Modeling

```
scenario_datasets = os.listdir("./Scenarios")
scenario_datasets = [f"./Scenarios/{x}" for x in scenario_datasets]
scenarios_description = [
    {"name": "Scenario 1_N", "encoding": "Label encoding", "remove_outliers": False, "remove_errors":
False, "scaling": "MinMax"},
    {"name": "Scenario 1_S", "encoding": "Label encoding", "remove_outliers": False, "remove_errors":
False, "scaling": "Standard"},
    {"name": "Scenario 2_N", "encoding": "Label encoding", "remove_outliers": False, "remove_errors":
False,
     "impute_zeros": {"RestingBP": "mean", "Cholesterol": "mean"}, "oldpeak_abs": True, "scaling":
"MinMax"},
    {"name": "Scenario 2_S", "encoding": "Label encoding", "remove_outliers": False, "remove_errors":
False,
     "impute_zeros": {"RestingBP": "mean", "Cholesterol": "mean"}, "oldpeak_abs": True, "scaling":
"Standard"},
    {"name": "Scenario 3_N", "encoding": "Label encoding", "remove_outliers": True, "remove_errors":
True, "scaling": "None"},
    {"name": "Scenario 3_S", "encoding": "Label encoding", "remove_outliers": True, "remove_errors":
True, "scaling": "Standard"},
    {"name": "Scenario 4_N", "encoding": "Label encoding", "remove_outliers": "replace_with_mean",
"remove_errors": True, "scaling": "MinMax"},
    {"name": "Scenario 4_S", "encoding": "Label encoding", "remove_outliers": "replace_with_mean",
"remove_errors": True, "scaling": "Standard"},
    {"name": "Scenario 5_N", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": False, "remove_errors": False, "scaling": "MinMax"},
    {"name": "Scenario 5_S", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": False, "remove_errors": False, "scaling": "Standard"},
    {"name": "Scenario 6_N", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": False, "remove_errors": True, "scaling": "None"},
    {"name": "Scenario 6_S", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": False, "remove_errors": True, "scaling": "Standard"},
    {"name": "Scenario 7_N", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": True, "remove_errors": True, "scaling": "MinMax"},
    {"name": "Scenario 7_S", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": True, "remove_errors": True, "scaling": "Standard"},
    {"name": "Scenario 8_N", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": "replace_with_mean", "remove_errors": True, "scaling": "None"},
    {"name": "Scenario 8_S", "encoding": {"Sex": "Label", "ExerciseAngina": "Label", "ST_Slope":
"Label", "RestingECG": "Label", "ChestPainType": "OneHot"},
     "remove_outliers": "replace_with_mean", "remove_errors": True, "scaling": "Standard"},
]
```

Figure 122: Scenarios Modeling

```python
def get_search_hyperparameters(model_type):
    if model_type == "NaiveBayes":
        return {
            "var_smoothing": [1e-9, 1e-8, 1e-7, 1e-6, 1e-5],
            "priors": [None],
        }
    elif model_type == "SVM":
        return {
            "C": [0.1, 1, 10, 100, 1000],
            "gamma": [1, 0.1, 0.01, 0.001, 0.0001],
            "kernel": ["rbf", "linear", "poly", "sigmoid"],
        }
    elif model_type == "KNN":
        return {
            "n_neighbors": [3, 5, 7, 9, 11],  # Number of neighbors
            "weights": ["uniform", "distance"],  # Weighting scheme
            "metric": ["euclidean", "manhattan", "minkowski"],  # Distance metrics
        }
    elif model_type == "DecisionTree":
        return {
            "criterion": ["gini", "entropy", "log_loss"],  # Criterion for split
            "max_depth": [None, 10, 20, 30, 40, 50],  # Maximum depth of the tree
            "min_samples_split": [2, 5, 10],  # Minimum number of samples to split a
node        "min_samples_leaf": [1, 2, 4],  # Minimum number of samples per leaf node
        }
    else:
        return {}
```

Figure 123: Search Space Params function

6.8 Modeling Util Classes

```python
class optuna_tuner:
    def __init__(self, model, X_train, y_train, direction="maximize", n_trials=100, n_jobs=-1, scoring="accuracy", cv=5):
        self.model = model
        self.X_train = X_train
        self.y_train = y_train
        self.direction = direction
        self.n_trials = n_trials
        self.n_jobs = n_jobs
        self.scoring = scoring
        self.cv = cv

    def objective(self, trial):
        # Handle Naive Bayes
        if isinstance(self.model, GaussianNB):
            model = GaussianNB(
                var_smoothing=trial.suggest_float("var_smoothing", 1e-9, 1e-5)
            )

        # Handle SVM
        elif isinstance(self.model, SVC):
            model = SVC(
                C=trial.suggest_float("C", 1e-3, 10),
                gamma=trial.suggest_float("gamma", 1e-3, 10),
                kernel=trial.suggest_categorical("kernel", ["rbf", "linear", "poly", "sigmoid"]),
            )

        # Handle KNN
        elif isinstance(self.model, KNeighborsClassifier):
            model = KNeighborsClassifier(
                n_neighbors=trial.suggest_int("n_neighbors", 3, 15),
                weights=trial.suggest_categorical("weights", ["uniform", "distance"]),
                metric=trial.suggest_categorical("metric", ["euclidean", "manhattan", "minkowski"]),
            )

        # Handle Decision Tree
        elif isinstance(self.model, DecisionTreeClassifier):
            model = DecisionTreeClassifier(
                criterion=trial.suggest_categorical("criterion", ["gini", "entropy", "log_loss"]),
                max_depth=trial.suggest_int("max_depth", 5, 50),
                min_samples_split=trial.suggest_int("min_samples_split", 2, 10),
                min_samples_leaf=trial.suggest_int("min_samples_leaf", 1, 5),
            )

        else:
            raise ValueError("Unsupported model type for optimization.")

        # Perform cross-validation and return the mean score
        scores = cross_val_score(model, self.X_train, self.y_train, cv=self.cv, scoring=self.scoring, n_jobs=self.n_jobs)
        return scores.mean()

    def optimize_study(self):
        study = optuna.create_study(direction=self.direction)
        study.optimize(self.objective, n_trials=self.n_trials)
        return study.best_params
```

Figure 124: Class Optuna Tuner

```python
class ModelTrainer:
    def __init__(self, dataset, target_column, model):
        self.dataset = dataset
        self.target_column = target_column
        self.X = dataset.drop(target_column, axis=1)
        self.y = dataset[target_column]
        self.model = model
        self.results = {}

    def split_data(self, test_size=0.2, val_size=0.1, random_state=42):
        X_train, X_temp, y_train, y_temp = train_test_split(
            self.X, self.y, test_size=test_size + val_size, random_state=random_state
        )
        val_ratio = val_size / (test_size + val_size)
        self.X_val, self.X_test, self.y_val, self.y_test = train_test_split(
            X_temp, y_temp, test_size=val_ratio, random_state=random_state
        )
        self.X_train, self.y_train = X_train, y_train

    def train(self):
        self.model.fit(self.X_train, self.y_train)

    def tune_and_train(self,search_type , search_config, param_grid):

        clf = self.model
        if search_type == "grid":
            search = GridSearchCV(clf, param_grid, cv=search_config["cv"], scoring=search_config["scoring"],
n_jobs=search_config["n_jobs"])
        elif search_type == "random":
            search = RandomizedSearchCV(clf, param_grid, cv=search_config["cv"], scoring=search_config["scoring"],
n_iter=search_config["n_iter"], n_jobs=search_config["n_jobs"])
        elif search_type == "optuna":
            search = optuna_tuner(self.model, self.X_train, self.y_train, search_config["direction"], search_config["n_trials"],
search_config["n_jobs"], search_config["scoring"], search_config["cv"])

        best_params = None
        if isinstance(search, optuna_tuner):
            best_params = search.optimize_study()
            self.model = self.model.set_params(**best_params)
            self.model.fit(self.X_train, self.y_train)
        elif search_type == "grid" or search_type == "random":
            search.fit(self.X_train, self.y_train)
            best_params = search.best_params_
            self.model = search.best_estimator_

        print("Best hyperparameters: ", best_params)

        return best_params


    def test_and_evaluate(self, save_dir=None):
        # Predict on validation and test sets
        y_val_pred = self.model.predict(self.X_val)
        y_test_pred = self.model.predict(self.X_test)

        # Calculate metrics for validation and test sets
        metrics_val = {
            "accuracy": accuracy_score(self.y_val, y_val_pred),
            "precision": precision_score(self.y_val, y_val_pred, average="weighted"),
            "recall": recall_score(self.y_val, y_val_pred, average="weighted"),
            "f1_score": f1_score(self.y_val, y_val_pred, average="weighted"),
            "confusion_matrix": confusion_matrix(self.y_val, y_val_pred),
        }

        metrics_test = {
            "accuracy": accuracy_score(self.y_test, y_test_pred),
            "precision": precision_score(self.y_test, y_test_pred, average="weighted"),
            "recall": recall_score(self.y_test, y_test_pred, average="weighted"),
            "f1_score": f1_score(self.y_test, y_test_pred, average="weighted"),
            "confusion_matrix": confusion_matrix(self.y_test, y_test_pred),
        }

        # Store results
        self.results["validation"] = metrics_val
        self.results["test"] = metrics_test

        # Print validation metrics
        print("Validation Metrics:")
        print(f"  Accuracy: {metrics_val['accuracy']:.4f}")
        print(f"  Precision: {metrics_val['precision']:.4f}")
        print(f"  Recall: {metrics_val['recall']:.4f}")
        print(f"  F1 Score: {metrics_val['f1_score']:.4f}")
        print("\nTest Metrics:")
        print(f"  Accuracy: {metrics_test['accuracy']:.4f}")
        print(f"  Precision: {metrics_test['precision']:.4f}")
        print(f"  Recall: {metrics_test['recall']:.4f}")
        print(f"  F1 Score: {metrics_test['f1_score']:.4f}")

        # Confusion matrix visualization for test set
        print("\nConfusion Matrix (Test Set):")
        cm = metrics_test["confusion_matrix"]
        print(cm)
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=True, yticklabels=True)
        plt.title("Confusion Matrix - Test Set")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        if save_dir:
            os.makedirs(save_dir, exist_ok=True)
            plt.savefig(os.path.join(save_dir, "confusion_matrix.png"))
        plt.show()

        return metrics_test

    def plot_results(self, save_dir=None):
        # Create the directory if saving is required
        if save_dir:
            os.makedirs(save_dir, exist_ok=True)

        # Plot Accuracy Comparison
        if "validation" in self.results and "test" in self.results:
            accuracies = {
                "Validation": self.results["validation"]["accuracy"],
                "Test": self.results["test"]["accuracy"],
            }
            plt.figure(figsize=(6, 4))
            plt.bar(accuracies.keys(), accuracies.values(), color=["skyblue", "lightgreen"])
            plt.title("Accuracy Comparison")
            plt.ylabel("Accuracy")
            plt.ylim(0, 1)
            for i, v in enumerate(accuracies.values()):
                plt.text(i, v + 0.02, f"{v:.2f}", ha="center")
            if save_dir:
                plt.savefig(os.path.join(save_dir, "accuracy_comparison.png"))
            plt.show()

        # Confusion Matrix Heatmap for Test Set
        y_test_pred = self.model.predict(self.X_test)
        cm = confusion_matrix(self.y_test, y_test_pred)
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=True, yticklabels=True)
        plt.title("Confusion Matrix - Test Set")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        if save_dir:
            plt.savefig(os.path.join(save_dir, "confusion_matrix_heatmap.png"))
        plt.show()

        # Plot Class-wise Precision, Recall, and F1-Score
        if hasattr(self.model, "classes_"):  # Ensure the model supports classes_
            report = classification_report(self.y_test, y_test_pred, output_dict=True)
            metrics = ['precision', 'recall', 'f1-score']
            class_names = list(report.keys())[:-3]  # Exclude avg/total keys
            class_metrics = {metric: [report[class_name][metric] for class_name in class_names] for metric in metrics}

            for metric, values in class_metrics.items():
                plt.figure(figsize=(6, 4))
                plt.bar(class_names, values, color="lightcoral")
                plt.title(f"{metric.capitalize()} by Class")
                plt.ylabel(metric.capitalize())
                plt.ylim(0, 1)
                for i, v in enumerate(values):
                    plt.text(i, v + 0.02, f"{v:.2f}", ha="center")
                if save_dir:
                    plt.savefig(os.path.join(save_dir, f"{metric}_by_class.png"))
                plt.show()
```

Figure 125: Class Model Tuner

6.9 Running HPO Scenarios

```
search_config_grid = {
    "cv": 5,
    "scoring": "accuracy",
    "n_jobs": -1,
}
search_config_random = {
    "cv": 5,
    "scoring": "accuracy",
    "n_iter": 50,
    "n_jobs": -1,
}
search_config_optuna = {
    "direction": "maximize",
    "n_trials": 100,
    "n_jobs": -1,
    "scoring": "accuracy",
    "cv": 5,
}

target_column = "HeartDisease"
```

Figure 126: HPO techniques configurations

```python
def run_scenario(scenario, model_type, search_type, search_config, param_grid):
    # Load the dataset
    dataset = pd.read_csv(scenario)
    model = None

    # Initialize the model based on the model_type
    if model_type == "NaiveBayes":
        model = GaussianNB()
    elif model_type == "SVM":
        model = SVC()
    elif model_type == "KNN":
        model = KNeighborsClassifier()
    elif model_type == "DecisionTree":
        model = DecisionTreeClassifier()
    else:
        raise ValueError(f"Model type {model_type} is not supported.")

    # Create a directory for saving plots and results
    scenario_name = os.path.splitext(os.path.basename(scenario))[0]  # Extract scenario name from file
    save_dir = os.path.join("SCORES", model_type, search_type, scenario_name)  # SCORES as top-level
directory
    os.makedirs(save_dir, exist_ok=True)

    print(f"Running {model_type} with {search_type} for {scenario_name}...")
    print(f"Results will be saved in: {save_dir}")

    # Initialize the trainer
    trainer = ModelTrainer(dataset, target_column, model)
    trainer.split_data()

    # Train and tune the model
    if search_type:
        best_params = trainer.tune_and_train(search_type, search_config, param_grid)
    else:
        trainer.train()  # Train without hyperparameter tuning

    # Evaluate the model
    metric_test = trainer.test_and_evaluate(save_dir=save_dir)

    # Plot results for the current scenario
    trainer.plot_results(save_dir=save_dir)

    print(f"Scenario {scenario_name} with {search_type} completed and results saved.\n")

    return metric_test, best_params
```

Figure 127: Run scenarios of HPO function

```python
def save_models(model, model_name, search_method):
    os.makedirs("./best_models", exist_ok=True)
    os.makedirs(f"./best_models/{model_name}", exist_ok=True)
    os.makedirs(f"./best_models/{model_name}/{search_method}", exist_ok=True)
    with open(f"./best_models/{model_name}/{search_method}/model_{search_method}.pkl", "wb") as f:
        pkl.dump(model, f)

def save_model_info(model_info, model_name, search_method):
    os.makedirs("./best_models", exist_ok=True)
    os.makedirs(f"./best_models/{model_name}", exist_ok=True)
    os.makedirs(f"./best_models/{model_name}/{search_method}", exist_ok=True)
    with open(f"./best_models/{model_name}/{search_method}/model_info_{search_method}.json", "w") as
f:      json.dump(model_info, f)
```

Figure 128: Saving models functions

6.10 HPO

6.10.1 Naive Bayes

```
naive_bayes_grid_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "NaiveBayes", "grid", search_config_grid,
get_search_hyperparameters("NaiveBayes"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        naive_bayes_grid_search['accuracy'] = best_score
        naive_bayes_grid_search['params'] = best_params
        naive_bayes_grid_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

nb = GaussianNB(**naive_bayes_grid_search['params'])
best_scenario = pd.read_csv(naive_bayes_grid_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, nb)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "NaiveBayes", "grid")
save_model_info(naive_bayes_grid_search, "NaiveBayes", "grid")
```

Figure 129: Naive Bayes GridSearch

```python
naive_bayes_random_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "NaiveBayes", "random", search_config_random,
get_search_hyperparameters("NaiveBayes"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        naive_bayes_random_search['accuracy'] = best_score
        naive_bayes_random_search['params'] = best_params
        naive_bayes_random_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

nb = GaussianNB(**naive_bayes_random_search['params'])
best_scenario = pd.read_csv(naive_bayes_random_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, nb)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "NaiveBayes", "random")
save_model_info(naive_bayes_random_search, "NaiveBayes", "random")
```

Figure 130: Naive Bayes Random Search

```
naive_bayes_optuna_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "NaiveBayes", "optuna", search_config_optuna,
get_search_hyperparameters("NaiveBayes"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        naive_bayes_optuna_search['accuracy'] = best_score
        naive_bayes_optuna_search['params'] = best_params
        naive_bayes_optuna_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

nb = GaussianNB(**naive_bayes_optuna_search['params'])
best_scenario = pd.read_csv(naive_bayes_optuna_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, nb)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "NaiveBayes", "optuna")
save_model_info(naive_bayes_optuna_search, "NaiveBayes", "optuna")
```

Figure 131: Naive Bayes Optuna

6.10.2 SVM

```
svm_grid_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "SVM", "grid", search_config_grid,
get_search_hyperparameters("SVM"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        svm_grid_search['accuracy'] = best_score
        svm_grid_search['params'] = best_params
        svm_grid_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

svc = SVC(**svm_grid_search['params'])
best_scenario = pd.read_csv(svm_grid_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, svc)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "SVM", "grid")
save_model_info(svm_grid_search, "SVM", "grid")
```

Figure 132: SVM GridSearch

```
svm_random_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "SVM", "random", search_config_random,
get_search_hyperparameters("SVM"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        svm_random_search['accuracy'] = best_score
        svm_random_search['params'] = best_params
        svm_random_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

svc = SVC(**svm_random_search['params'])
best_scenario = pd.read_csv(svm_random_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, svc)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "SVM", "random")
save_model_info(svm_random_search, "SVM", "random")
```

Figure 133: SVM Random Searchh

```
svm_optuna_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "SVM", "optuna", search_config_optuna,
get_search_hyperparameters("SVM"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        svm_optuna_search['accuracy'] = best_score
        svm_optuna_search['params'] = best_params
        svm_optuna_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

svc = SVC(**svm_optuna_search['params'])
best_scenario = pd.read_csv(svm_optuna_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, svc)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "SVM", "optuna")
save_model_info(svm_optuna_search, "SVM", "optuna")
```

Figure 134: SVM Optuna

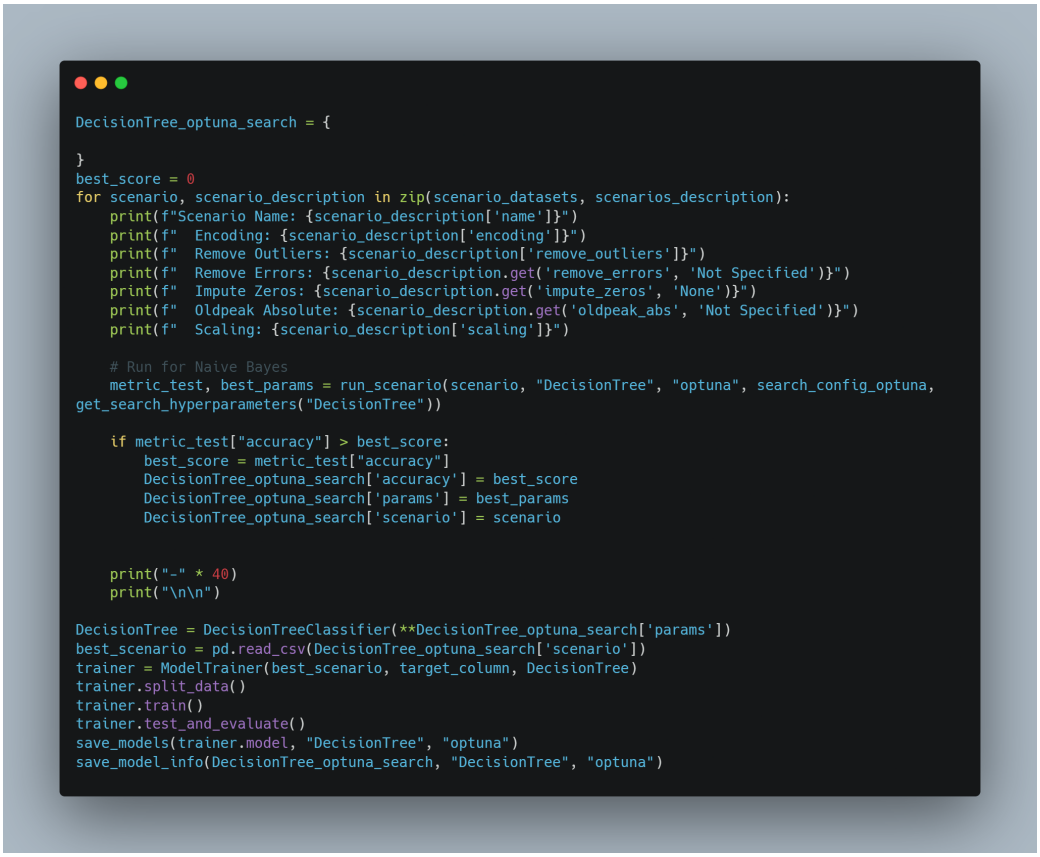
6.10.3 Decision Tree

```python
DecisionTree_grid_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "DecisionTree", "grid", search_config_grid,
get_search_hyperparameters("DecisionTree"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        DecisionTree_grid_search['accuracy'] = best_score
        DecisionTree_grid_search['params'] = best_params
        DecisionTree_grid_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

DecisionTree = DecisionTreeClassifier(**DecisionTree_grid_search['params'])
best_scenario = pd.read_csv(DecisionTree_grid_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, DecisionTree)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "DecisionTree", "grid")
save_model_info(DecisionTree_grid_search, "DecisionTree", "grid")
```

Figure 135: DT GridSearch

```
DecisionTree_random_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "DecisionTree", "random", search_config_random,
get_search_hyperparameters("DecisionTree"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        DecisionTree_random_search['accuracy'] = best_score
        DecisionTree_random_search['params'] = best_params
        DecisionTree_random_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

DecisionTree = DecisionTreeClassifier(**DecisionTree_random_search['params'])
best_scenario = pd.read_csv(DecisionTree_random_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, DecisionTree)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "DecisionTree", "random")
save_model_info(DecisionTree_random_search, "DecisionTree", "random")
```

Figure 136: DT Random Search



```
DecisionTree_optuna_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "DecisionTree", "optuna", search_config_optuna,
get_search_hyperparameters("DecisionTree"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        DecisionTree_optuna_search['accuracy'] = best_score
        DecisionTree_optuna_search['params'] = best_params
        DecisionTree_optuna_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

DecisionTree = DecisionTreeClassifier(**DecisionTree_optuna_search['params'])
best_scenario = pd.read_csv(DecisionTree_optuna_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, DecisionTree)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "DecisionTree", "optuna")
save_model_info(DecisionTree_optuna_search, "DecisionTree", "optuna")
```

Figure 137: DT Optuna

6.10.4 KNN

```
KNN_grid_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "KNN", "grid", search_config_grid,
get_search_hyperparameters("KNN"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        KNN_grid_search['accuracy'] = best_score
        KNN_grid_search['params'] = best_params
        KNN_grid_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

knn = KNeighborsClassifier(**KNN_grid_search['params'])
best_scenario = pd.read_csv(KNN_grid_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, knn)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "KNN", "grid")
save_model_info(KNN_grid_search, "KNN", "grid")
```

Figure 138: KNN GridSearch

```
KNN_random_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "KNN", "random", search_config_random,
get_search_hyperparameters("KNN"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        KNN_random_search['accuracy'] = best_score
        KNN_random_search['params'] = best_params
        KNN_random_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

knn = KNeighborsClassifier(**KNN_random_search['params'])
best_scenario = pd.read_csv(KNN_random_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, knn)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "KNN", "random")
save_model_info(KNN_random_search, "KNN", "random")
```

Figure 139: KNN Random Search

```
KNN_optuna_search = {

}
best_score = 0
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not Specified')}")
    print(f"  Scaling: {scenario_description['scaling']}")

    # Run for Naive Bayes
    metric_test, best_params = run_scenario(scenario, "KNN", "optuna", search_config_optuna,
get_search_hyperparameters("KNN"))

    if metric_test["accuracy"] > best_score:
        best_score = metric_test["accuracy"]
        KNN_optuna_search['accuracy'] = best_score
        KNN_optuna_search['params'] = best_params
        KNN_optuna_search['scenario'] = scenario


    print("-" * 40)
    print("\n\n")

knn = KNeighborsClassifier(**KNN_optuna_search['params'])
best_scenario = pd.read_csv(KNN_optuna_search['scenario'])
trainer = ModelTrainer(best_scenario, target_column, knn)
trainer.split_data()
trainer.train()
trainer.test_and_evaluate()
save_models(trainer.model, "KNN", "optuna")
save_model_info(KNN_optuna_search, "KNN", "optuna")
```

Figure 140: KNN Optuna

6.11 Comparing Models Utils

```python
# Directories where models are saved
base_dir = "./best_models"

# Dynamically retrieve optimization methods and algorithms from directory structure
def get_dir_names(base_dir):
    """Retrieve directories (optimization methods or algorithms) under the given base directory."""
    if os.path.exists(base_dir):
        return [name for name in os.listdir(base_dir) if os.path.isdir(os.path.join(base_dir, name))]
    return []

# Retrieve algorithms and optimization methods
algorithms = get_dir_names(base_dir)
optimization_methods = set()
for algo in algorithms:
    optimization_methods.update(get_dir_names(os.path.join(base_dir, algo)))

optimization_methods = list(optimization_methods)

# Helper to load model info
def load_model_info(algorithm, optimization_method):
    """Load the model information from JSON file."""
    path = os.path.join(base_dir, algorithm, optimization_method,
f"model_info_{optimization_method}.json")
    if os.path.exists(path):
        with open(path, "r") as file:
            return json.load(file)
    return None
best_models_per_method = {method: None for method in optimization_methods}
highest_accuracy_overall = 0
best_overall_models = []
all_model_combinations = list(itertools.product(algorithms, optimization_methods))
all_model_combinations = {f"{algo}_{opt}": load_model_info(algo, opt)["accuracy"] for algo, opt in
all_model_combinations}

# Colors for each model type
colors = {
    "DecisionTree": "skyblue",
    "KNN": "orange",
    "NaiveBayes": "green",
    "SVM": "purple"
}

# Assign colors based on the model type
bar_colors = [colors[key.split("_")[0]] for key in all_model_combinations.keys()]

# Plotting
plt.figure(figsize=(12, 6))
bars = plt.bar(all_model_combinations.keys(), all_model_combinations.values(), color=bar_colors)

# Adding the value on top of each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2.0, height, f"{height:.2f}", ha="center", va="bottom",
fontsize=10)

plt.title("Accuracy Comparison for All Model Combinations")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.xticks(rotation=45, ha="right")
plt.show()
```

Figure 141: Utils for comparing models

```
print("Evaluating Models Across Optimization Methods:\n")
for method in optimization_methods:
    best_models_for_method = []
    best_accuracy_for_method = 0

    for algo in algorithms:
        model_info = load_model_info(algo, method)
        if model_info:
            accuracy = model_info["accuracy"]
            print(f"Method: {method.upper()}, Algorithm: {algo}, Accuracy: {accuracy:.4f}")

            # Update the best models for the current method
            if accuracy > best_accuracy_for_method:
                best_accuracy_for_method = accuracy
                best_models_for_method = [algo]
            elif accuracy == best_accuracy_for_method:
                best_models_for_method.append(algo)

            # Check and update the overall best models
            if accuracy > highest_accuracy_overall:
                highest_accuracy_overall = accuracy
                best_overall_models = [
                    {
                        "method": method,
                        "algorithm": algo,
                        "accuracy": accuracy,
                        "scenario": model_info["scenario"],
                    }
                ]
            elif accuracy == highest_accuracy_overall:
                best_overall_models.append(
                    {
                        "method": method,
                        "algorithm": algo,
                        "accuracy": accuracy,
                        "scenario": model_info["scenario"],
                    }
                )

    # Save the best model(s) for this method
    if best_models_for_method:
        best_models_per_method[method] = {
            "algorithms": best_models_for_method,
            "accuracy": best_accuracy_for_method,
        }
        algorithms_str = ", ".join(best_models_for_method)
        print(f"Best Model for {method.upper()}: {algorithms_str} (Accuracy:
{best_accuracy_for_method:.4f})\n")
```

Figure 142: Evaluating Models across optimization methods

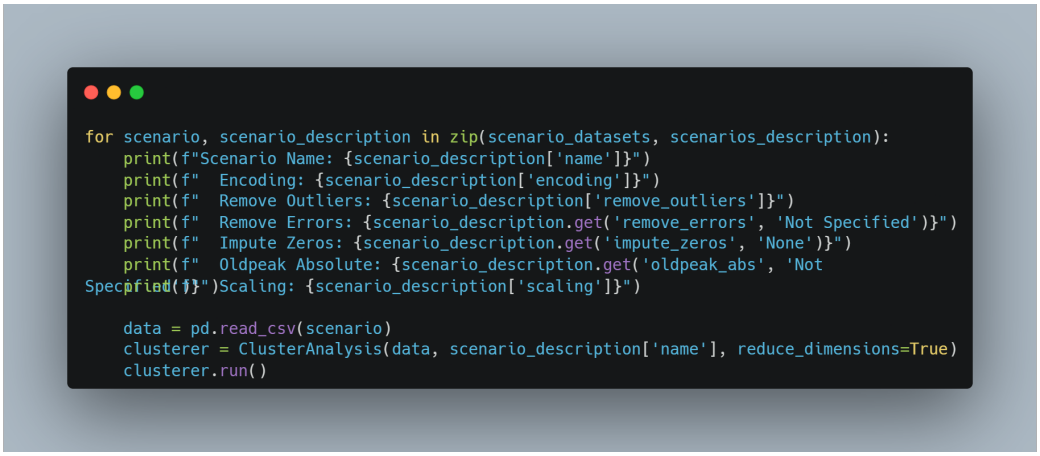
```
print("Best Models Per Optimization Method:\n")
for method, details in best_models_per_method.items():
    algorithms_str = ", ".join(details['algorithms'])
    accuracy = details['accuracy']
    print(f"Optimization Method: {method.upper()}")
    print(f"   Best Algorithms: {algorithms_str}")
    print(f"   Accuracy: {accuracy}")

    for algo in details['algorithms']:
        algo_path = os.path.join(base_dir, algo, method)
        model_info_path = os.path.join(algo_path,
f"model_info_{method}.json")
        pkl_path = os.path.join(algo_path, f"model_{method}.pkl")

        scenario = "N/A"
        parameters = "N/A"
        if os.path.exists(model_info_path):
            with open(model_info_path, 'r') as file:
                model_info = json.load(file)
                scenario = model_info.get("scenario", "N/A")
                parameters = model_info.get("params", "N/A")

        print(f"   Parameters: {parameters}")
        print(f"   Scenario: {scenario}")
        print(f"   PKL Path: {'./' + pkl_path}\n")
```

Figure 143: Best Models / Optimization Method

6.12 Cluster Analysis

```python
class ClusterAnalysis:
    def __init__(self, data, name, reduce_dimensions=False):
        self.ytrue = data["HeartDisease"]

        out_dir = 'Clustering'
        self.data = data.drop(columns=["HeartDisease"])
        self.methods = ["complete", "average", "single"]
        self.types = ["euclidean", "manhattan", "cosine", "correlation"]
        if reduce_dimensions:
            self.data = self._reduce_dimensions()
            out_dir = f"{out_dir}_Reduced"
        self.save_dir = f"{out_dir}/{name}"
        os.makedirs(self.save_dir, exist_ok=True)



    def _reduce_dimensions(self, method="PCA", n_components=2):
        if method == "PCA":
            pca = PCA(n_components=n_components)
            self.data = pd.DataFrame(pca.fit_transform(self.data))
            self.data.columns = [f"PC{i}" for i in range(1, n_components + 1)]
        return self.data


    def plot_dendrogram(self, method="single", type="euclidean"):
        if type == "manhattan":
            type = "cityblock"
        Z = linkage(self.data, method=method, metric=type)
        plt.figure(figsize=(25, 10))
        plt.title('Hierarchical Clustering Dendrogram - Method: {} - Type: {}'.format(method,
type))  dendrogram(
            Z,
            leaf_rotation=45.,
            leaf_font_size=3.,
        )
        plt.savefig(f"{self.save_dir}/Dendrogram_{method}_{type}.png")

    def plot_clusters(self, method="single", type="euclidean"):
        model = AgglomerativeClustering(linkage=method, metric=type)
        model.fit(self.data)
        labels = model.labels_
        self.data["Cluster"] = labels
        sns.pairplot(self.data, hue="Cluster")
        plt.savefig(f"{self.save_dir}/Clusters_{method}_{type}.png")

    def validate_clusters(self):
        accuracy = accuracy_score(self.ytrue, self.data["Cluster"])
        print(f"        Accuracy of Clustering: {accuracy}")


    def run(self):
        for method in self.methods:
            for type in self.types:
                print(f"    Method: {method} - Type: {type}")
                self.plot_dendrogram(method, type)
                self.plot_clusters(method, type)
                self.validate_clusters()
```

Figure 144: Class ClusterAnalysis



```
for scenario, scenario_description in zip(scenario_datasets, scenarios_description):
    print(f"Scenario Name: {scenario_description['name']}")
    print(f"  Encoding: {scenario_description['encoding']}")
    print(f"  Remove Outliers: {scenario_description['remove_outliers']}")
    print(f"  Remove Errors: {scenario_description.get('remove_errors', 'Not Specified')}")
    print(f"  Impute Zeros: {scenario_description.get('impute_zeros', 'None')}")
    print(f"  Oldpeak Absolute: {scenario_description.get('oldpeak_abs', 'Not
Specified')}")print(f"Scaling: {scenario_description['scaling']}")

    data = pd.read_csv(scenario)
    clusterer = ClusterAnalysis(data, scenario_description['name'], reduce_dimensions=True)
    clusterer.run()
```

Figure 145: Running Clustering Scenarios

7 Dendrogram Analysis

7.1 Hierarchical Clustering and Proximity Measures

Hierarchical clustering was utilized to group the dataset into clusters based on proximity measures and linkage methods. This analysis provided insights into the natural groupings of the data and their relationship to the target variable (`HeartDisease`).

7.1.1 Hierarchical Clustering

Hierarchical clustering creates a dendrogram to visualize the hierarchical relationships between data points and iteratively groups them into clusters. The approach used in this project involved:

- **Linkage Methods**:
  - *Complete Linkage*: Maximizes the distance between points in different clusters.
  - *Average Linkage*: Considers the average distance between points in different clusters.
  - *Single Linkage*: Minimizes the distance between the closest points in different clusters.
- **Proximity Measures**:
  - *Euclidean Distance*: Measures the straight-line distance between points.
  - *Manhattan Distance (Cityblock)*: Considers the sum of absolute differences between coordinates.
  - *Cosine Similarity*: Measures the cosine of the angle between two vectors.
  - *Correlation Distance*: Captures how correlated data points are.

7.1.2 Dimensionality Reduction

- **Principal Component Analysis (PCA)** was optionally applied to reduce the dataset to two dimensions for visualization and computational efficiency.
- Reduced datasets enhanced interpretability without significant loss of information.

7.1.3 Visualization

- **Dendrograms**:
  - Represented the hierarchical structure of clusters for each combination of linkage method and proximity measure.
  - Helped identify the number of clusters and the relationships between data points.
- **Cluster Plots**:
  - Visualized the data points colored by their cluster assignments.
  - Allowed for the inspection of cluster separability and overlap.

7.1.4 Cluster Validation

- Cluster labels were compared with the true labels (`HeartDisease`) to evaluate clustering accuracy.
- **Accuracy Score**: Measured the agreement between cluster assignments and the actual target variable, though clustering is inherently unsupervised.

7.1.5 Process Workflow

- For each dataset scenario, clustering was performed twice:
  - **With Dimensionality Reduction**: Used PCA-reduced data for clustering and visualization.
  - **Without Dimensionality Reduction**: Used the full dataset to retain all features for clustering.
- Steps involved in clustering:
  1. Generate dendrograms using each linkage method and proximity measure.
  2. Perform clustering with `AgglomerativeClustering` and visualize the clusters.
  3. Validate the clusters by comparing them with the `HeartDisease` target variable.

7.1.6 Key Insights

- **Linkage Method Comparison**:
  - Complete and average linkage provided more balanced clusters.
  - Single linkage tended to create elongated clusters, often leading to poor separation.
- **Proximity Measure Comparison**:
  - Euclidean distance and Manhattan distance performed well for numeric features.
  - Cosine similarity and correlation distance were effective in scenarios with standardized data.
- **Dimensionality Reduction**:
  - PCA reduced computational complexity and facilitated better visual interpretation of clusters.
  - However, slight accuracy loss was observed due to the reduction in feature space.
- **Overall Observations**:
  - Dendrograms revealed the hierarchical structure and the optimal number of clusters.
  - Clusters aligned reasonably well with the `HeartDisease` labels, indicating meaningful groupings.
  - The choice of proximity measure and linkage method significantly influenced clustering results.

7.2 Visualization and Observations

7.2.1 Scenario 1_N

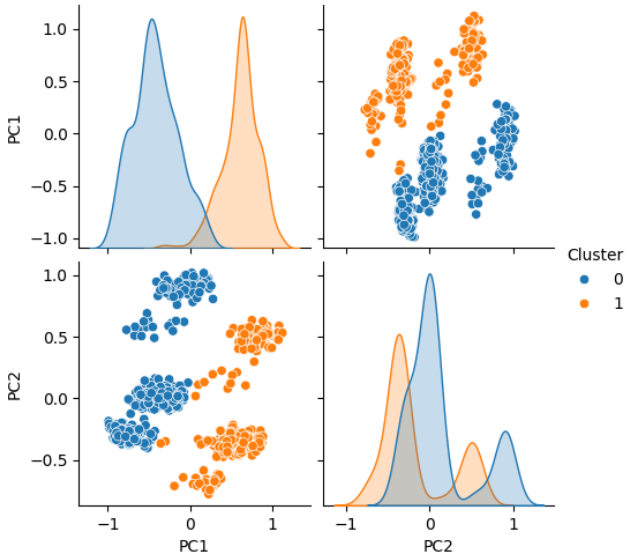


Figure 146: Clusters average correlation

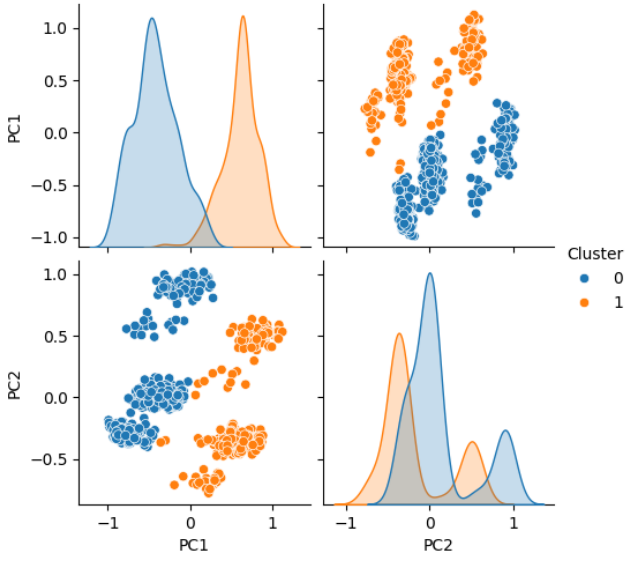


Figure 147: Clusters average cosine

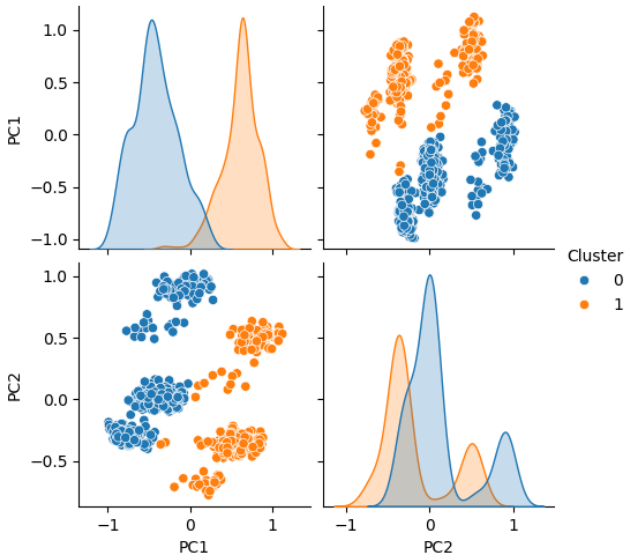


Figure 148: Clusters average euclidean

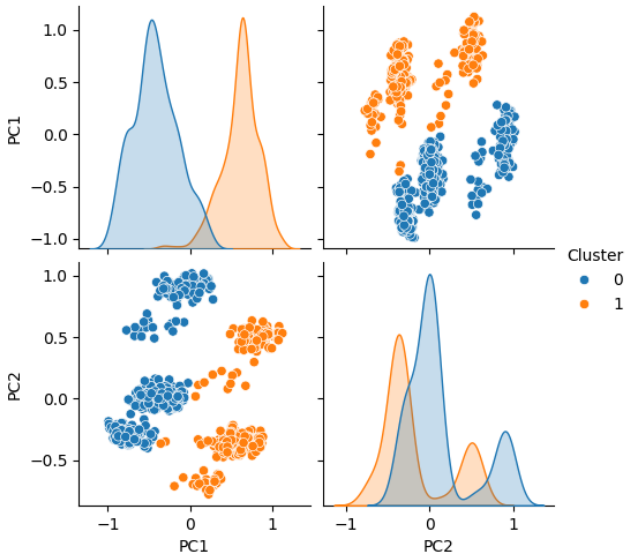


Figure 149: Clusters average manhattan

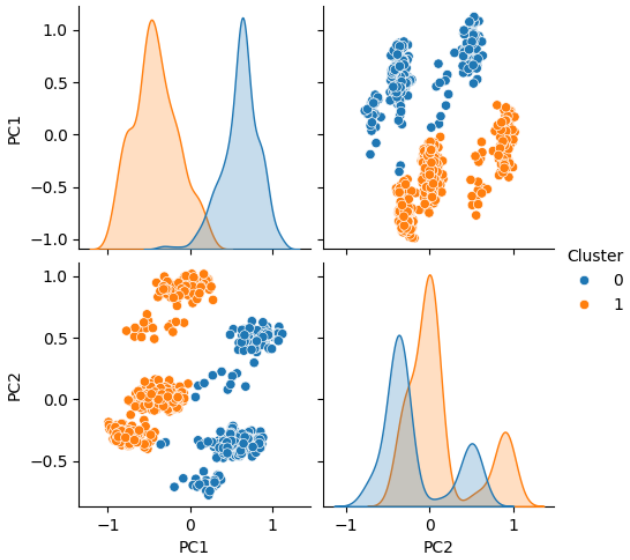


Figure 150: Clusters complete correlation

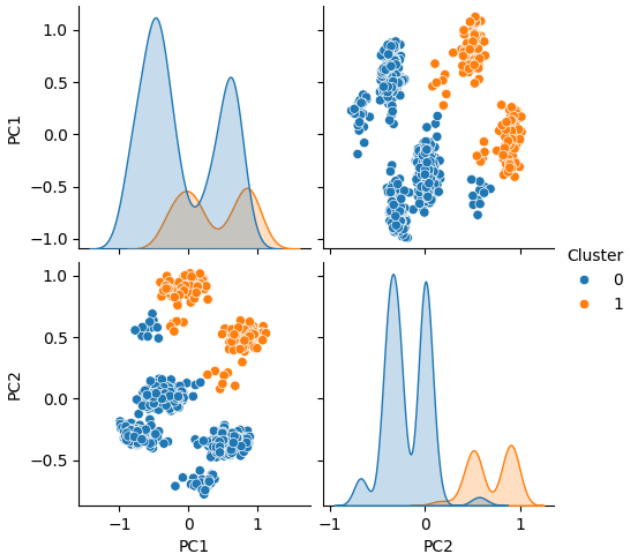


Figure 151: Clusters complete cosine

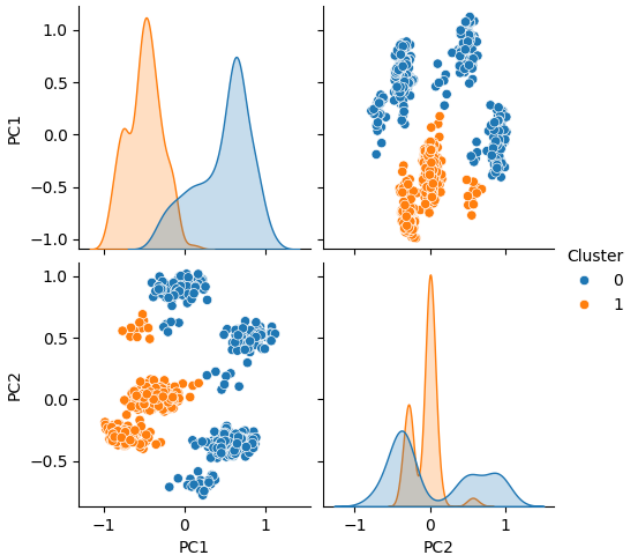


Figure 152: Clusters complete euclidean

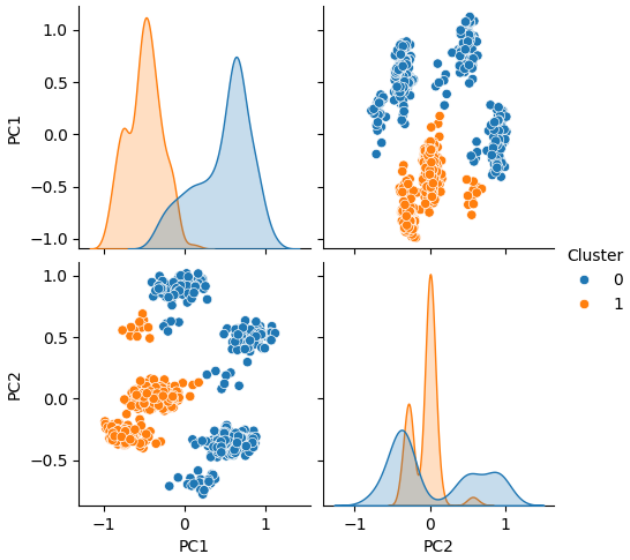


Figure 153: Clusters complete manhattan

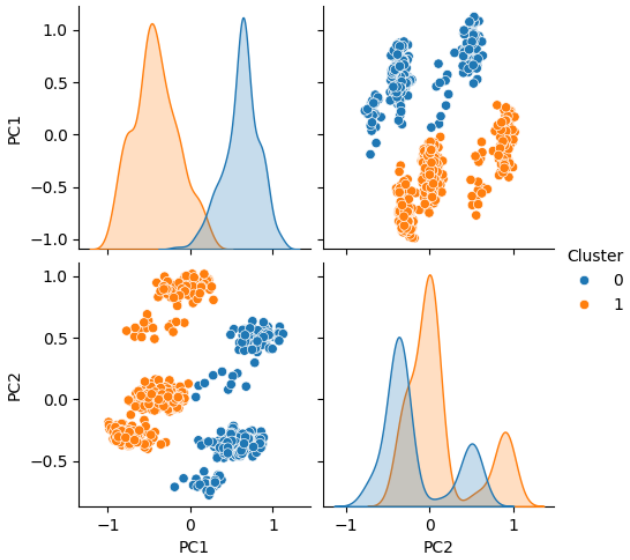


Figure 154: Clusters single correlation

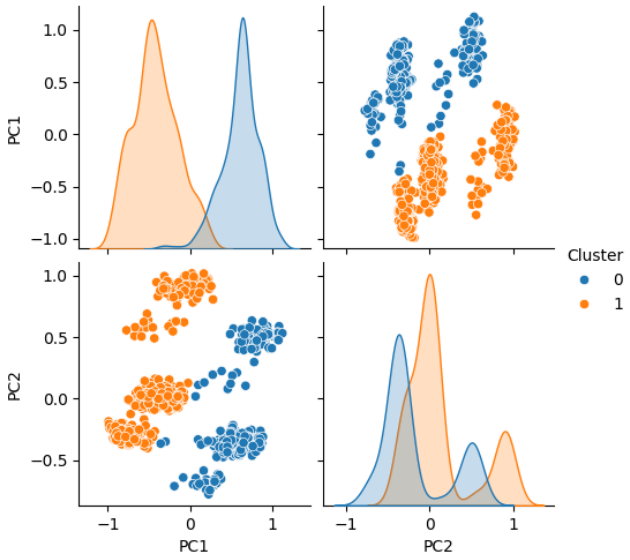


Figure 155: Clusters single cosine

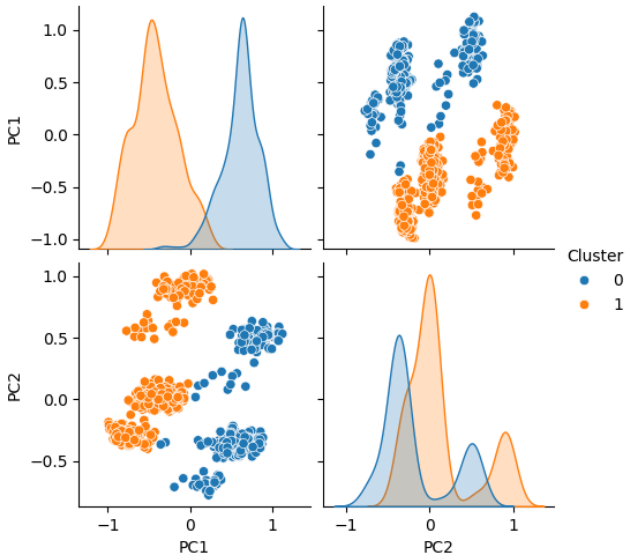


Figure 156: Clusters single euclidean

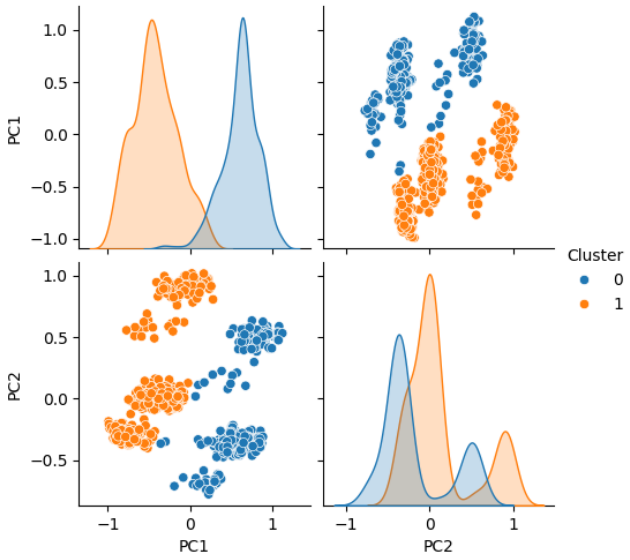


Figure 157: Clusters single manhattan

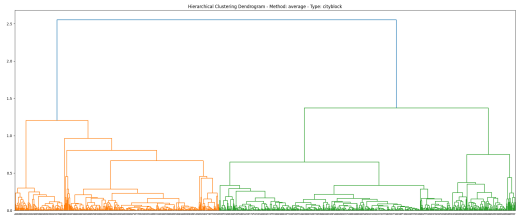


Figure 158: Dendrogram average cityblock

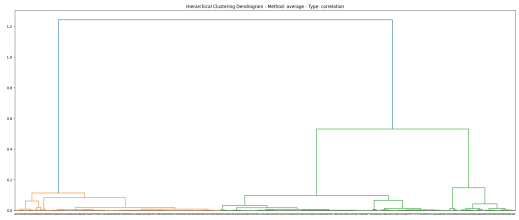


Figure 159: Dendrogram average correlation

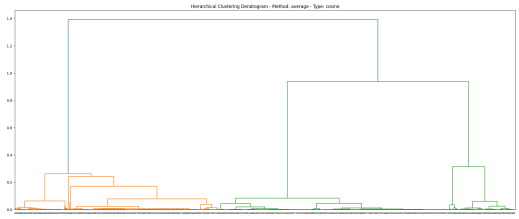


Figure 160: Dendrogram average cosine

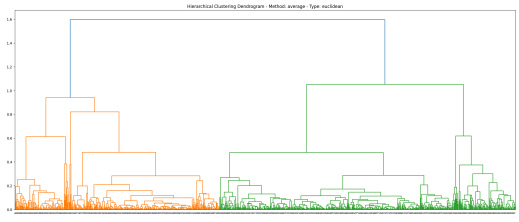


Figure 161: Dendrogram average euclidean

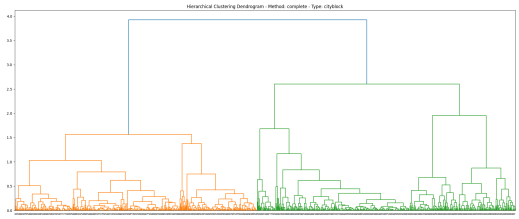


Figure 162: Dendrogram complete cityblock

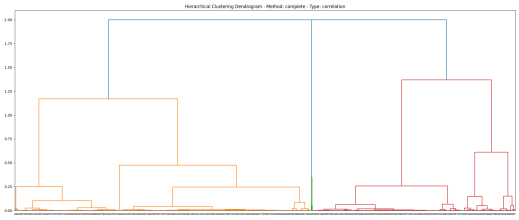


Figure 163: Dendrogram complete correlation

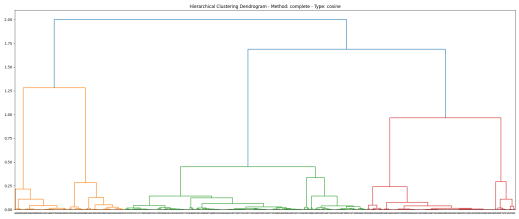


Figure 164: Dendrogram complete cosine

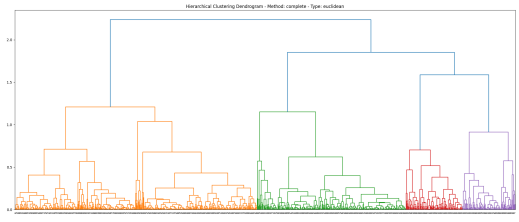


Figure 165: Dendrogram complete euclidean

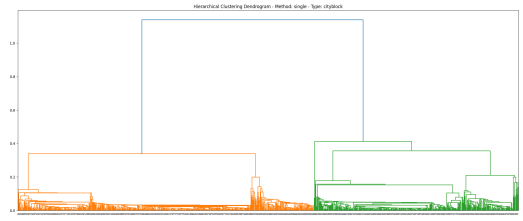


Figure 166: Dendrogram single cityblock

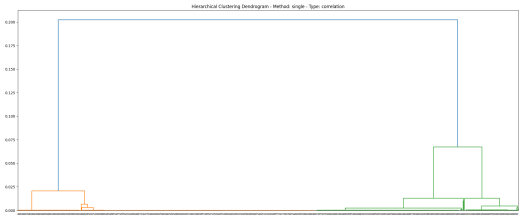


Figure 167: Dendrogram single correlation

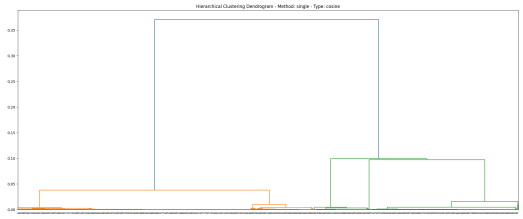


Figure 168: Dendrogram single cosine

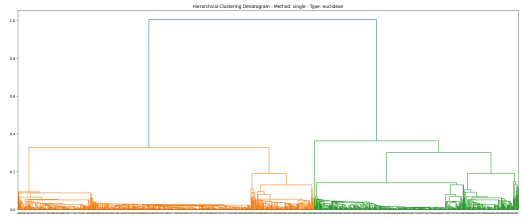


Figure 169: Dendrogram single euclidean

7.2.2 Scenario 1_S

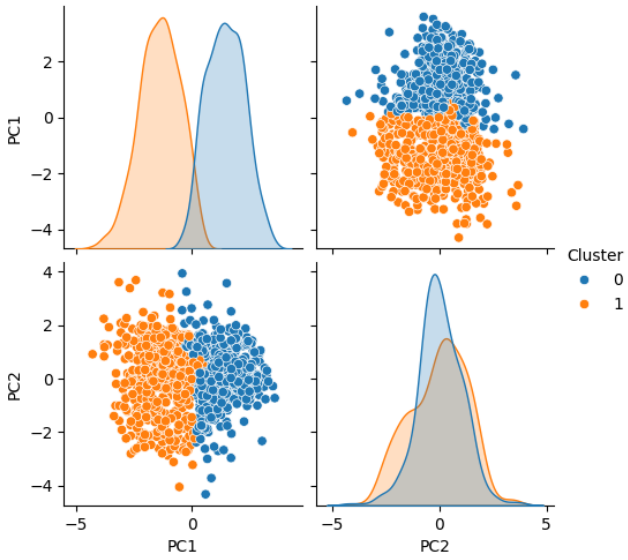


Figure 170: Clusters average correlation

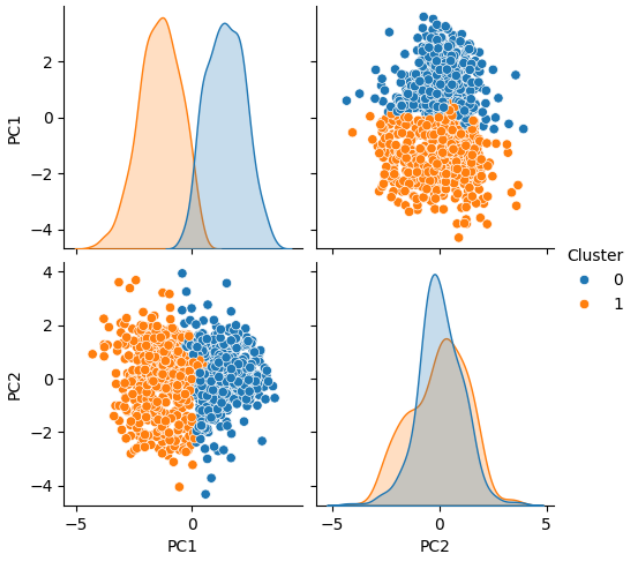


Figure 171: Clusters average cosine

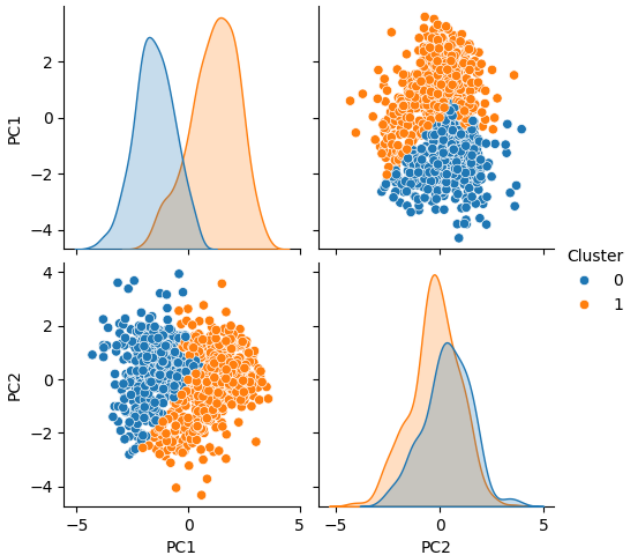


Figure 172: Clusters average euclidean

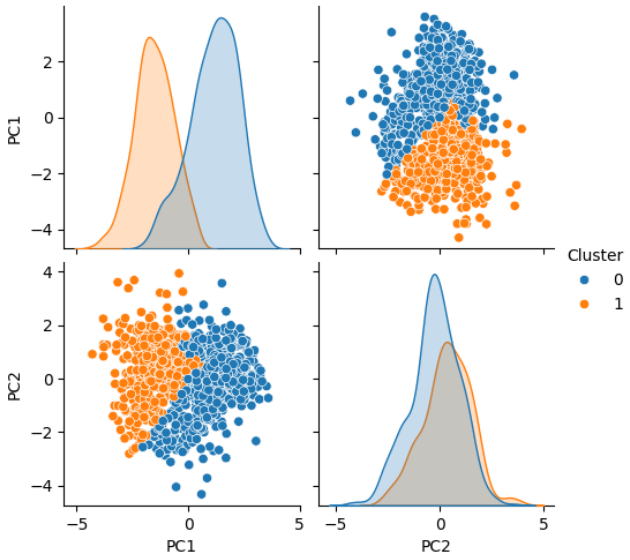


Figure 173: Clusters average manhattan

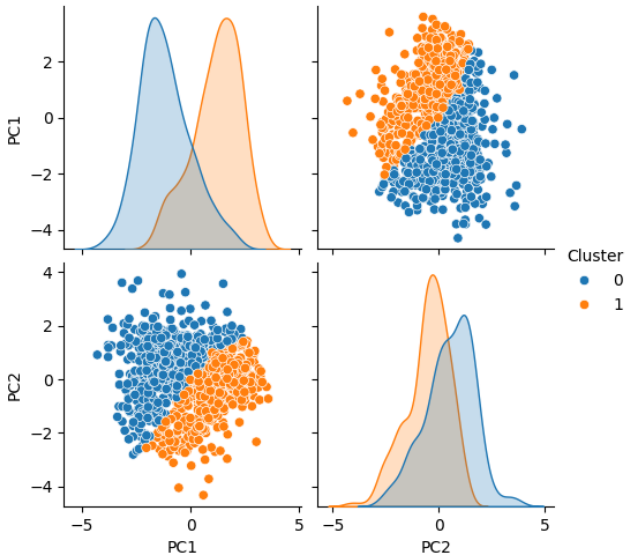


Figure 174: Clusters complete correlation

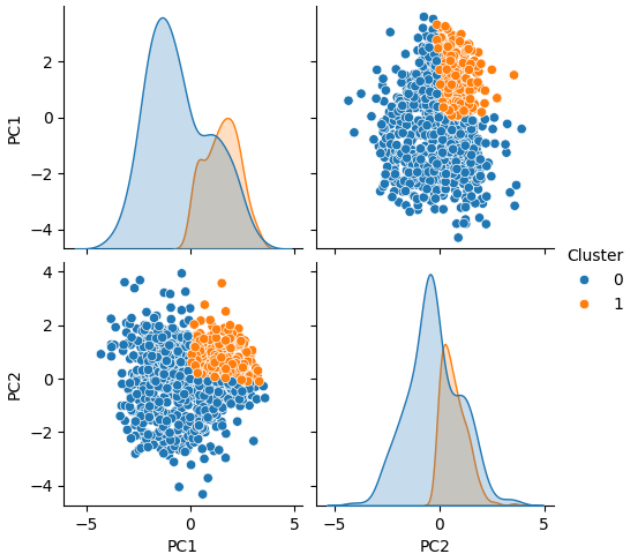


Figure 175: Clusters complete cosine

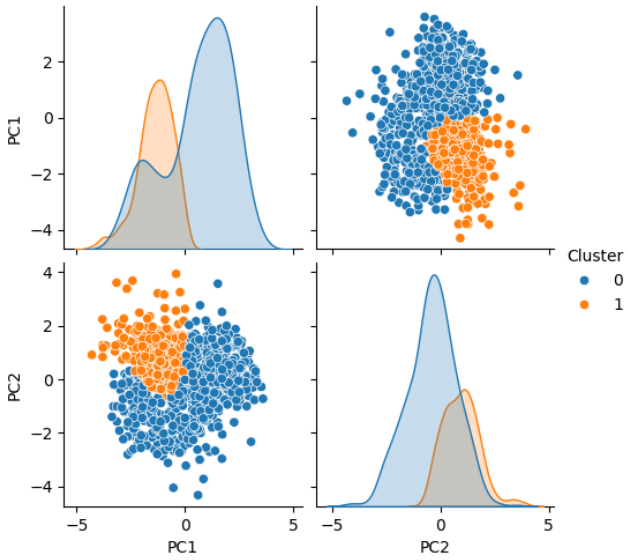


Figure 176: Clusters complete euclidean

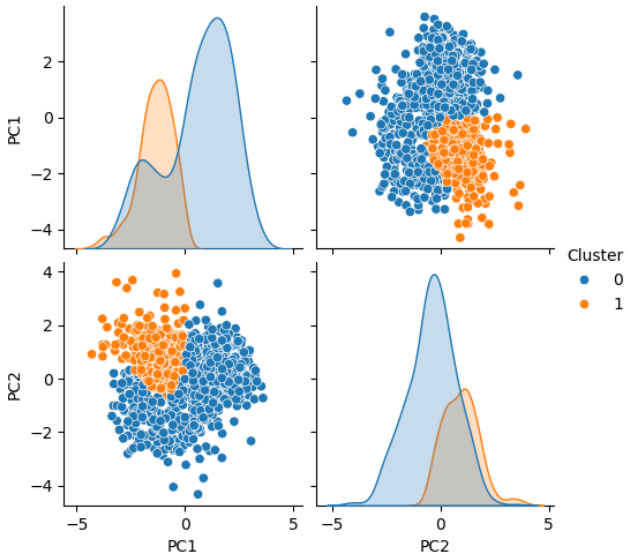


Figure 177: Clusters complete manhattan

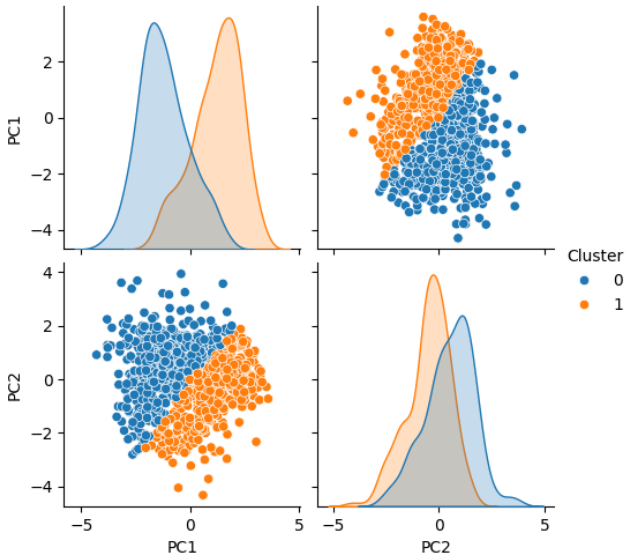


Figure 178: Clusters single correlation

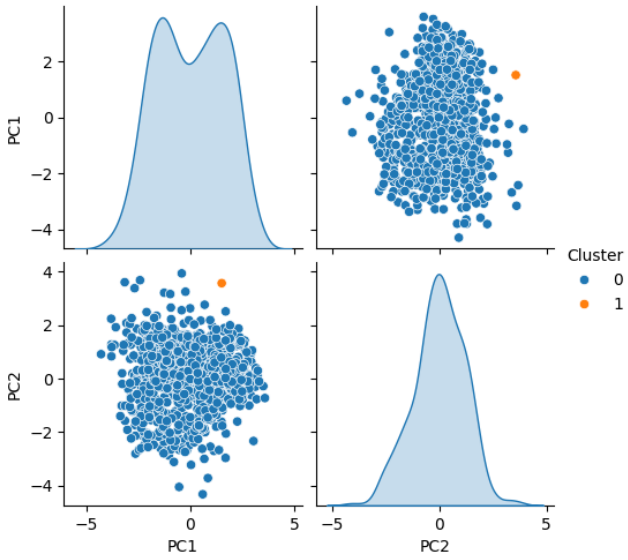


Figure 179: Clusters single cosine

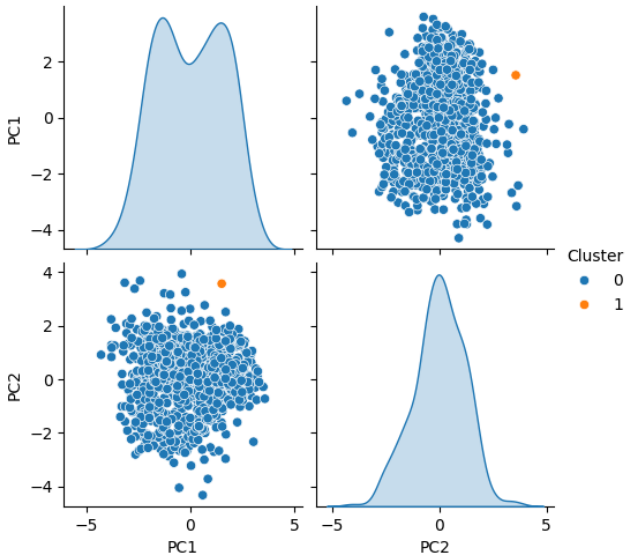


Figure 180: Clusters single euclidean

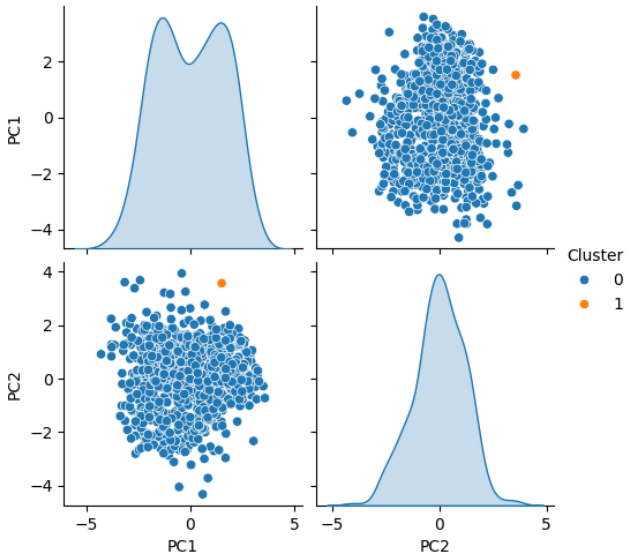


Figure 181: Clusters single manhattan

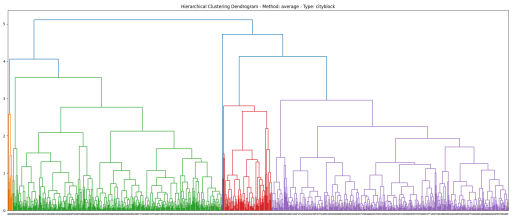


Figure 182: Dendrogram average cityblock

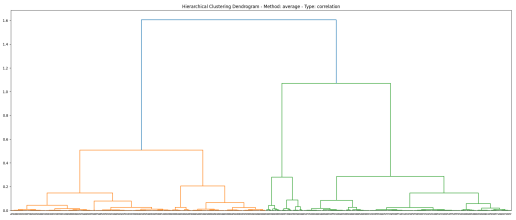


Figure 183: Dendrogram average correlation

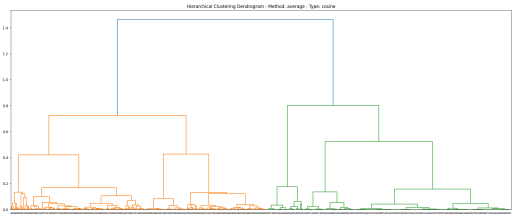


Figure 184: Dendrogram average cosine

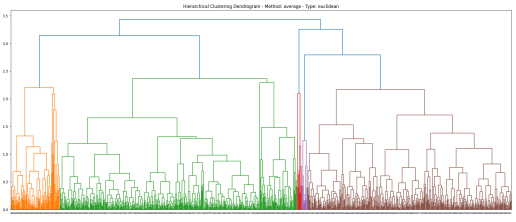


Figure 185: Dendrogram average euclidean

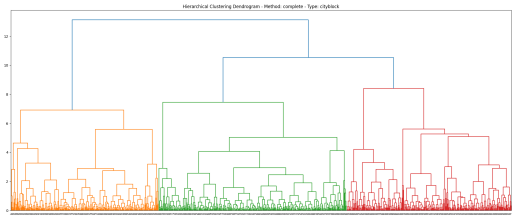


Figure 186: Dendrogram complete cityblock

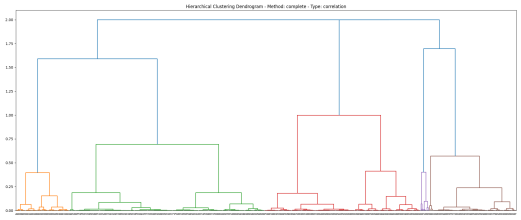


Figure 187: Dendrogram complete correlation

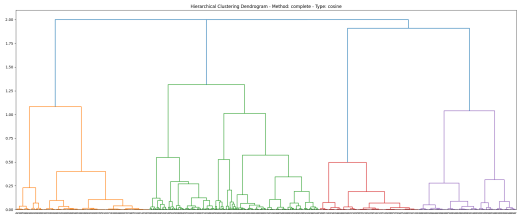


Figure 188: Dendrogram complete cosine

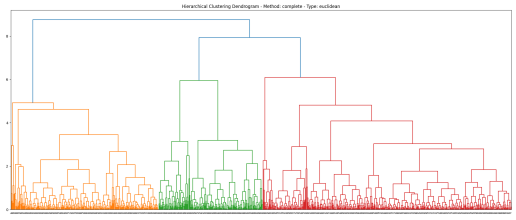


Figure 189: Dendrogram complete euclidean

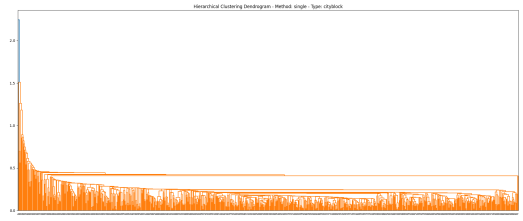


Figure 190: Dendrogram single cityblock

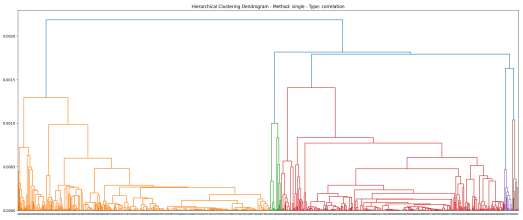


Figure 191: Dendrogram single correlation

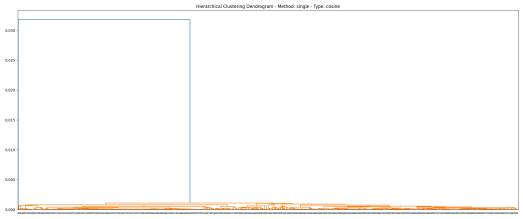


Figure 192: Dendrogram single cosine

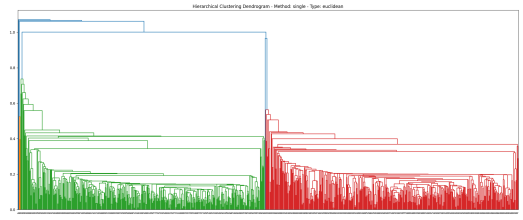


Figure 193: Dendrogram single euclidean

7.2.3 Scenario 2_N

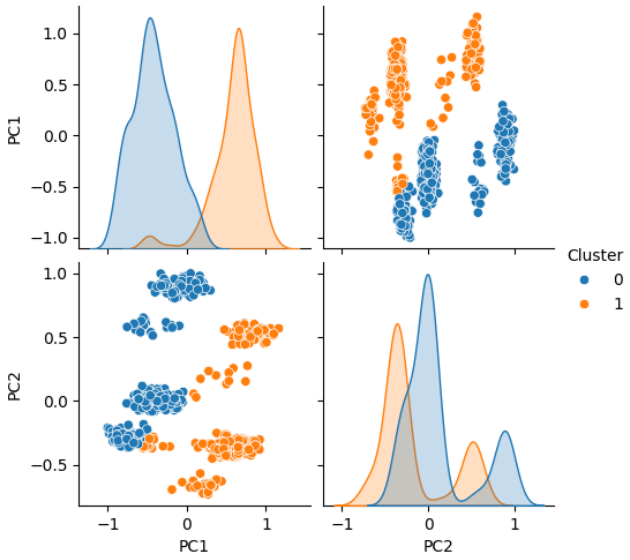


Figure 194: Clusters average correlation

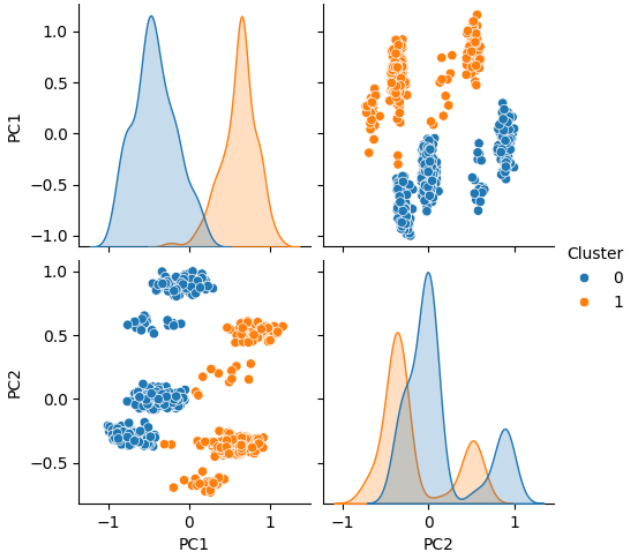


Figure 195: Clusters average cosine

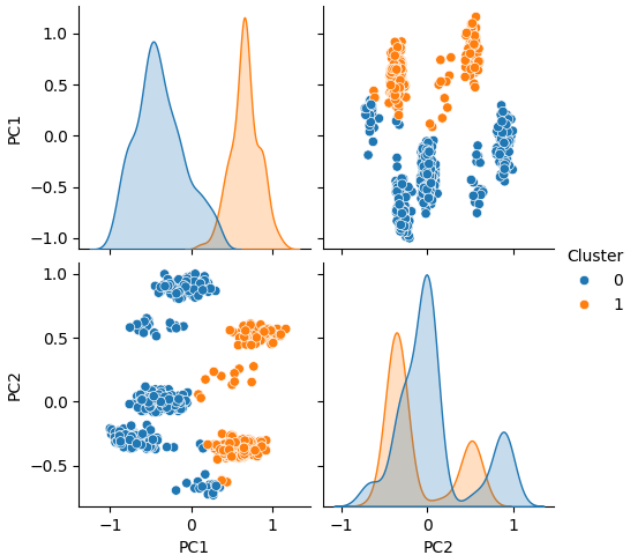


Figure 196: Clusters average euclidean

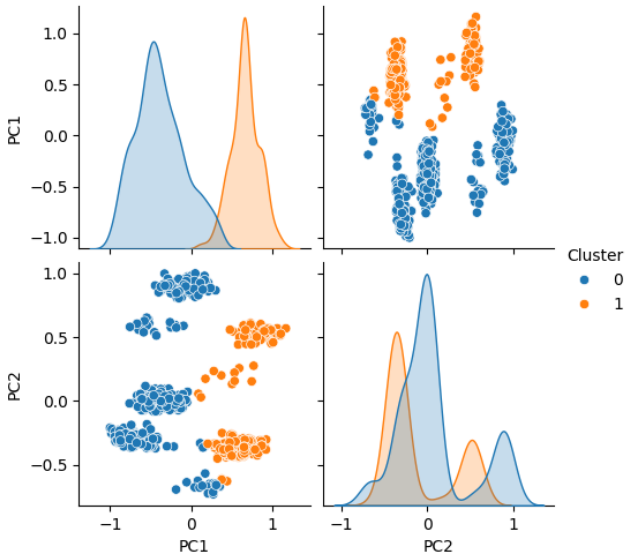


Figure 197: Clusters average manhattan

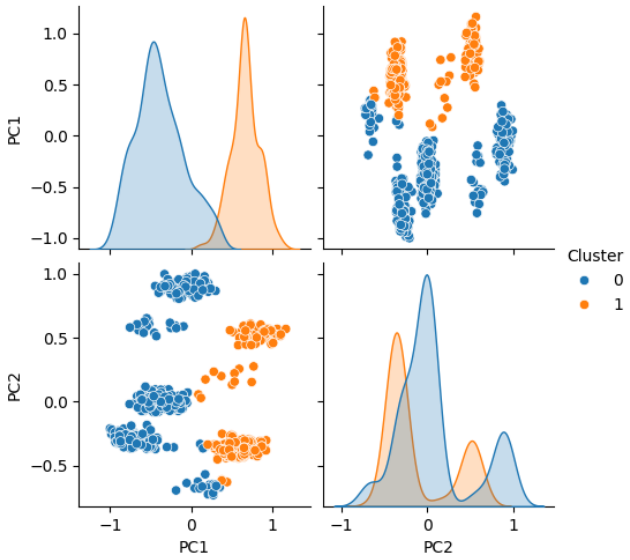


Figure 198: Clusters complete correlation

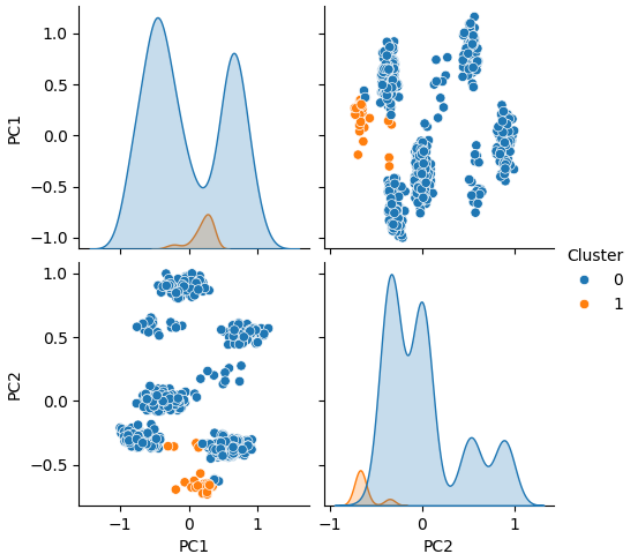


Figure 199: Clusters complete cosine

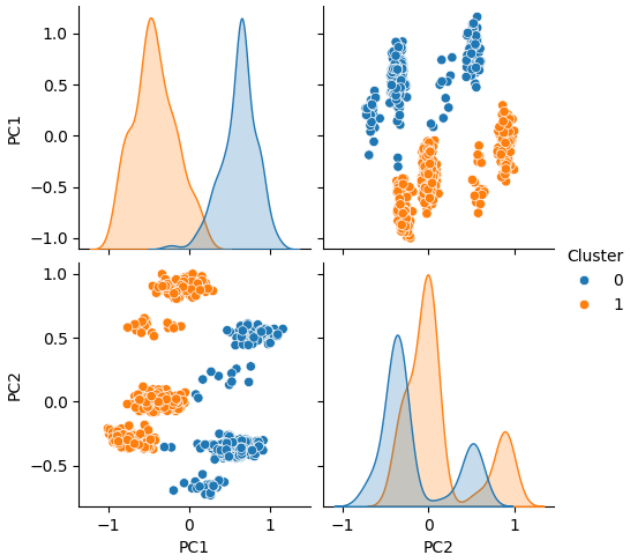


Figure 200: Clusters complete euclidean

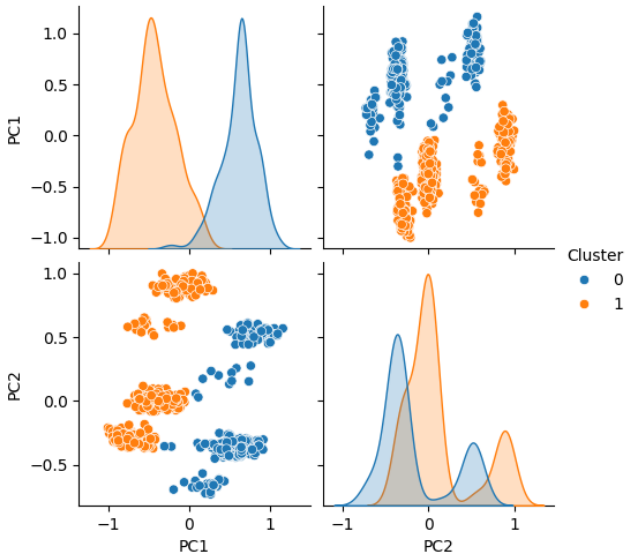


Figure 201: Clusters complete manhattan

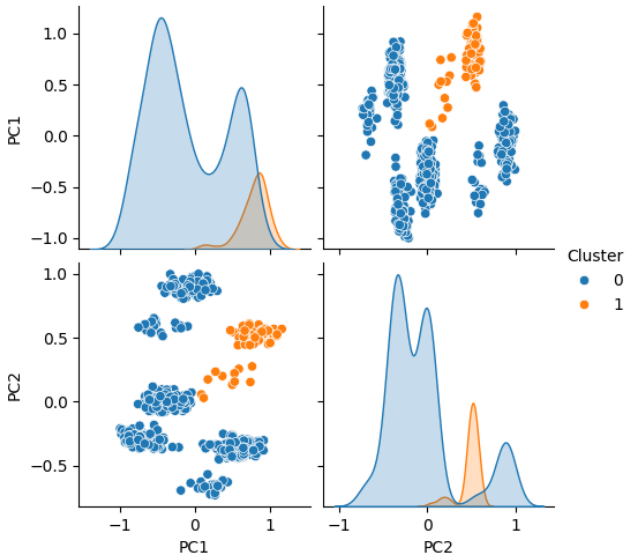


Figure 202: Clusters single correlation

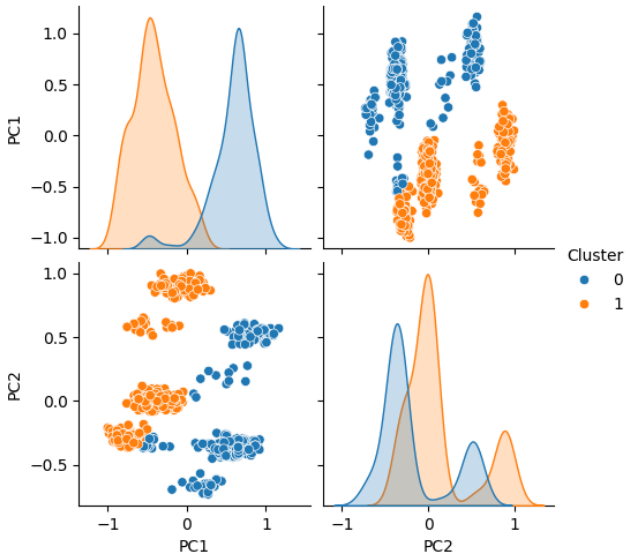


Figure 203: Clusters single cosine

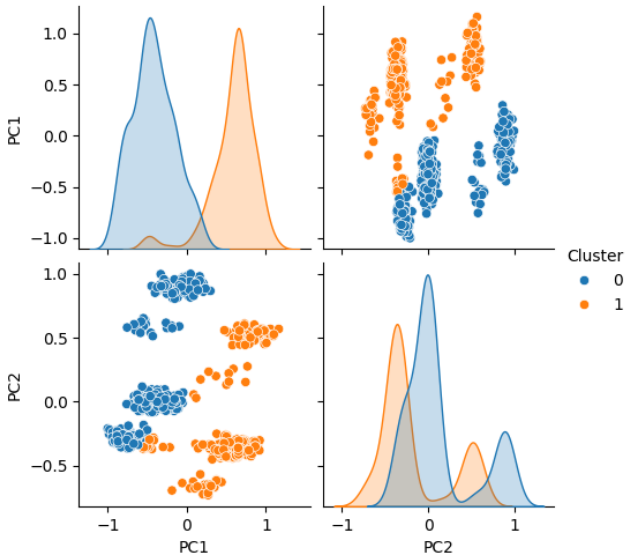


Figure 204: Clusters single euclidean

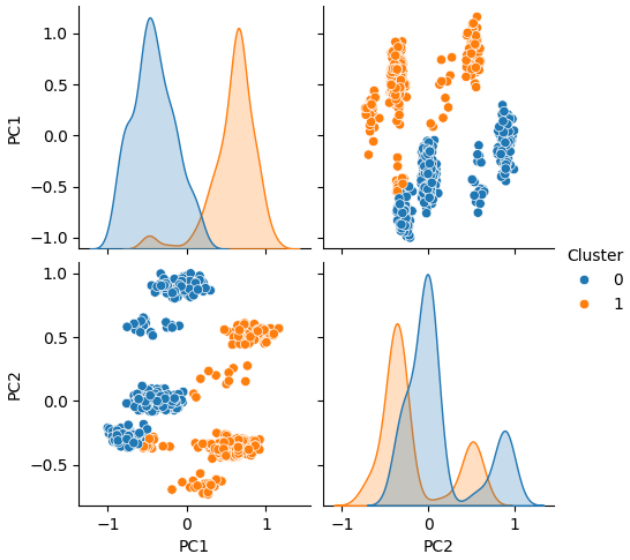


Figure 205: Clusters single manhattan

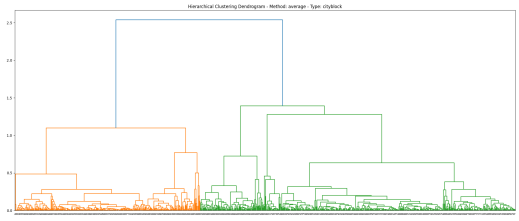


Figure 206: Dendrogram average cityblock

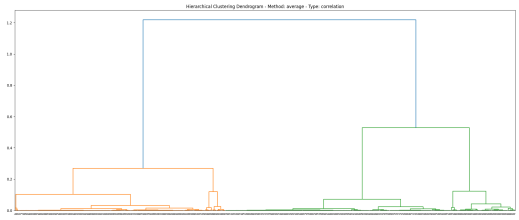


Figure 207: Dendrogram average correlation

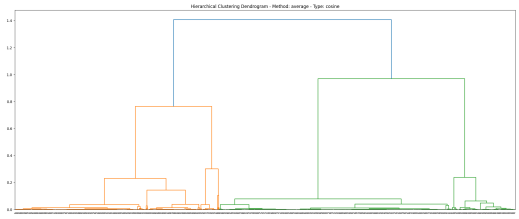


Figure 208: Dendrogram average cosine

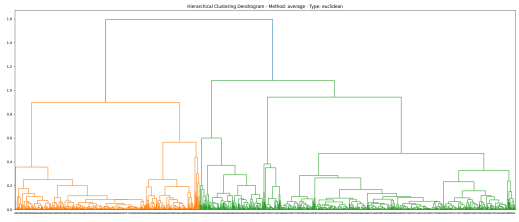


Figure 209: Dendrogram average euclidean

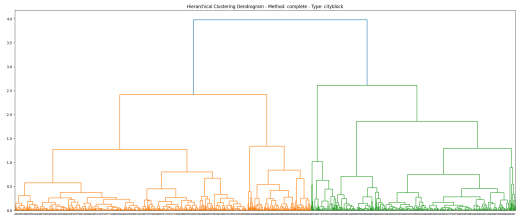


Figure 210: Dendrogram complete cityblock

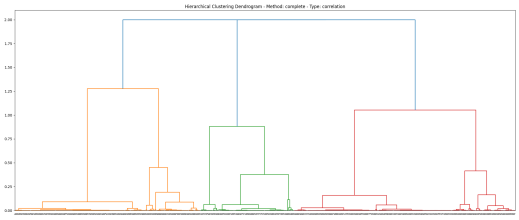


Figure 211: Dendrogram complete correlation

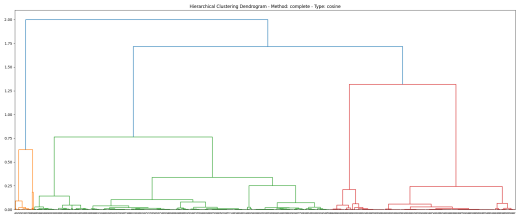


Figure 212: Dendrogram complete cosine

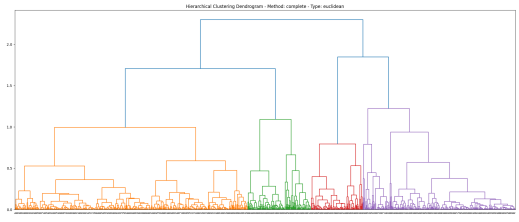


Figure 213: Dendrogram complete euclidean

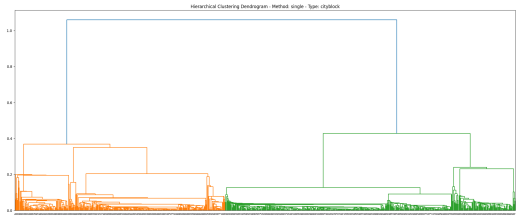


Figure 214: Dendrogram single cityblock

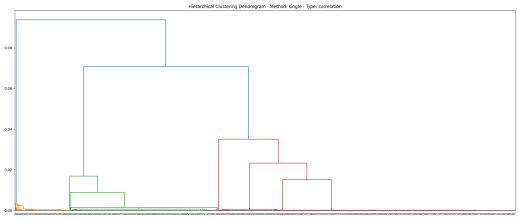


Figure 215: Dendrogram single correlation

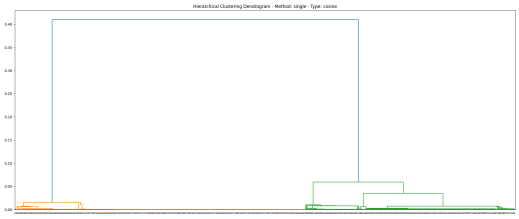


Figure 216: Dendrogram single cosine

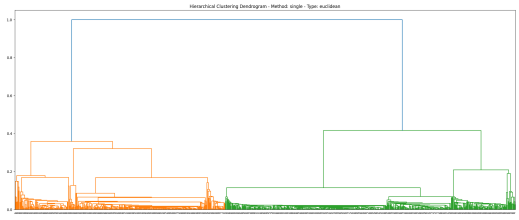


Figure 217: Dendrogram single euclidean

7.2.4 Scenario 2_S

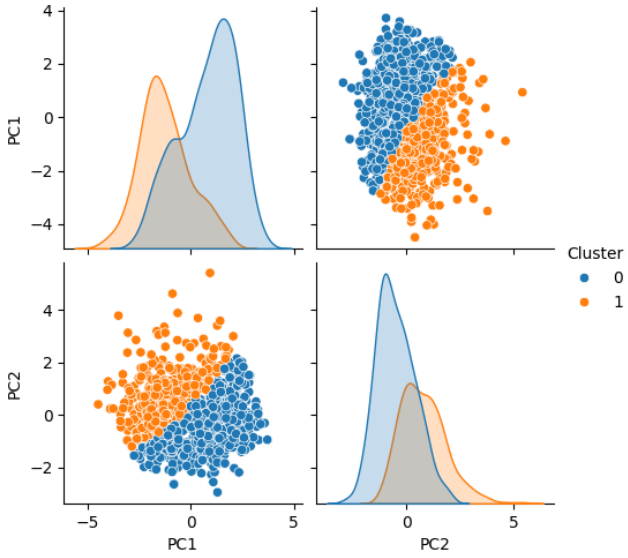


Figure 218: Clusters average correlation

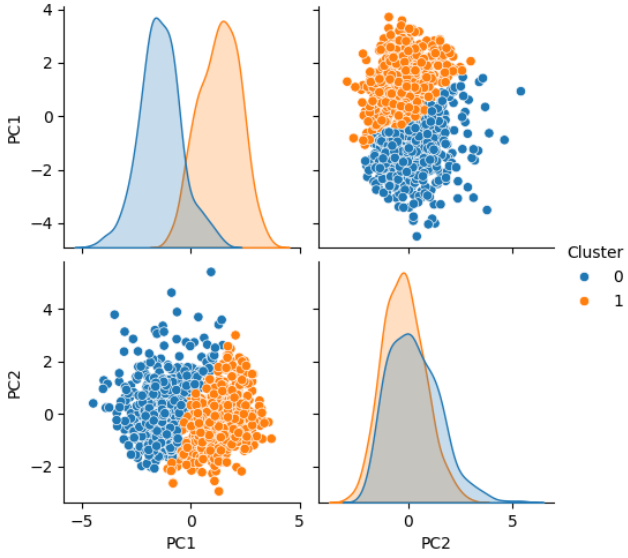


Figure 219: Clusters average cosine

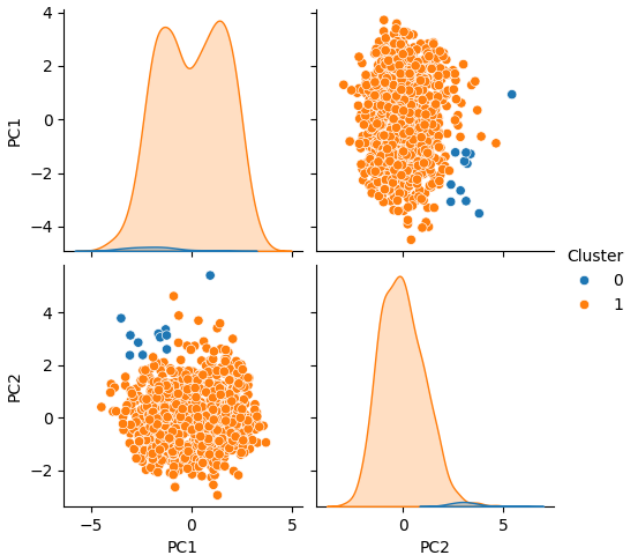


Figure 220: Clusters average euclidean

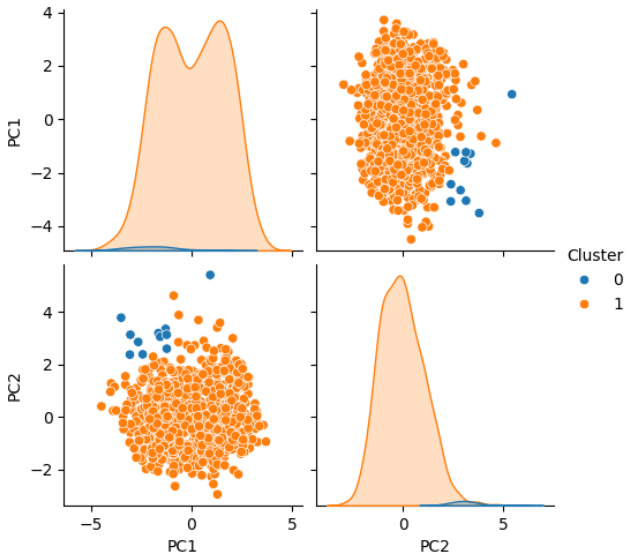


Figure 221: Clusters average manhattan

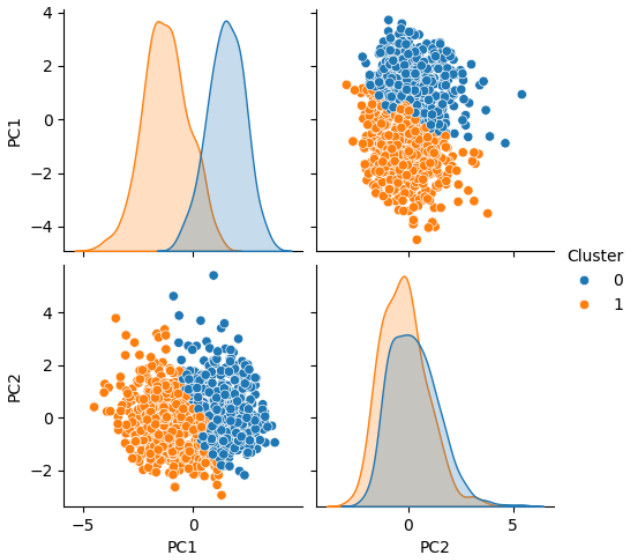


Figure 222: Clusters complete correlation

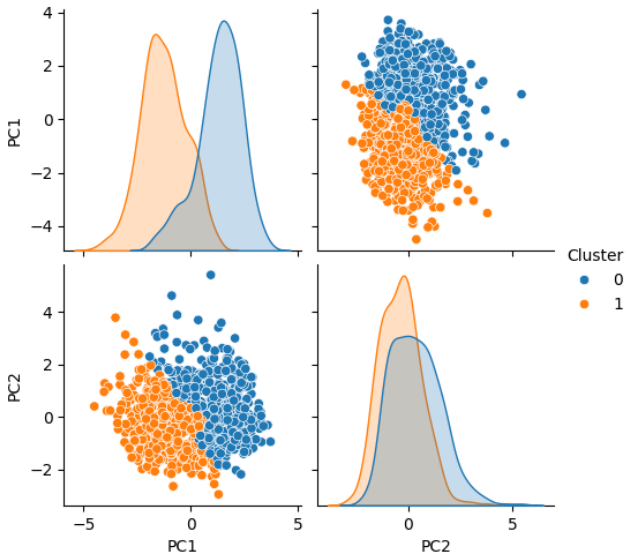


Figure 223: Clusters complete cosine

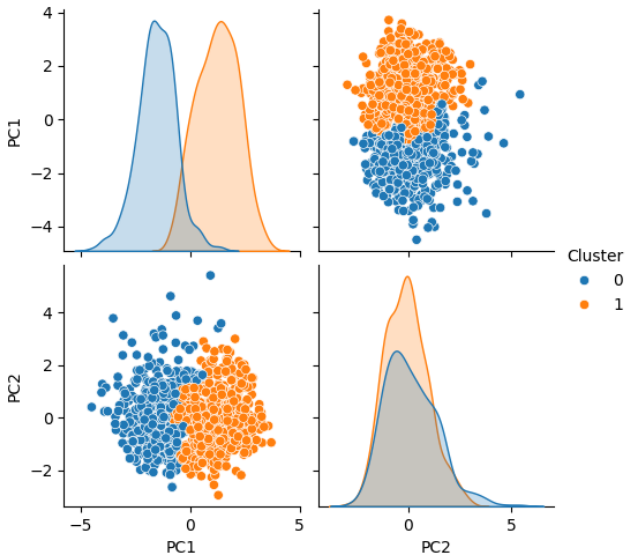


Figure 224: Clusters complete euclidean

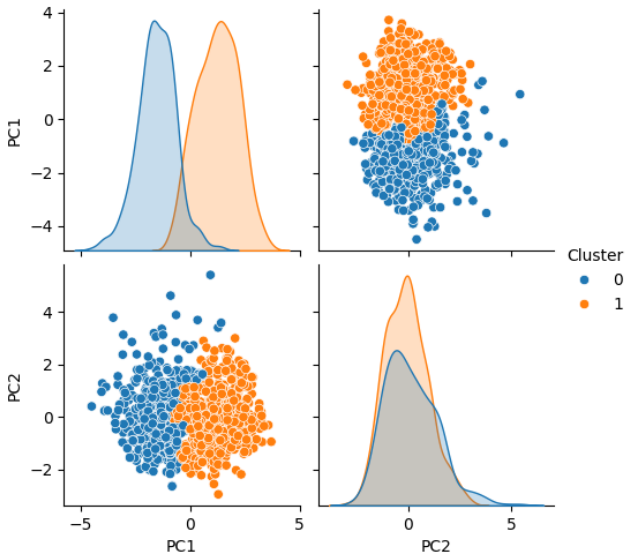


Figure 225: Clusters complete manhattan

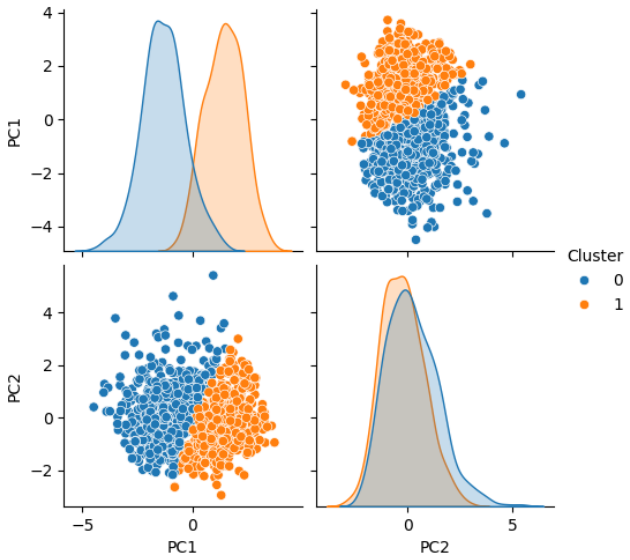


Figure 226: Clusters single correlation

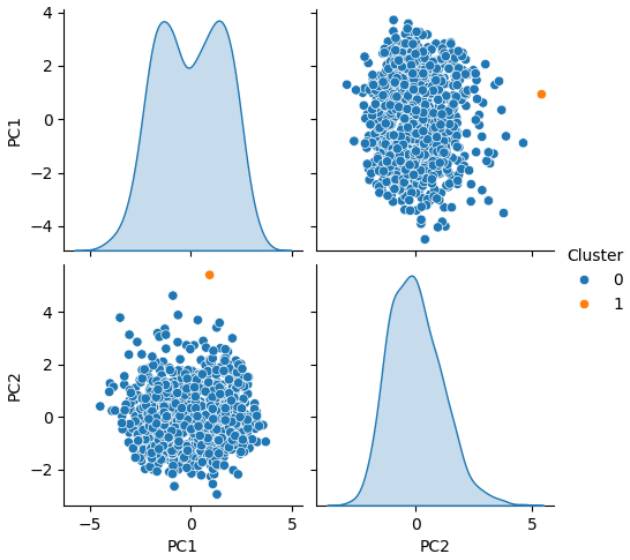


Figure 227: Clusters single cosine

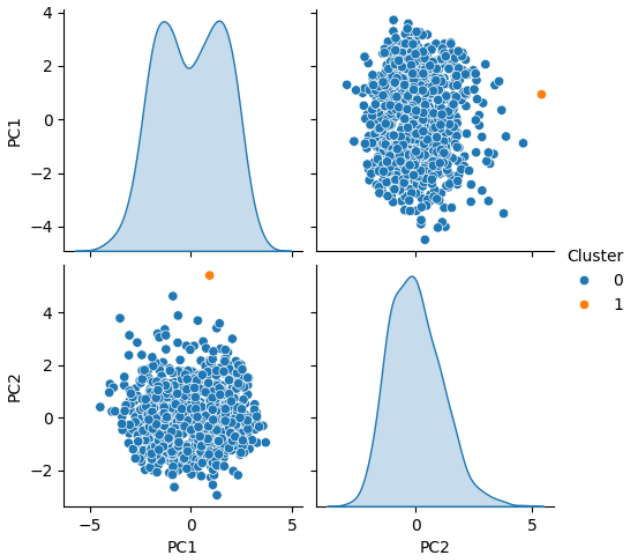


Figure 228: Clusters single euclidean

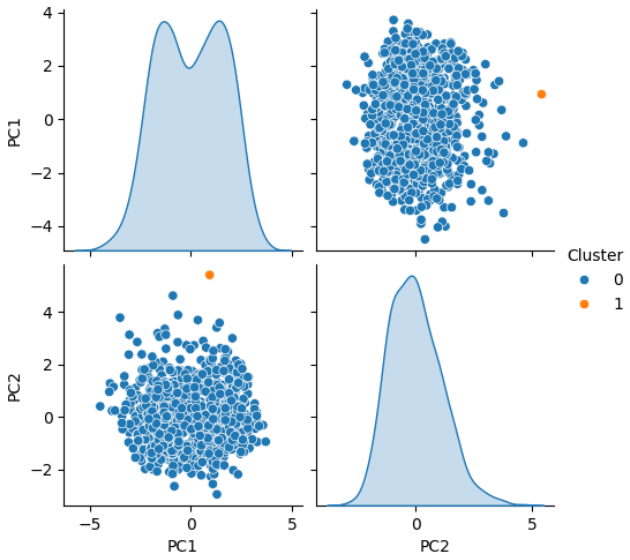


Figure 229: Clusters single manhattan

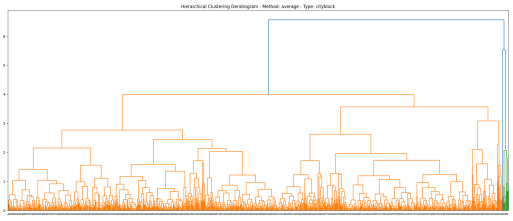


Figure 230: Dendrogram average cityblock

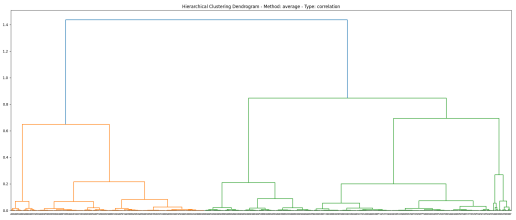


Figure 231: Dendrogram average correlation

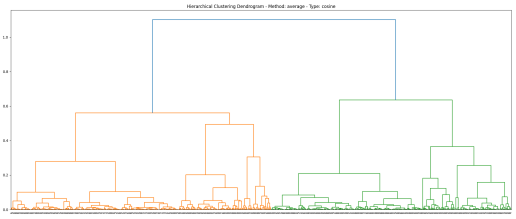


Figure 232: Dendrogram average cosine

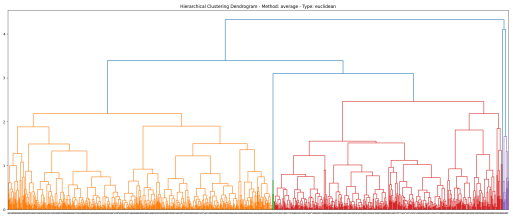


Figure 233: Dendrogram average euclidean

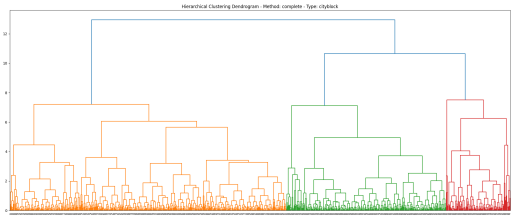


Figure 234: Dendrogram complete cityblock

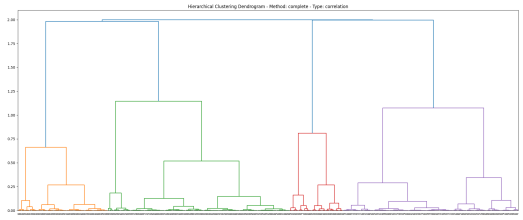


Figure 235: Dendrogram complete correlation

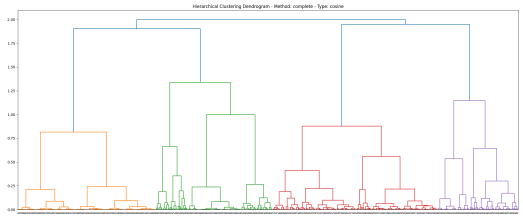


Figure 236: Dendrogram complete cosine

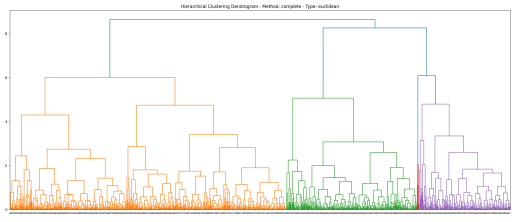


Figure 237: Dendrogram complete euclidean

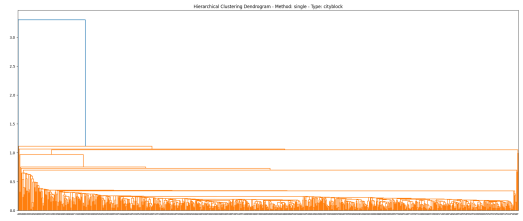


Figure 238: Dendrogram single cityblock

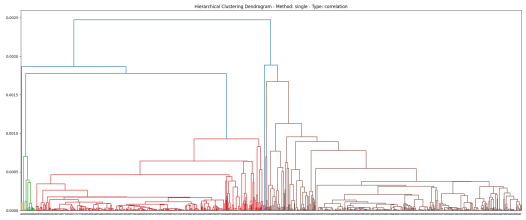


Figure 239: Dendrogram single correlation

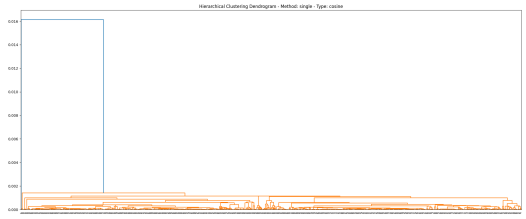


Figure 240: Dendrogram single cosine

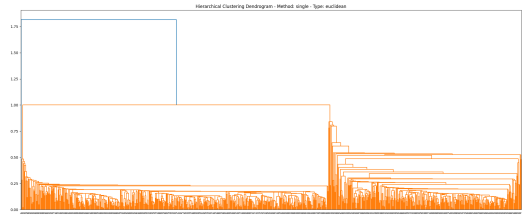


Figure 241: Dendrogram single euclidean

7.2.5 Scenario 3_N

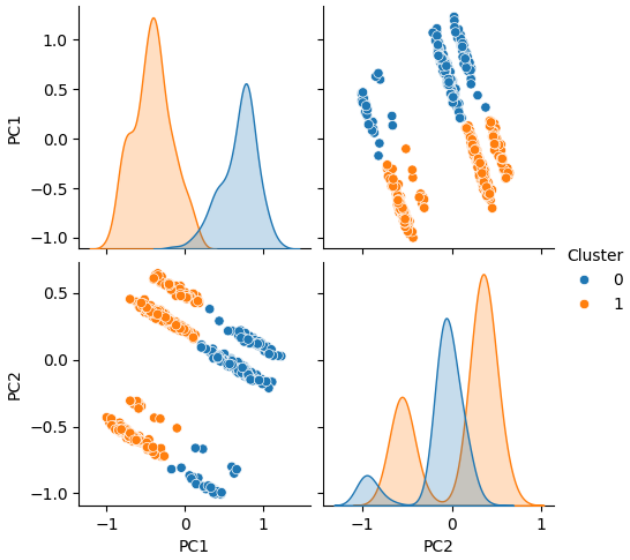


Figure 242: Clusters average correlation

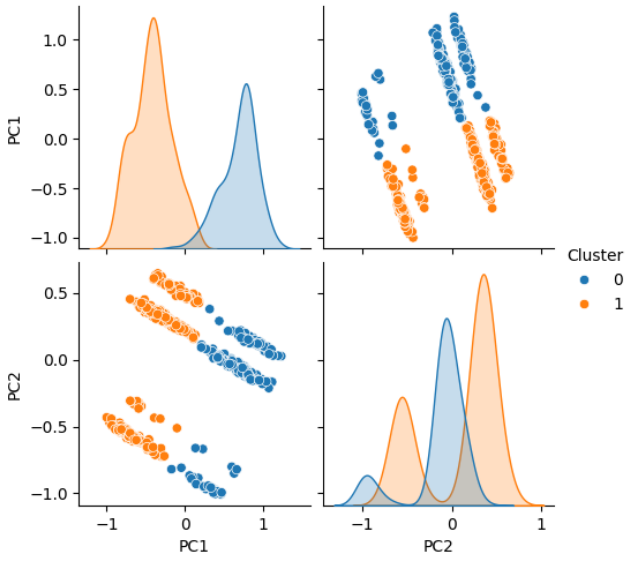


Figure 243: Clusters average cosine

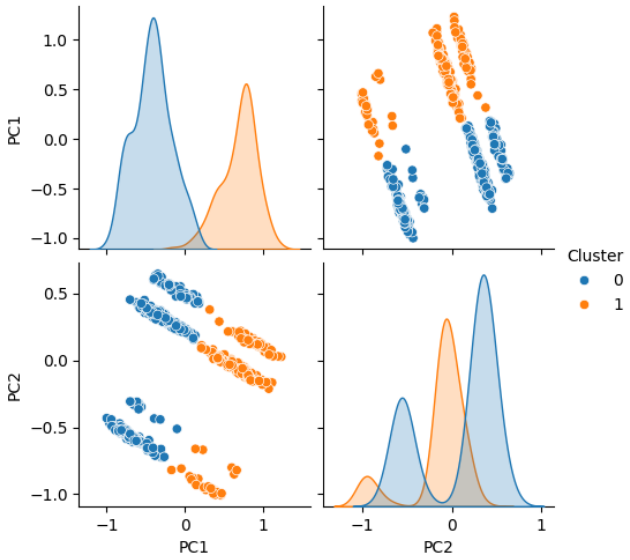


Figure 244: Clusters average euclidean

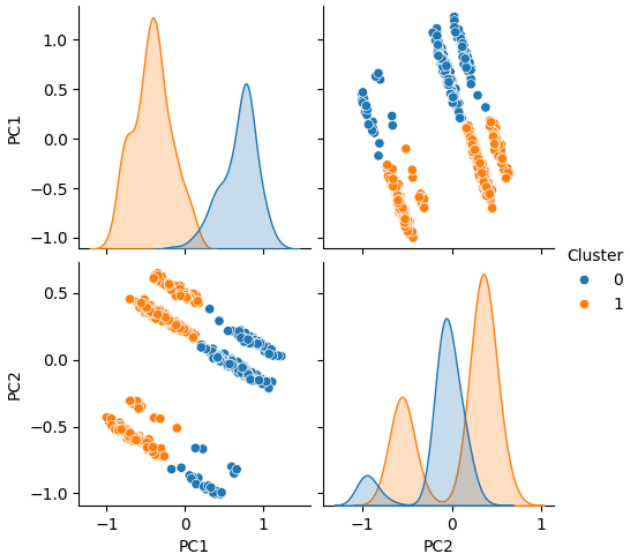


Figure 245: Clusters average manhattan

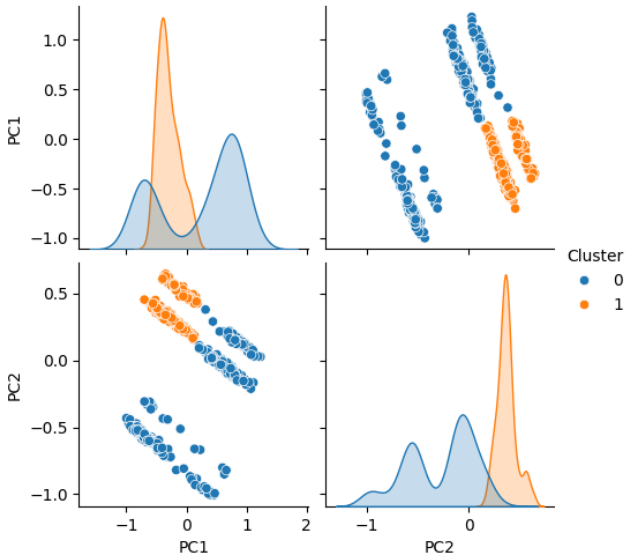


Figure 246: Clusters complete correlation

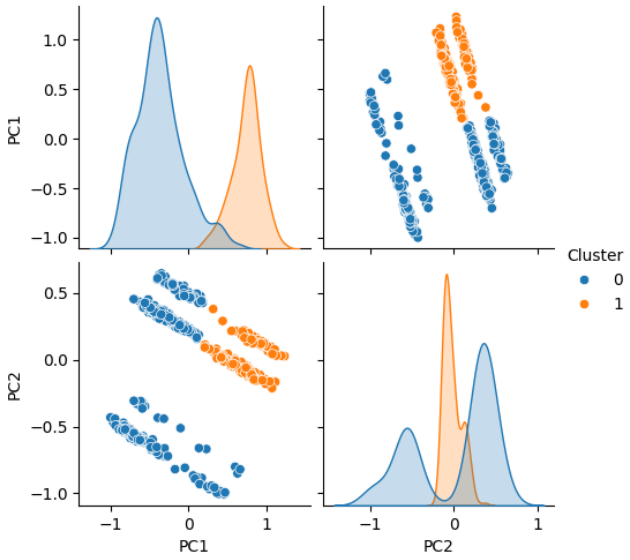


Figure 247: Clusters complete cosine

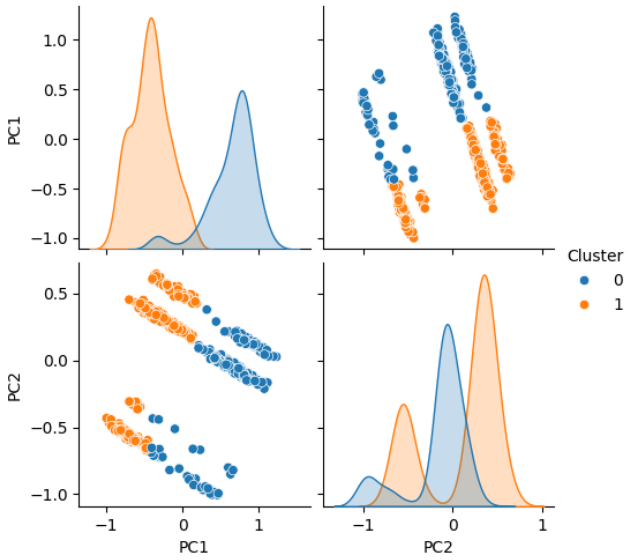


Figure 248: Clusters complete euclidean

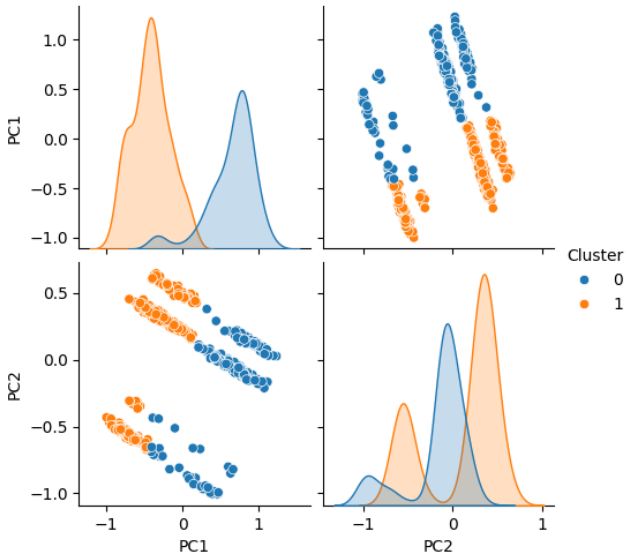


Figure 249: Clusters complete manhattan

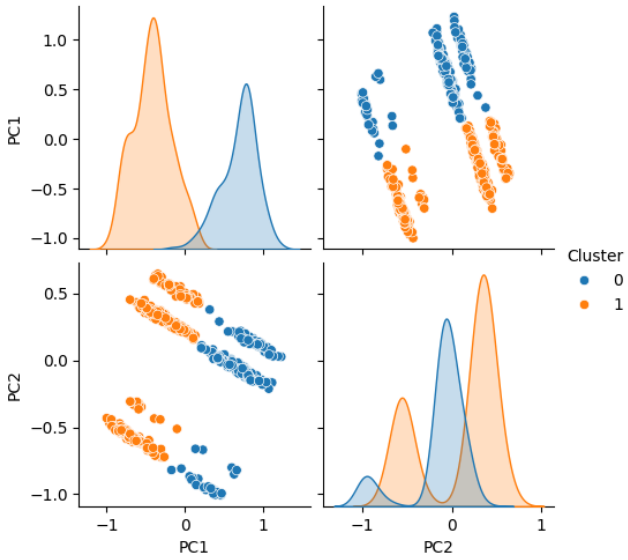


Figure 250: Clusters single correlation

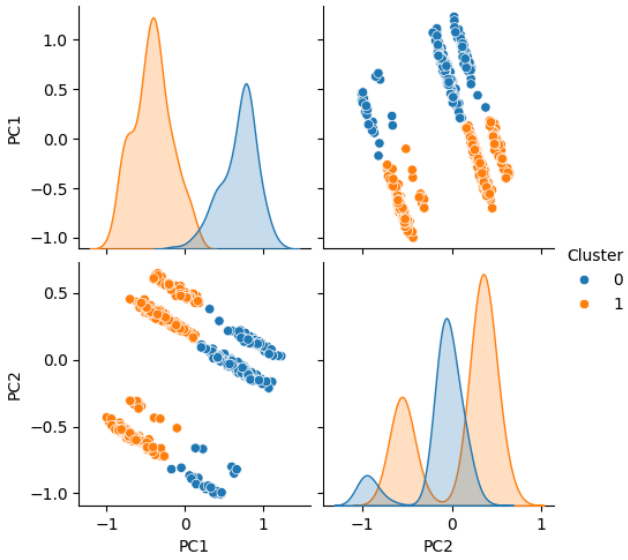


Figure 251: Clusters single cosine

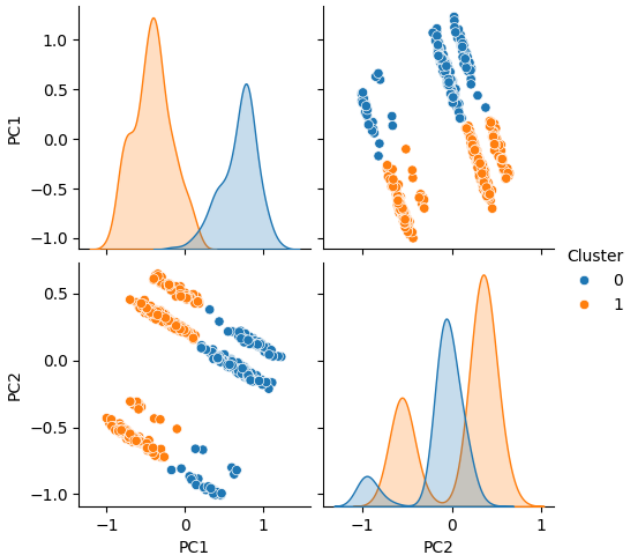


Figure 252: Clusters single euclidean

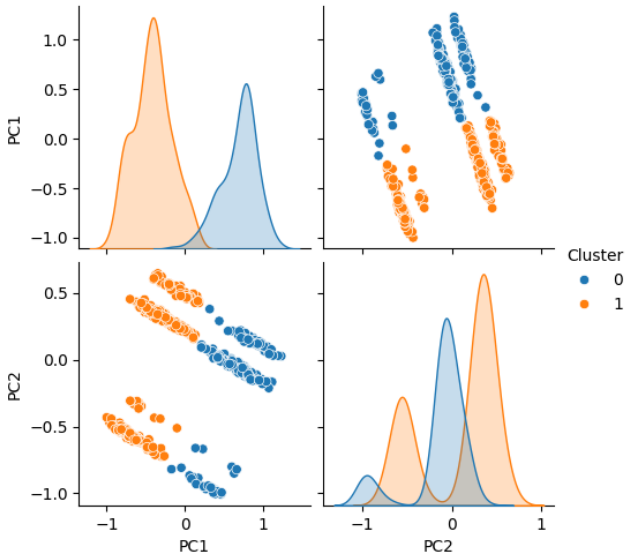


Figure 253: Clusters single manhattan

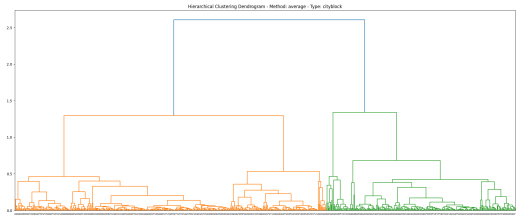


Figure 254: Dendrogram average cityblock

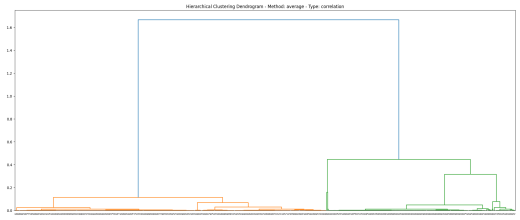


Figure 255: Dendrogram average correlation

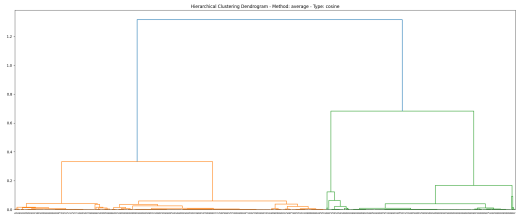


Figure 256: Dendrogram average cosine

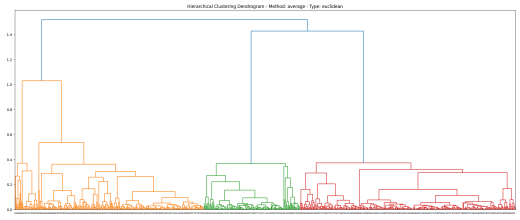


Figure 257: Dendrogram average euclidean

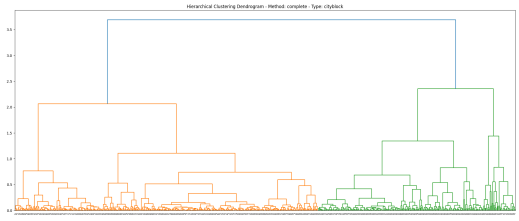


Figure 258: Dendrogram complete cityblock

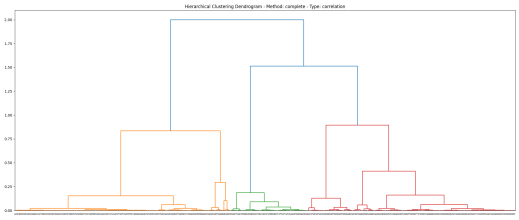


Figure 259: Dendrogram complete correlation

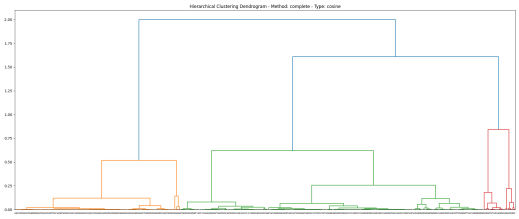


Figure 260: Dendrogram complete cosine

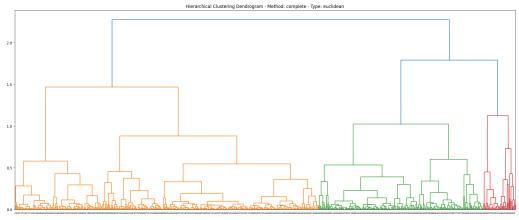


Figure 261: Dendrogram complete euclidean

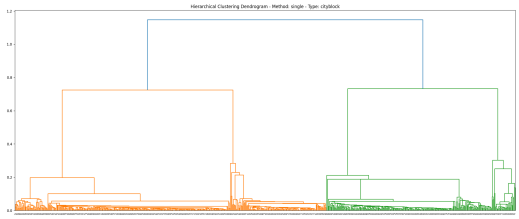


Figure 262: Dendrogram single cityblock

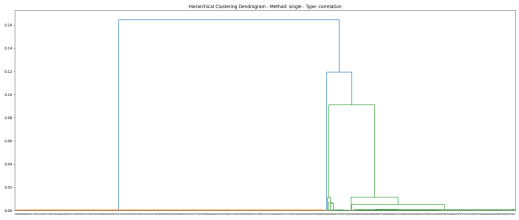


Figure 263: Dendrogram single correlation

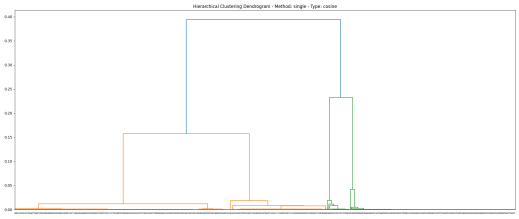


Figure 264: Dendrogram single cosine

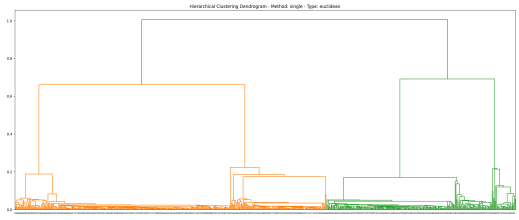


Figure 265: Dendrogram single euclidean

7.2.6 Scenario 3_S

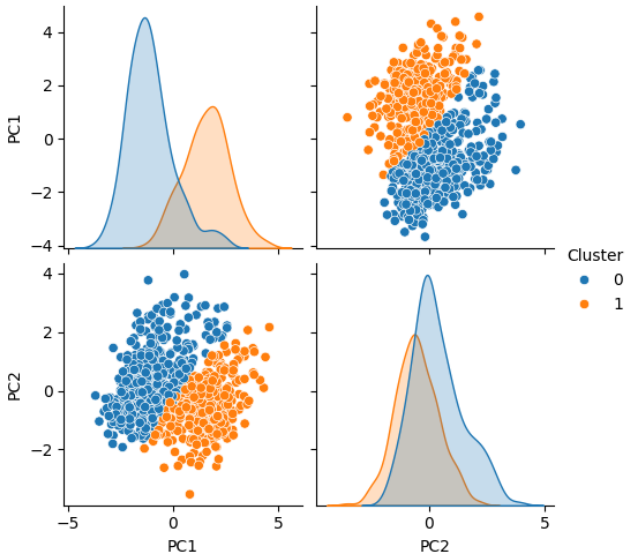


Figure 266: Clusters average correlation

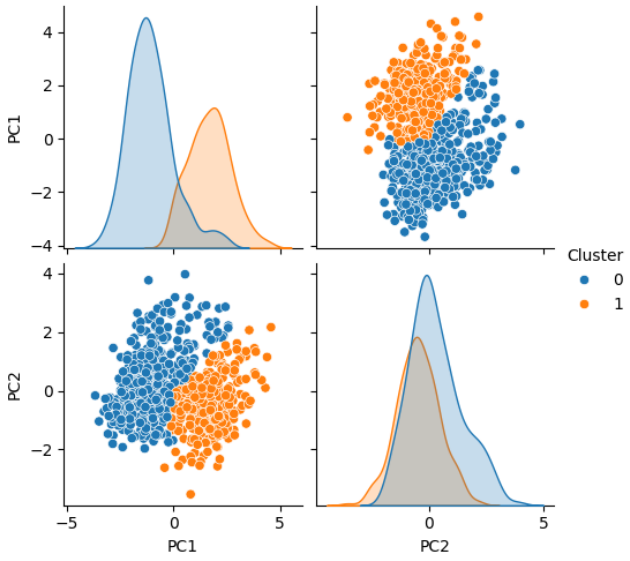


Figure 267: Clusters average cosine

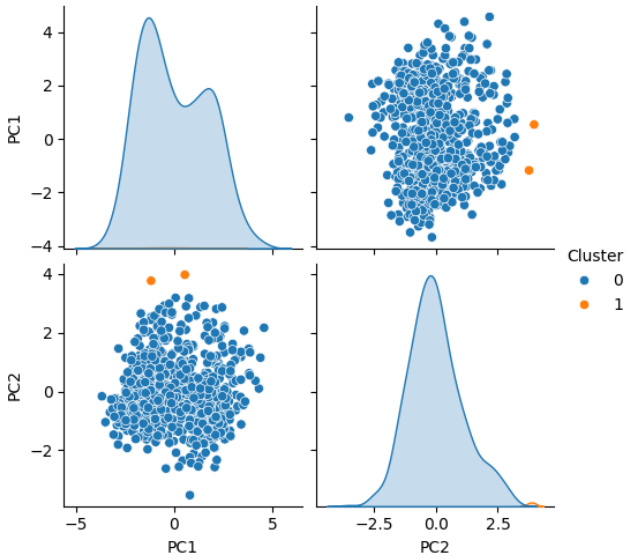


Figure 268: Clusters average euclidean

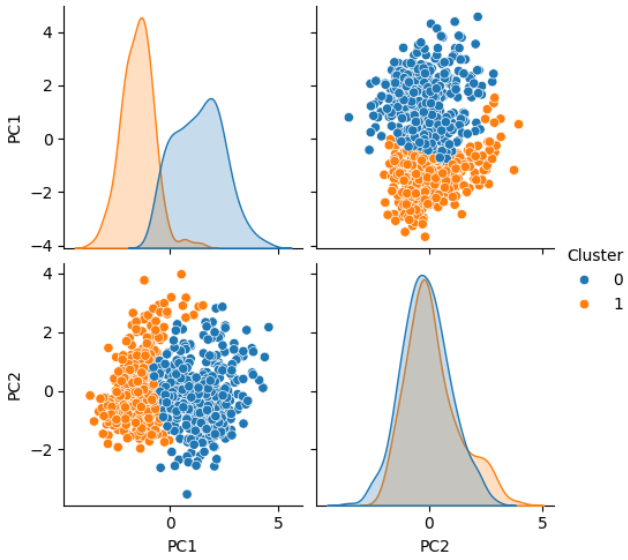


Figure 269: Clusters average manhattan

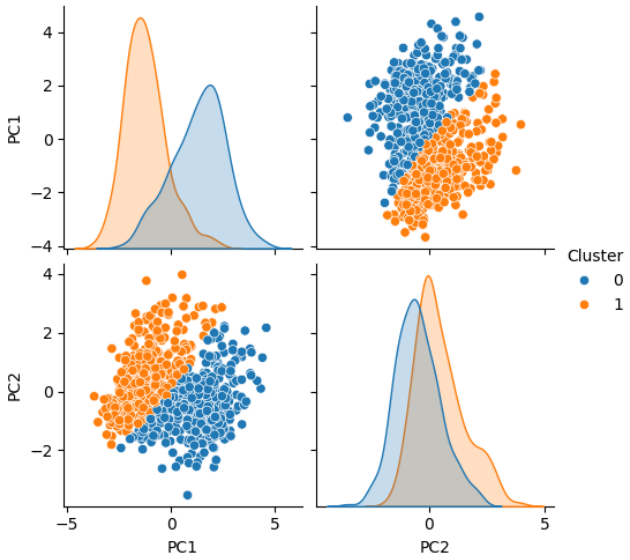


Figure 270: Clusters complete correlation

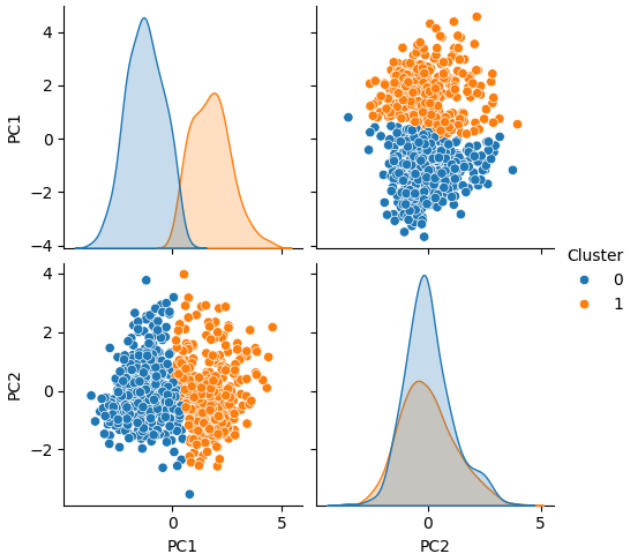


Figure 271: Clusters complete cosine

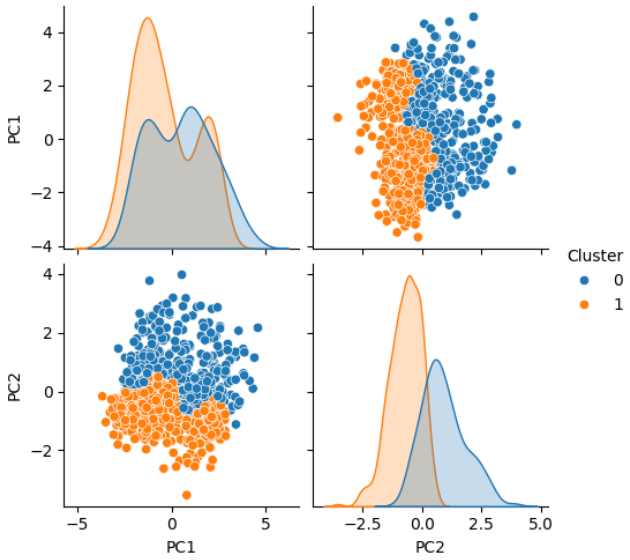


Figure 272: Clusters complete euclidean

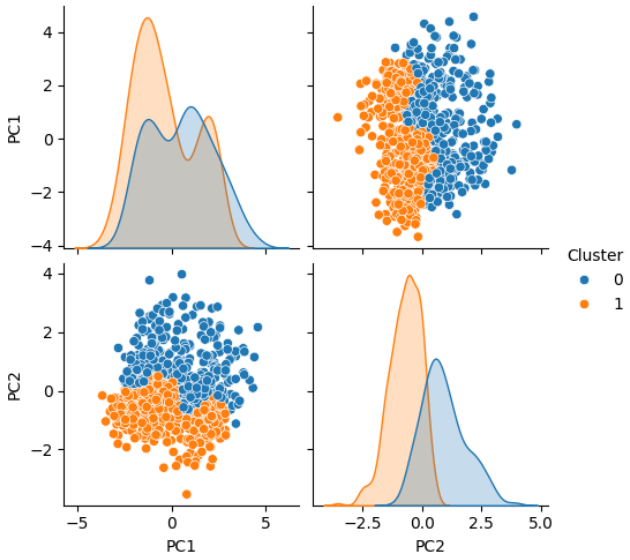


Figure 273: Clusters complete manhattan

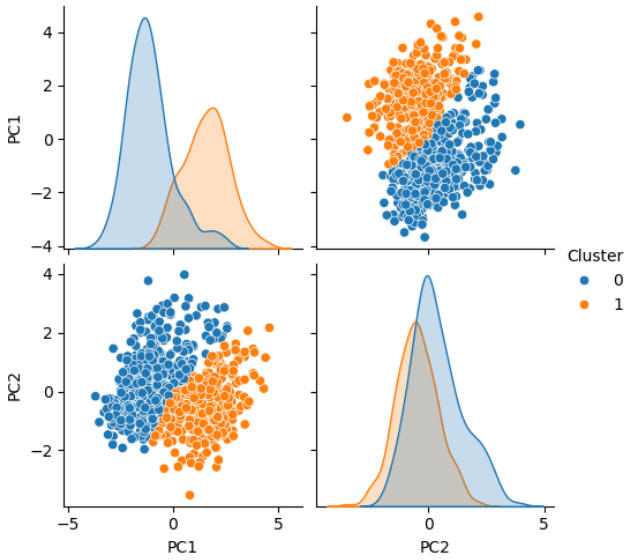


Figure 274: Clusters single correlation

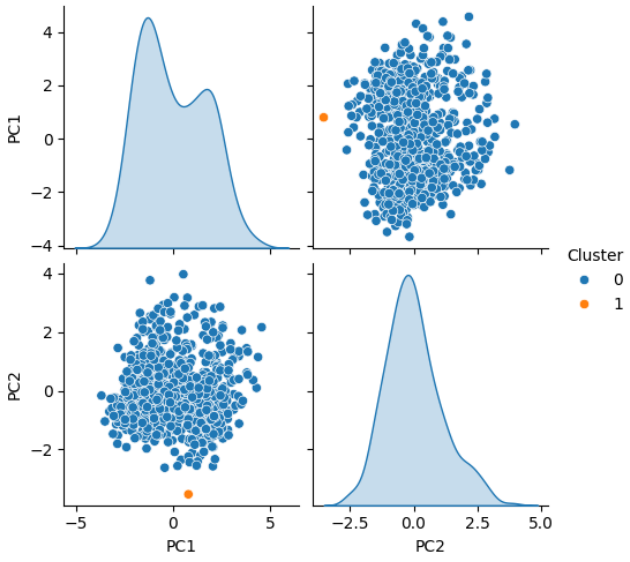


Figure 275: Clusters single cosine

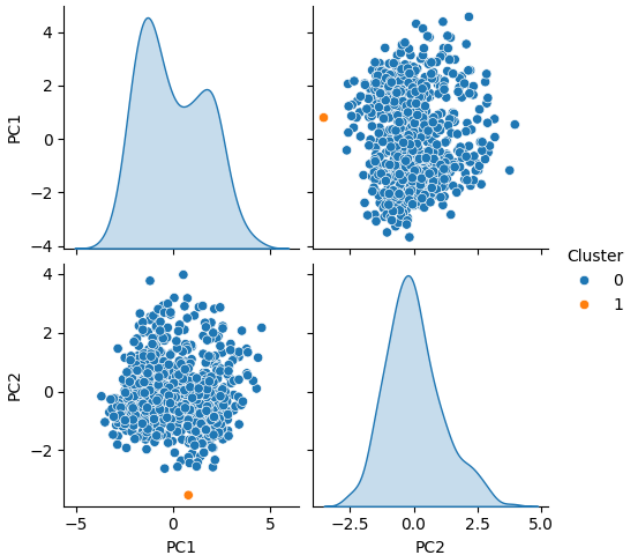


Figure 276: Clusters single euclidean

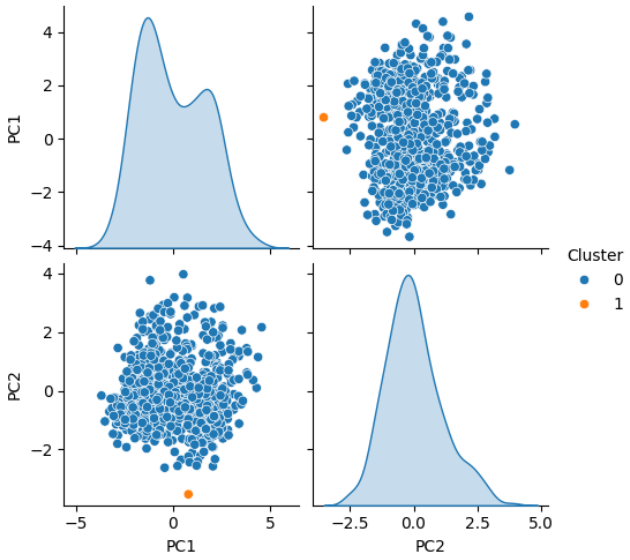


Figure 277: Clusters single manhattan

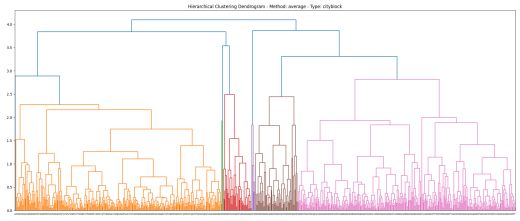


Figure 278: Dendrogram average cityblock

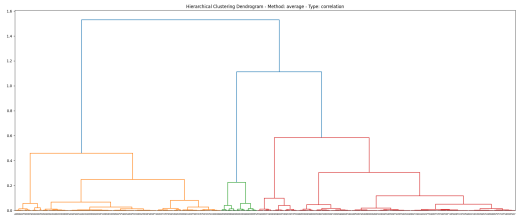


Figure 279: Dendrogram average correlation

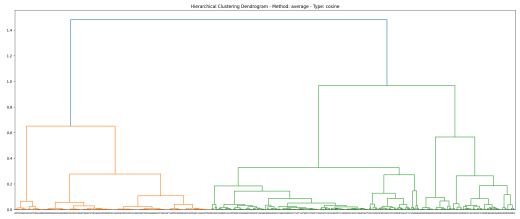


Figure 280: Dendrogram average cosine

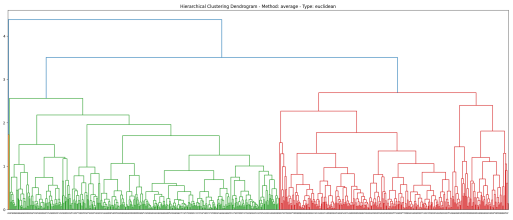


Figure 281: Dendrogram average euclidean

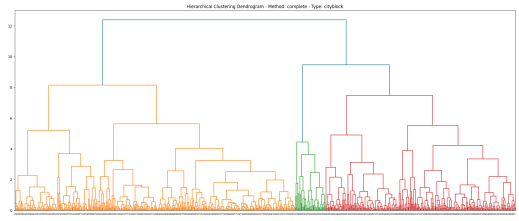


Figure 282: Dendrogram complete cityblock

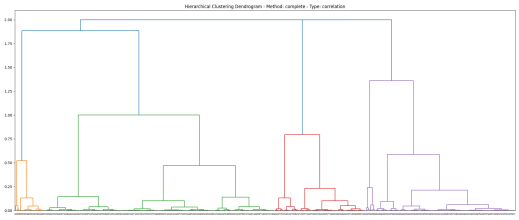


Figure 283: Dendrogram complete correlation

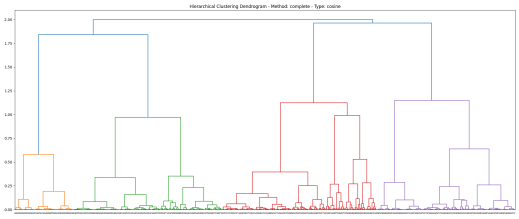


Figure 284: Dendrogram complete cosine

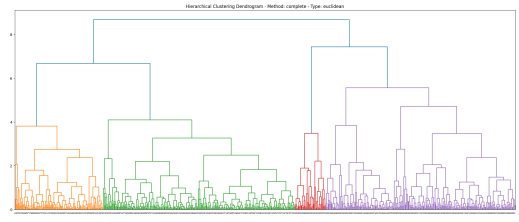


Figure 285: Dendrogram complete euclidean

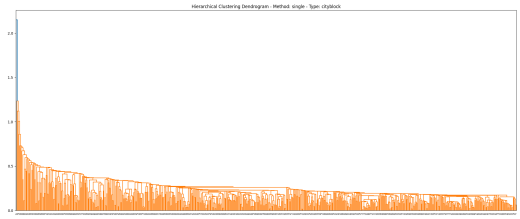


Figure 286: Dendrogram single cityblock

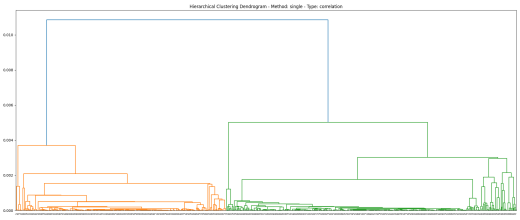


Figure 287: Dendrogram single correlation

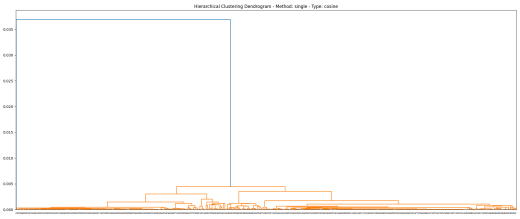


Figure 288: Dendrogram single cosine

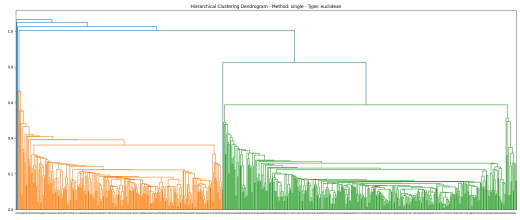


Figure 289: Dendrogram single euclidean

7.2.7 Scenario 4_N

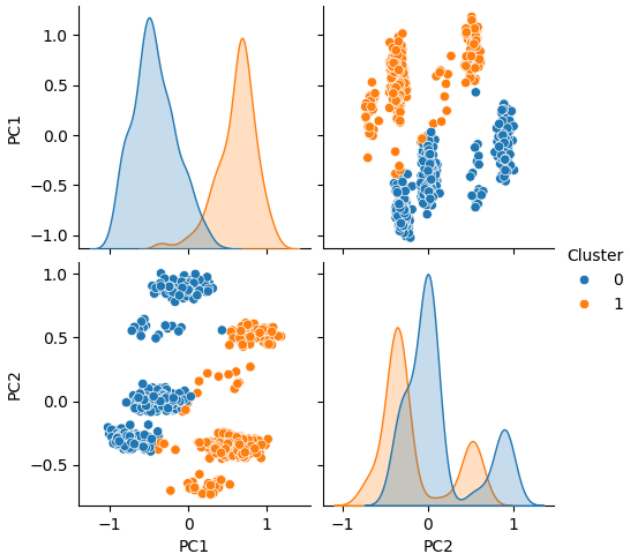


Figure 290: Clusters average correlation

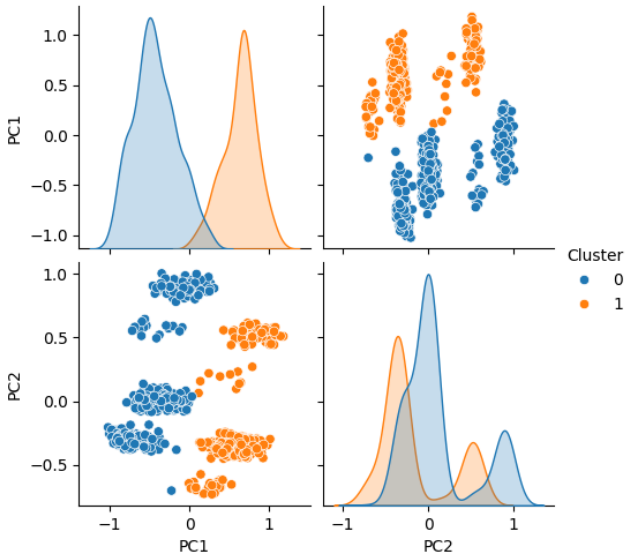


Figure 291: Clusters average cosine

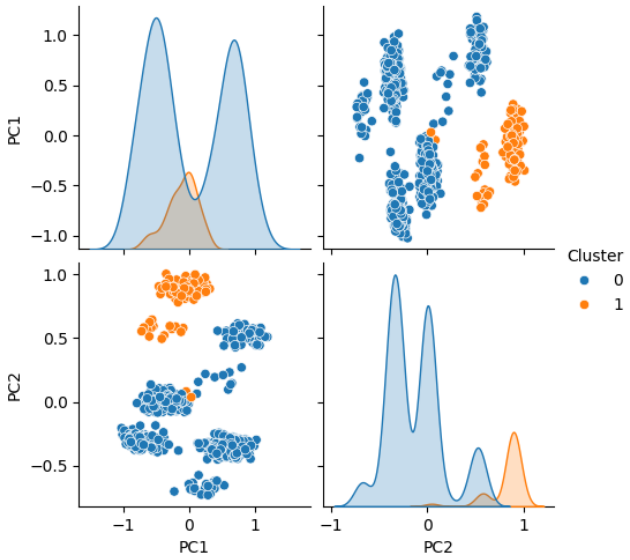


Figure 292: Clusters average euclidean

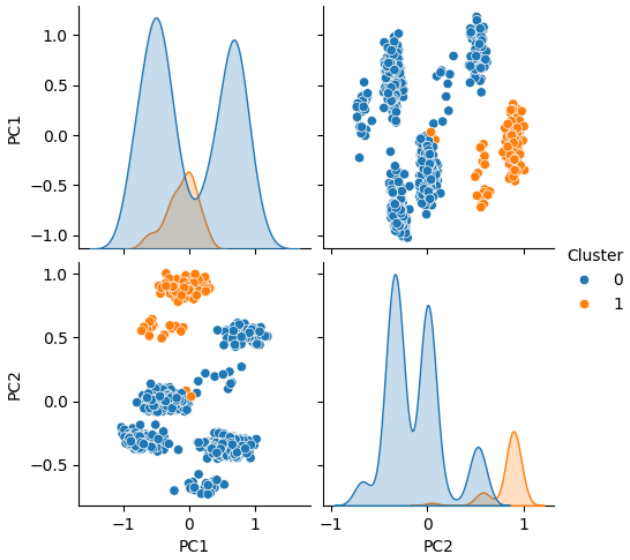


Figure 293: Clusters average manhattan

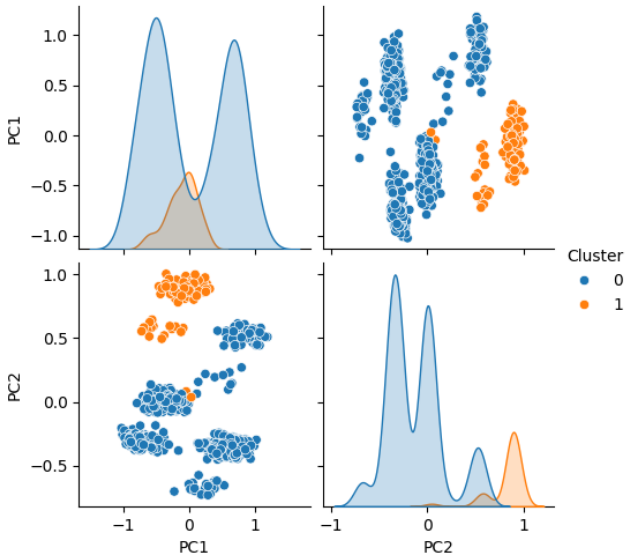


Figure 294: Clusters complete correlation

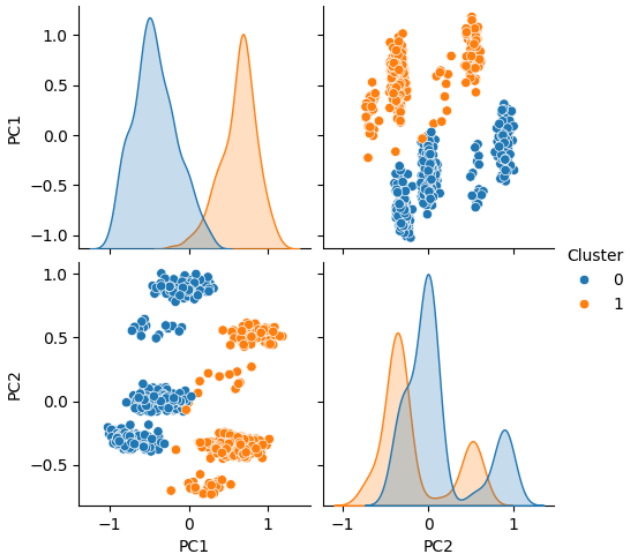


Figure 295: Clusters complete cosine

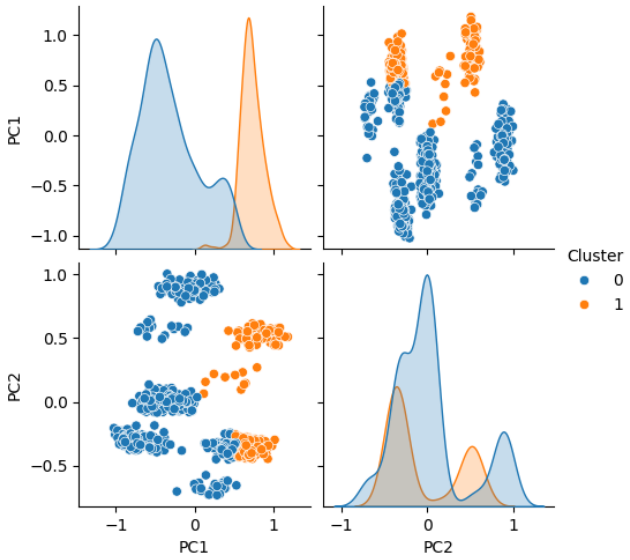


Figure 296: Clusters complete euclidean

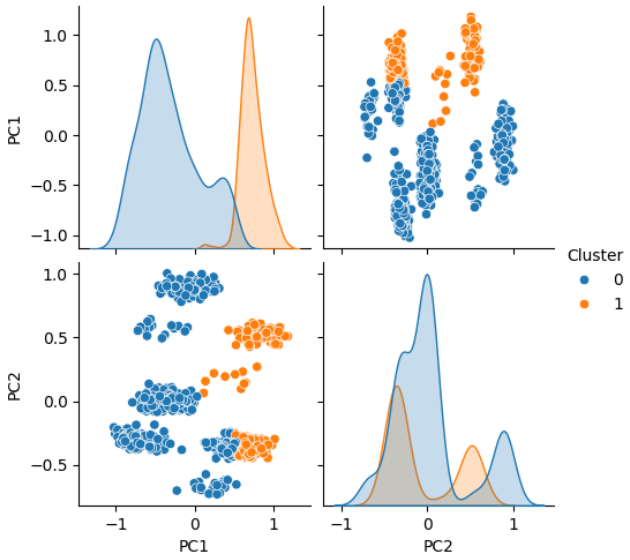


Figure 297: Clusters complete manhattan

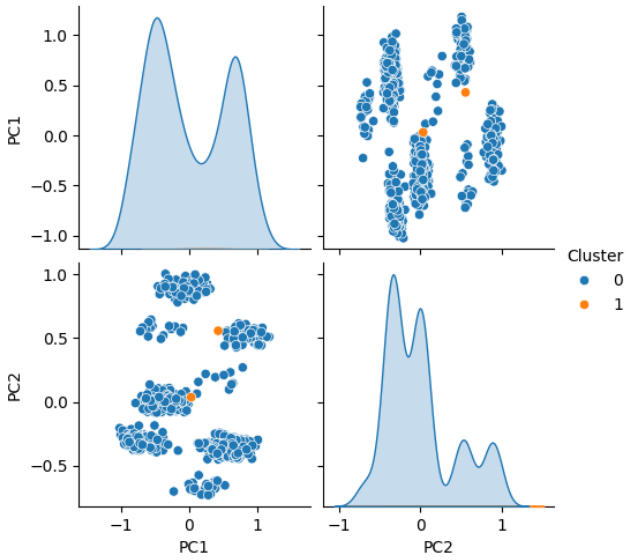


Figure 298: Clusters single correlation

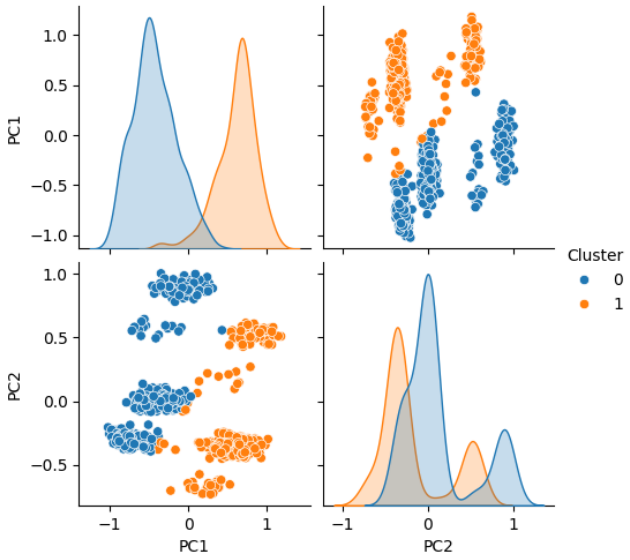


Figure 299: Clusters single cosine

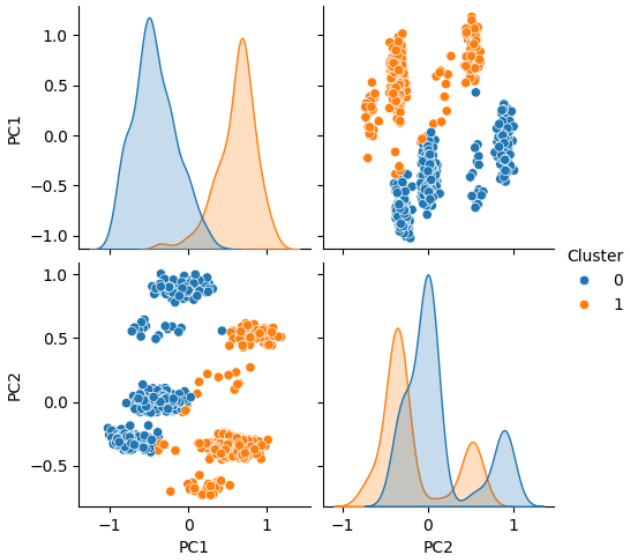


Figure 300: Clusters single euclidean

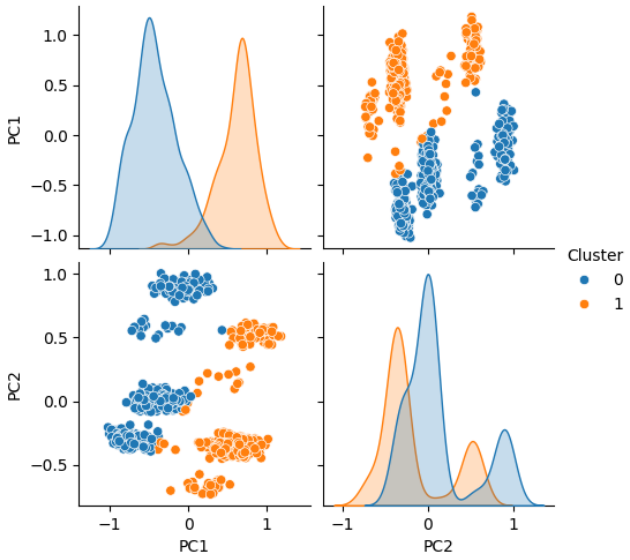


Figure 301: Clusters single manhattan

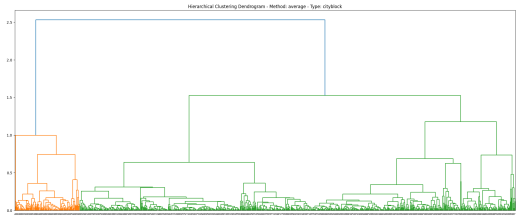


Figure 302: Dendrogram average cityblock

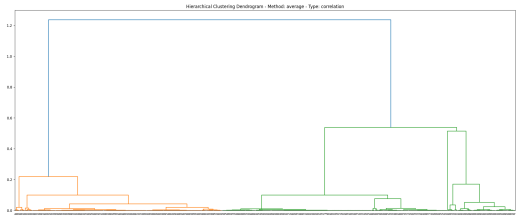


Figure 303: Dendrogram average correlation

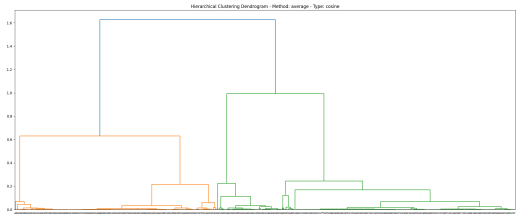


Figure 304: Dendrogram average cosine

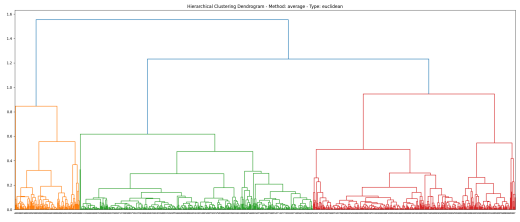


Figure 305: Dendrogram average euclidean

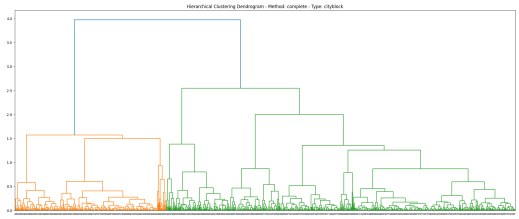


Figure 306: Dendrogram complete cityblock

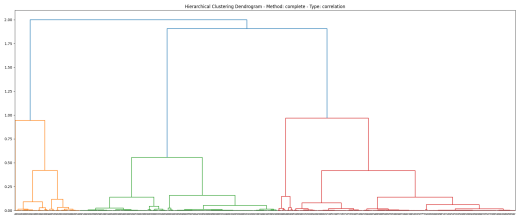


Figure 307: Dendrogram complete correlation

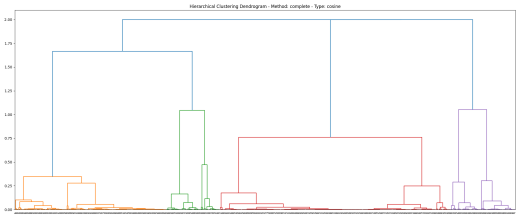


Figure 308: Dendrogram complete cosine

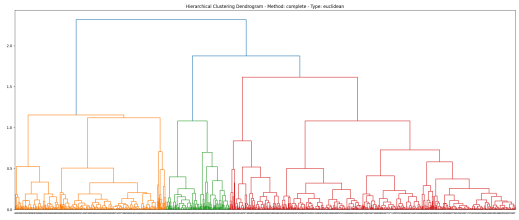


Figure 309: Dendrogram complete euclidean

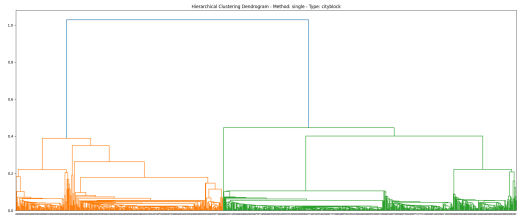


Figure 310: Dendrogram single cityblock

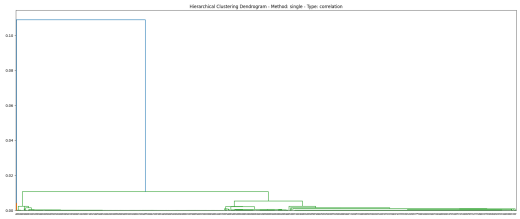


Figure 311: Dendrogram single correlation

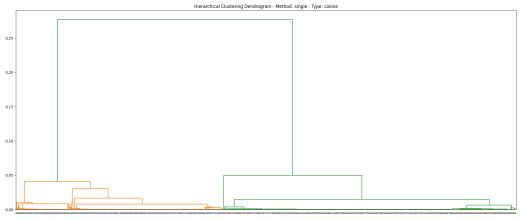


Figure 312: Dendrogram single cosine

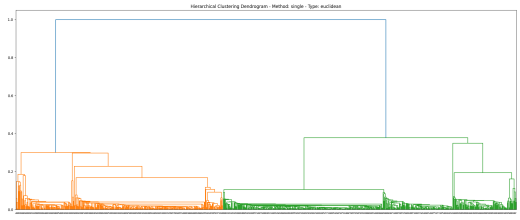


Figure 313: Dendrogram single euclidean

7.2.8 Scenario 4_S

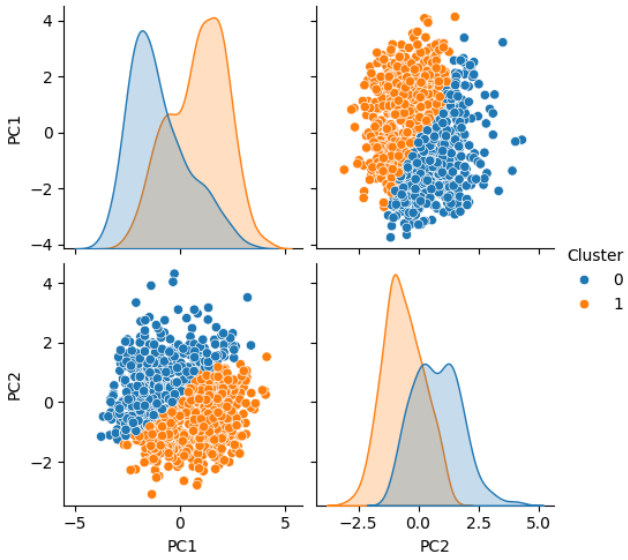


Figure 314: Clusters average correlation

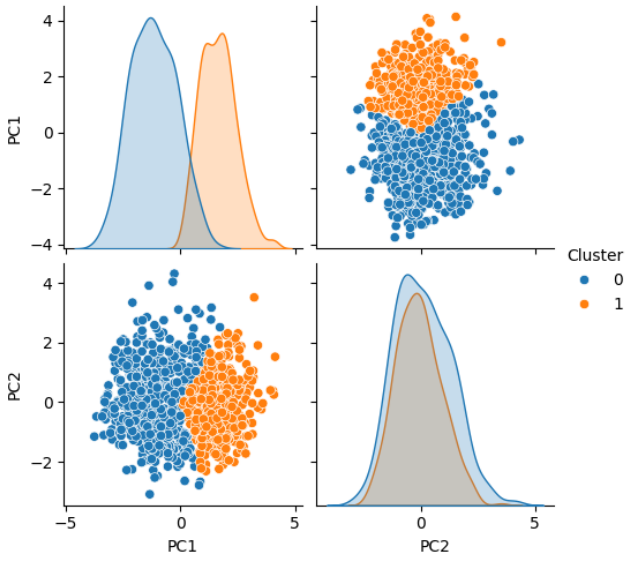


Figure 315: Clusters average cosine

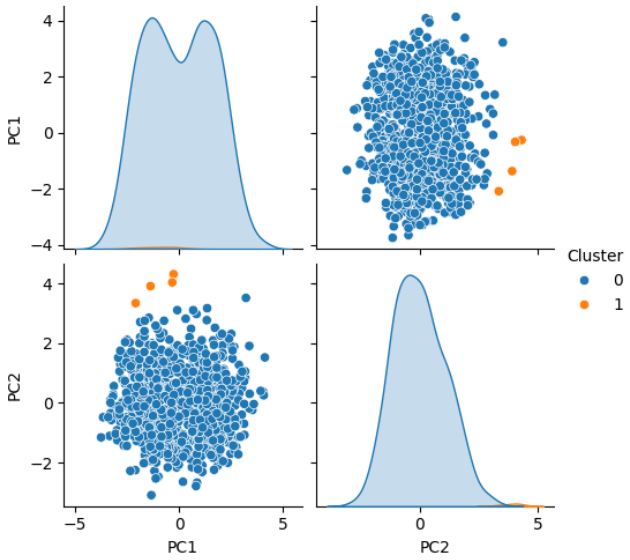


Figure 316: Clusters average euclidean

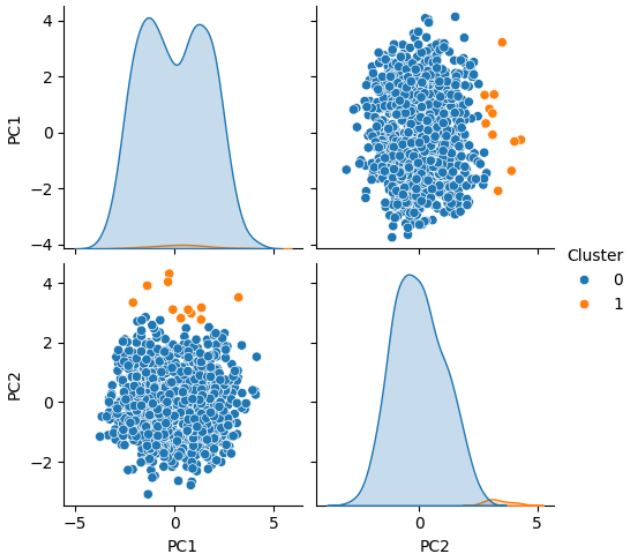


Figure 317: Clusters average manhattan

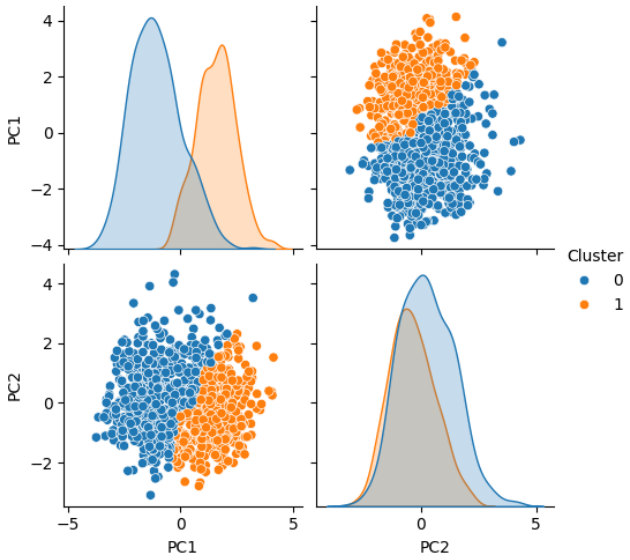


Figure 318: Clusters complete correlation

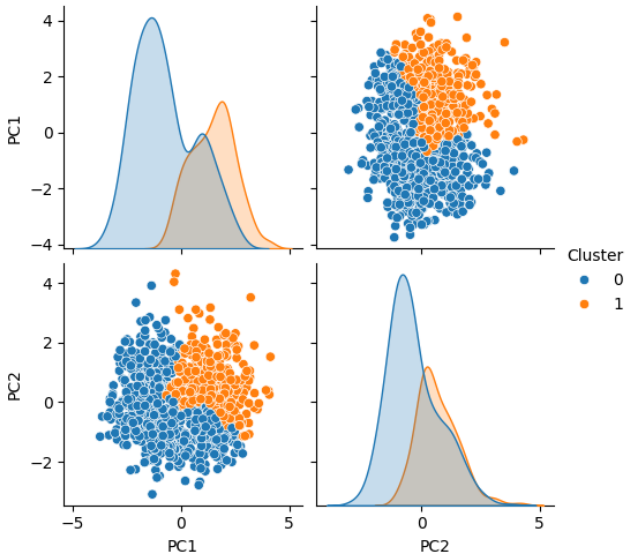


Figure 319: Clusters complete cosine

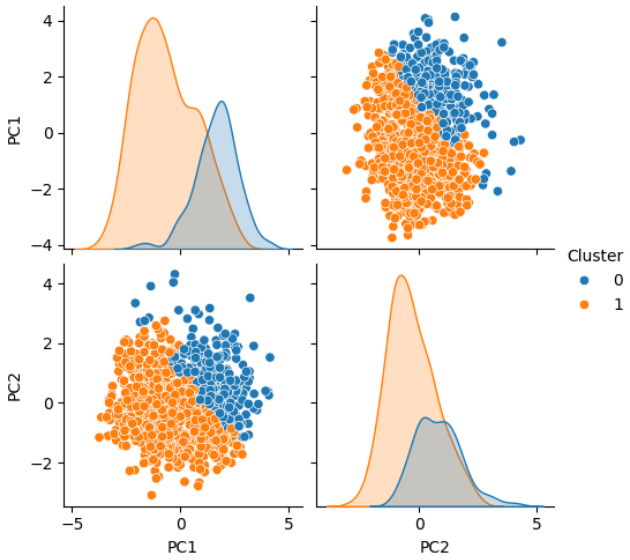


Figure 320: Clusters complete euclidean

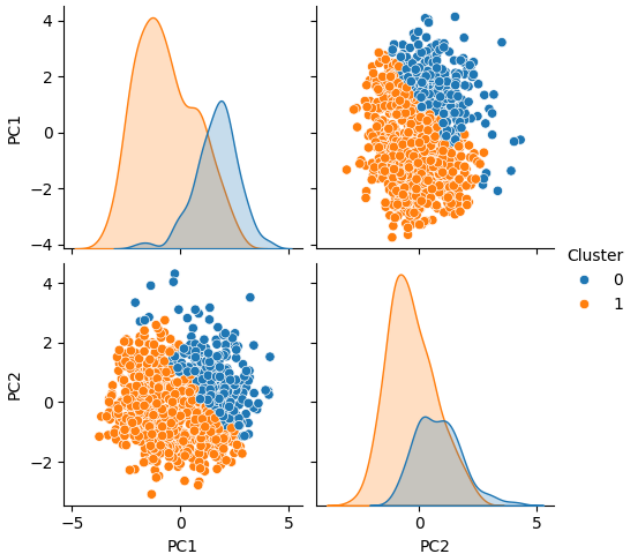


Figure 321: Clusters complete manhattan

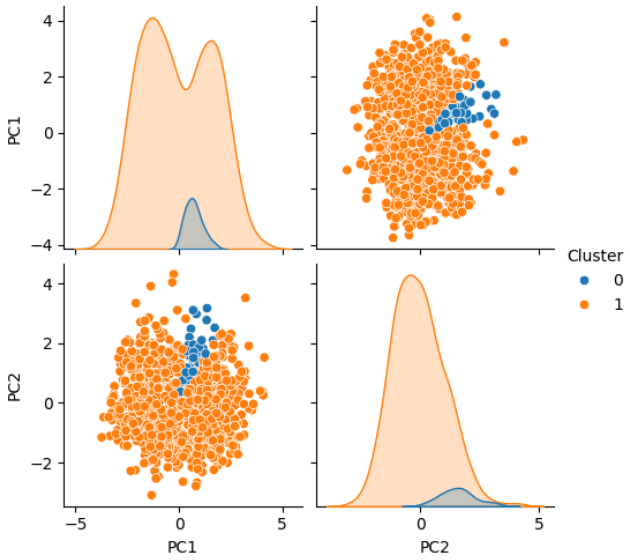


Figure 322: Clusters single correlation

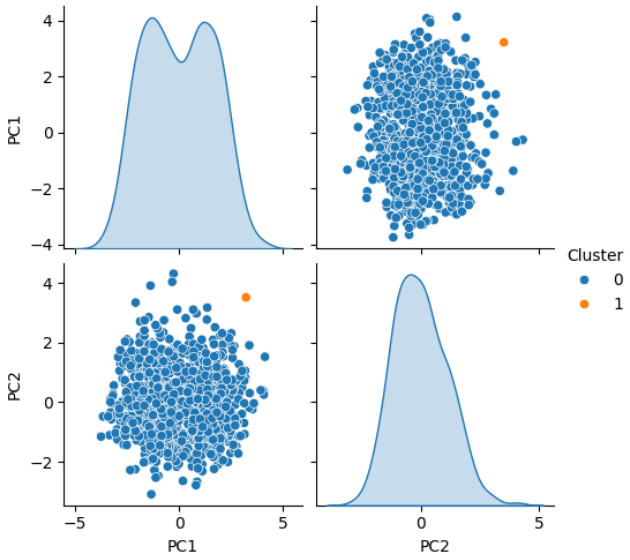


Figure 323: Clusters single cosine

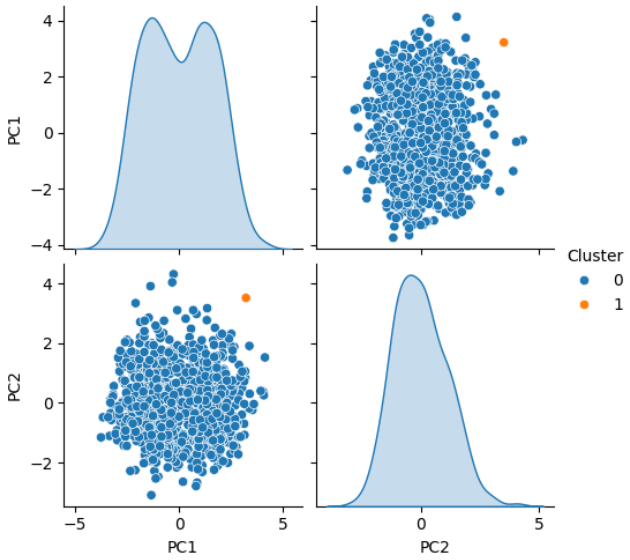


Figure 324: Clusters single euclidean

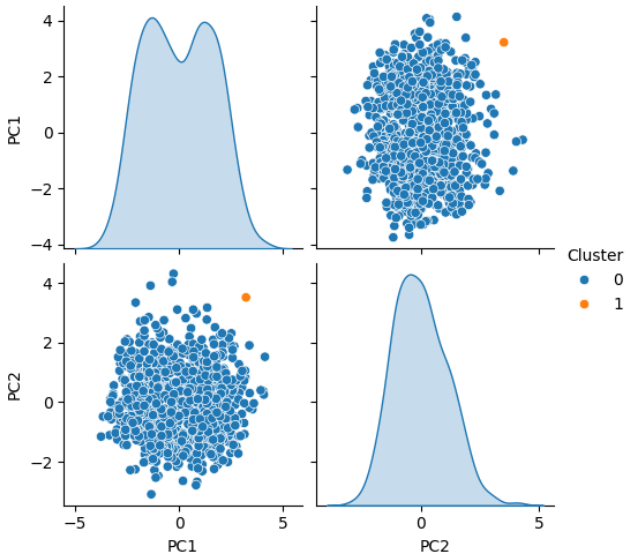


Figure 325: Clusters single manhattan

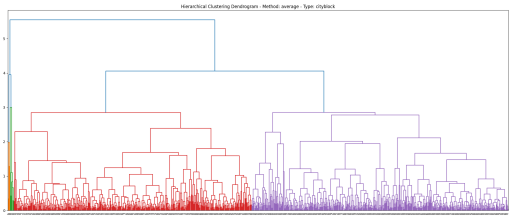


Figure 326: Dendrogram average cityblock

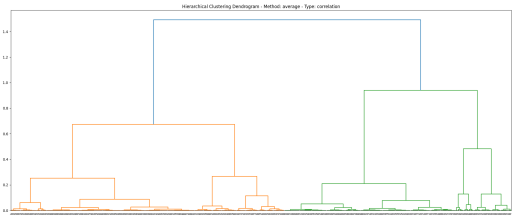


Figure 327: Dendrogram average correlation

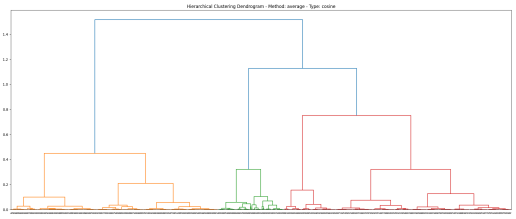


Figure 328: Dendrogram average cosine

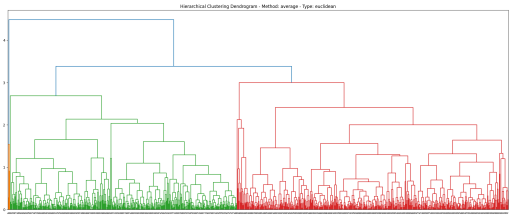


Figure 329: Dendrogram average euclidean

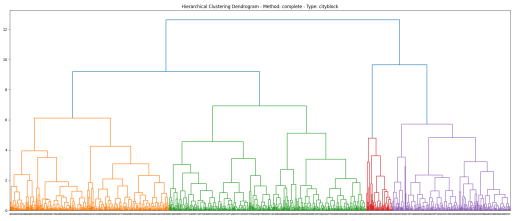


Figure 330: Dendrogram complete cityblock

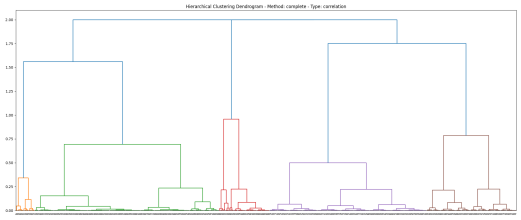


Figure 331: Dendrogram complete correlation

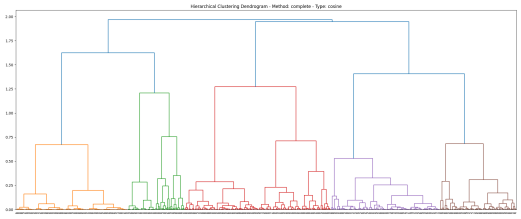


Figure 332: Dendrogram complete cosine

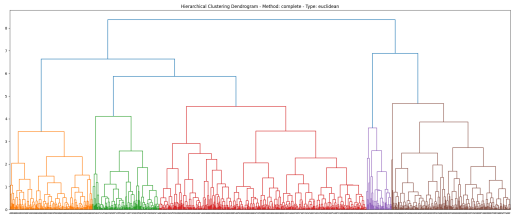


Figure 333: Dendrogram complete euclidean

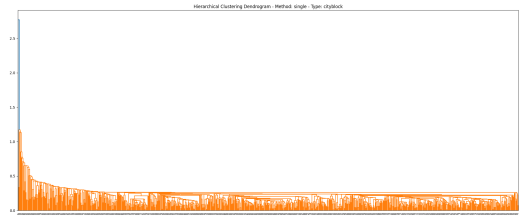


Figure 334: Dendrogram single cityblock

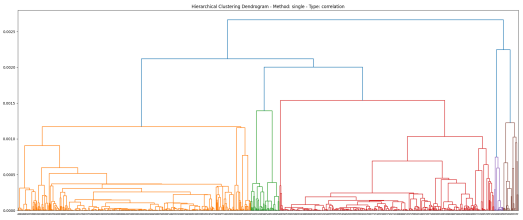


Figure 335: Dendrogram single correlation

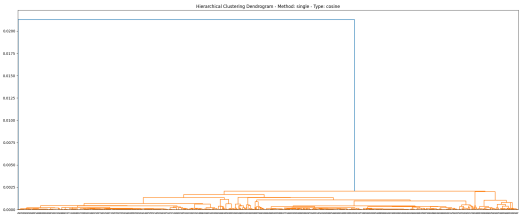


Figure 336: Dendrogram single cosine

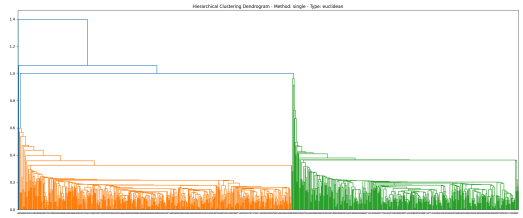


Figure 337: Dendrogram single euclidean

7.2.9 Scenario 5_N

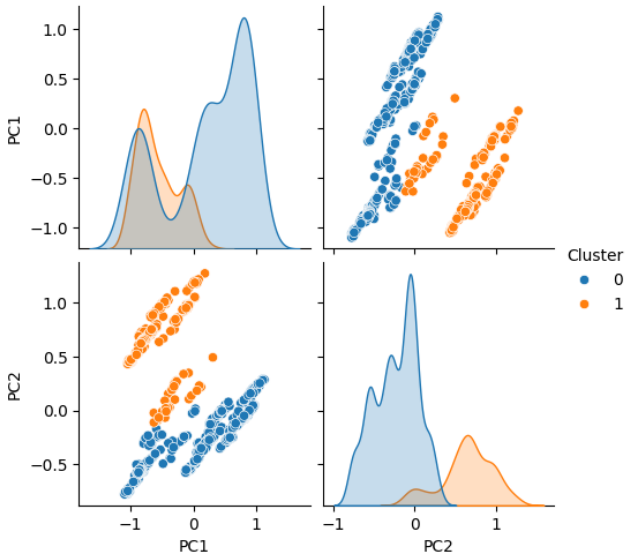


Figure 338: Clusters average correlation

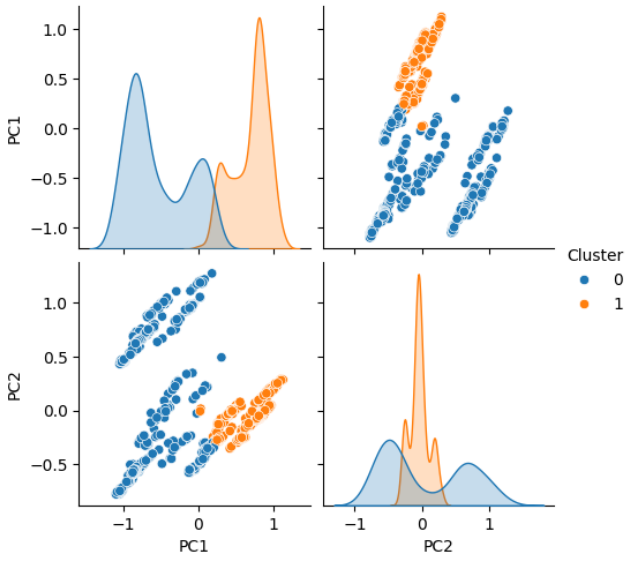


Figure 339: Clusters average cosine

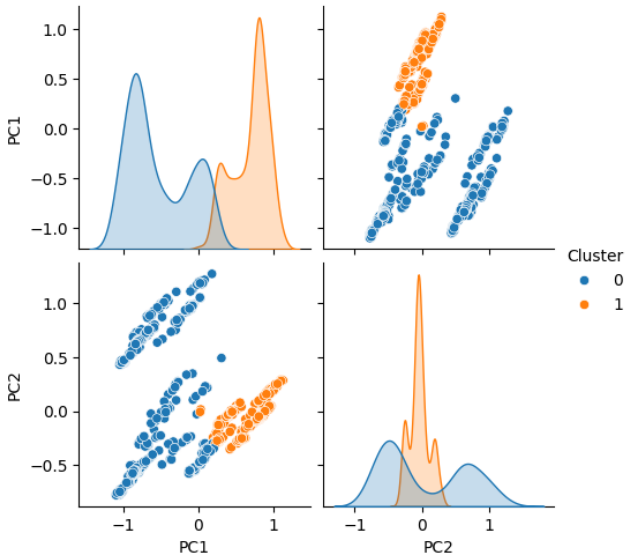


Figure 340: Clusters average euclidean

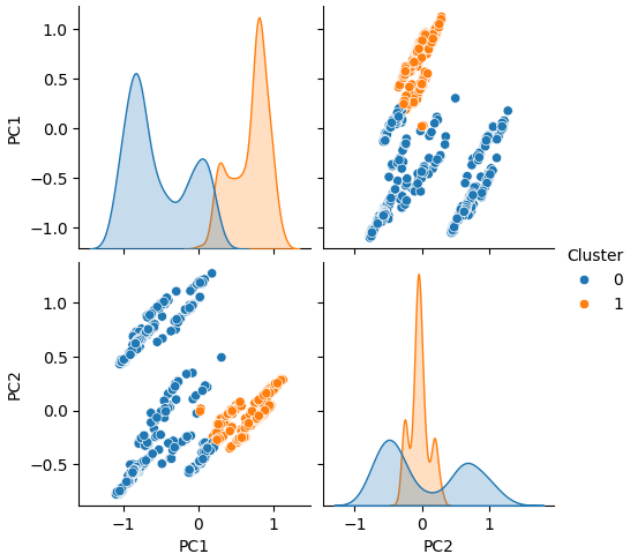


Figure 341: Clusters average manhattan

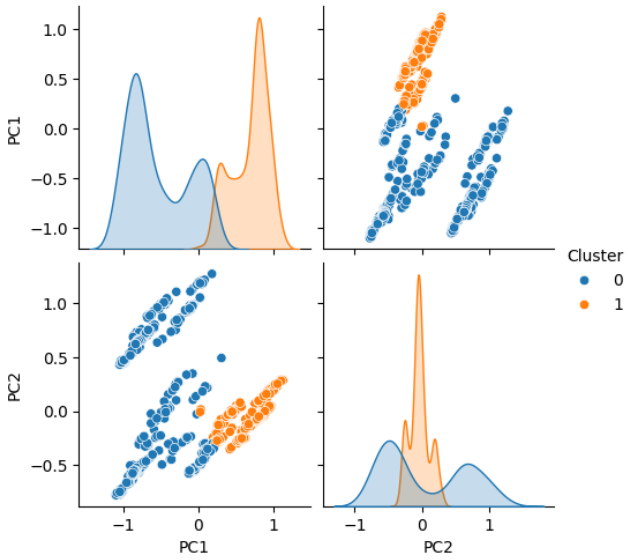


Figure 342: Clusters complete correlation

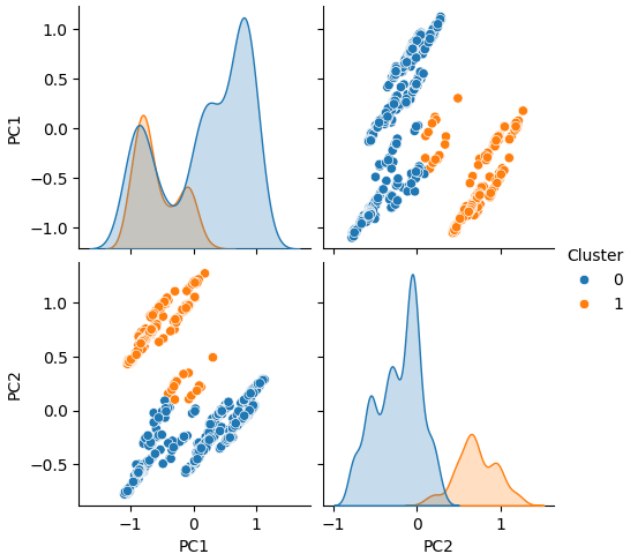


Figure 343: Clusters complete cosine

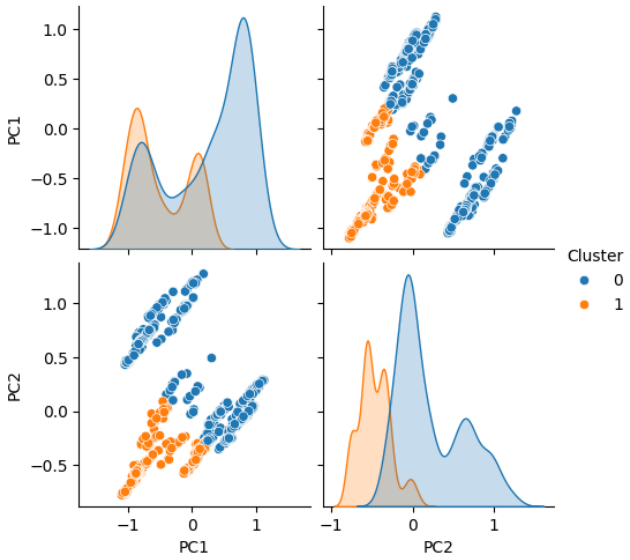


Figure 344: Clusters complete euclidean

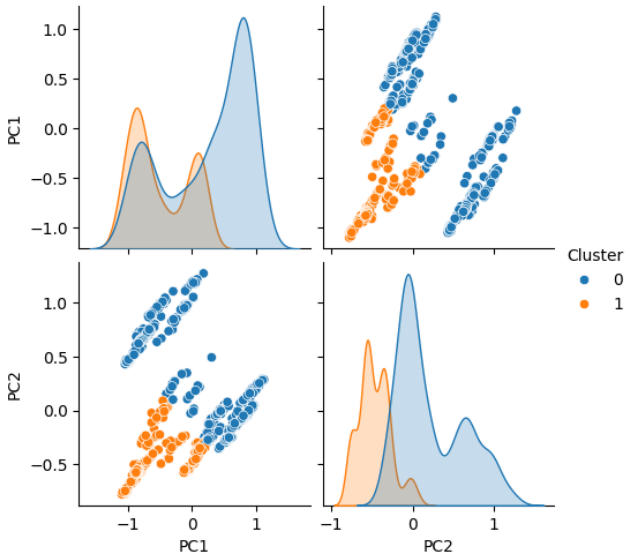


Figure 345: Clusters complete manhattan

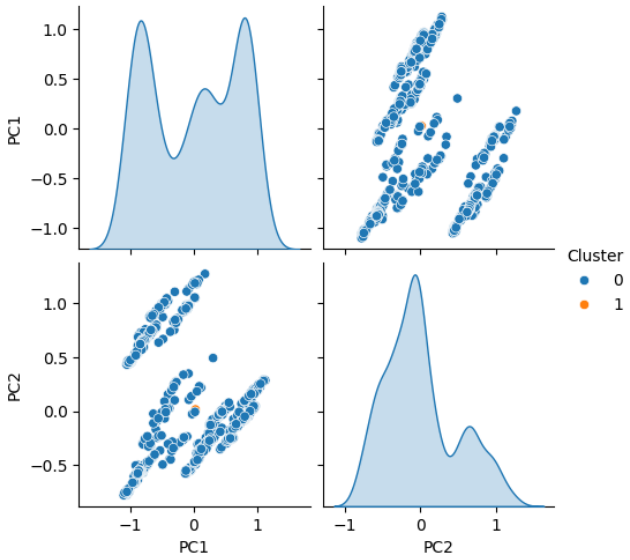


Figure 346: Clusters single correlation

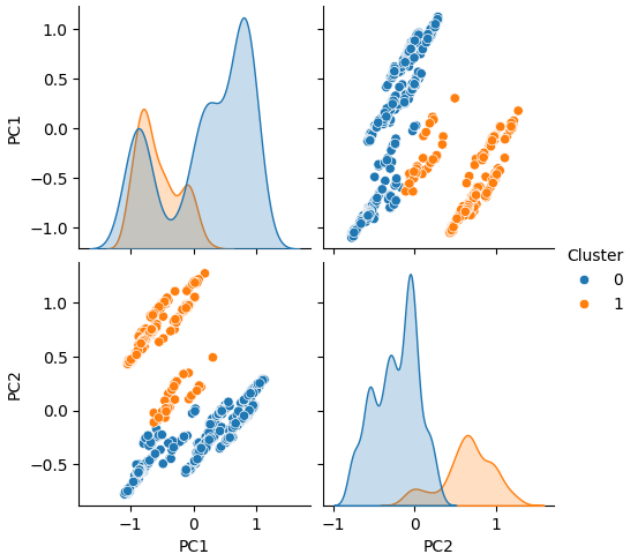


Figure 347: Clusters single cosine

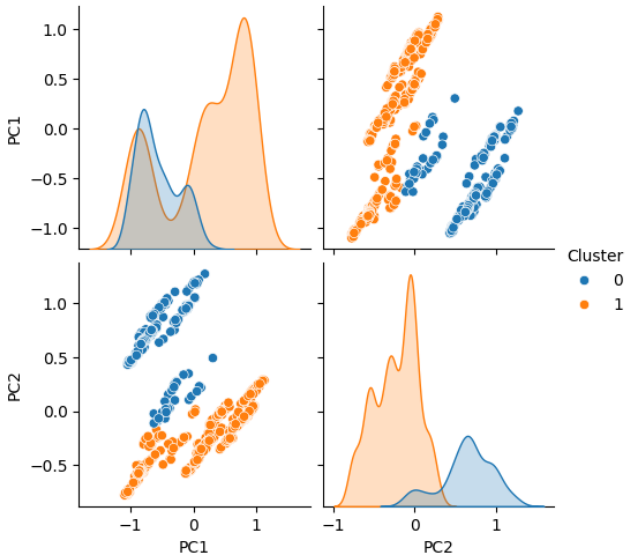


Figure 348: Clusters single euclidean

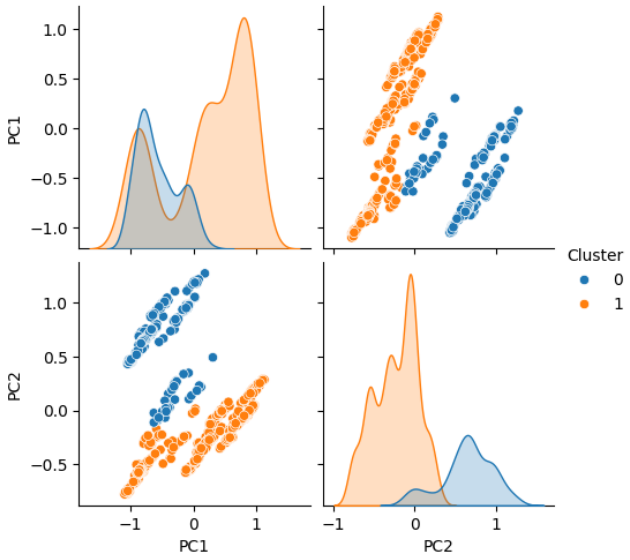


Figure 349: Clusters single manhattan

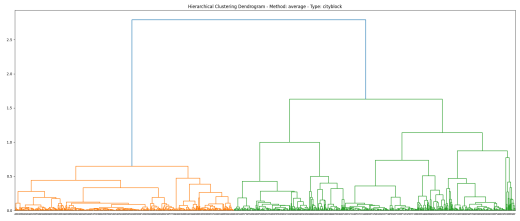


Figure 350: Dendrogram average cityblock

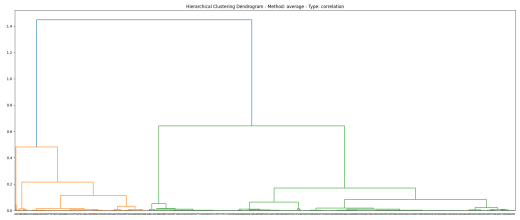


Figure 351: Dendrogram average correlation

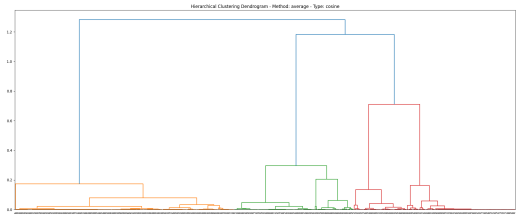


Figure 352: Dendrogram average cosine

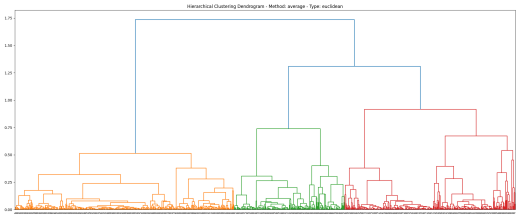


Figure 353: Dendrogram average euclidean

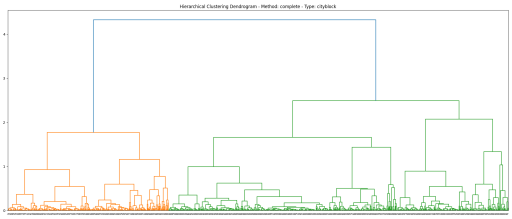


Figure 354: Dendrogram complete cityblock

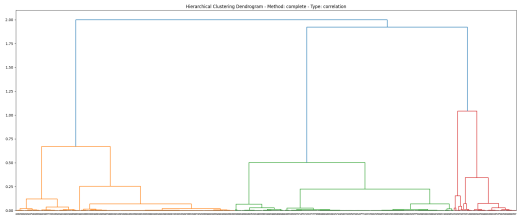


Figure 355: Dendrogram complete correlation

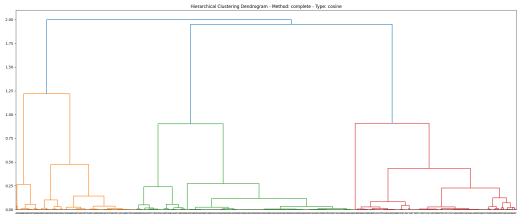


Figure 356: Dendrogram complete cosine

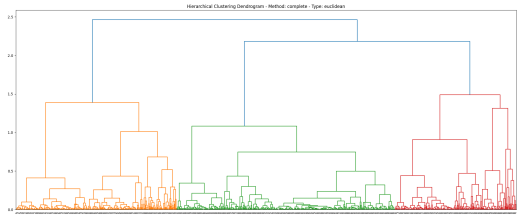


Figure 357: Dendrogram complete euclidean

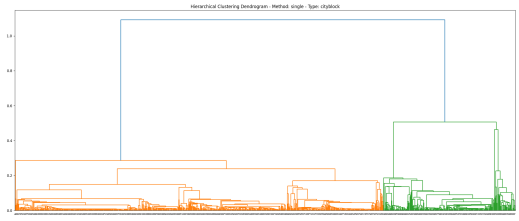


Figure 358: Dendrogram single cityblock

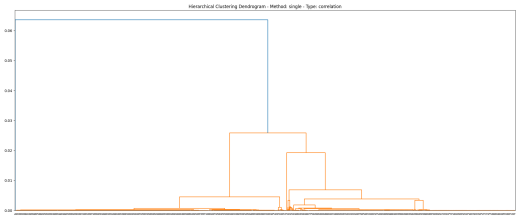


Figure 359: Dendrogram single correlation

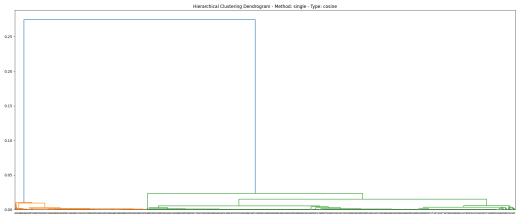


Figure 360: Dendrogram single cosine

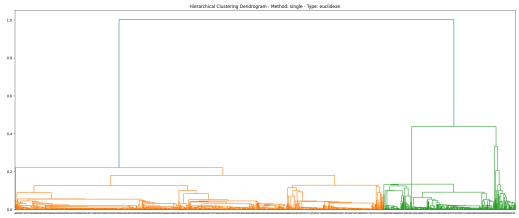


Figure 361: Dendrogram single euclidean

7.2.10 Scenario 5_S

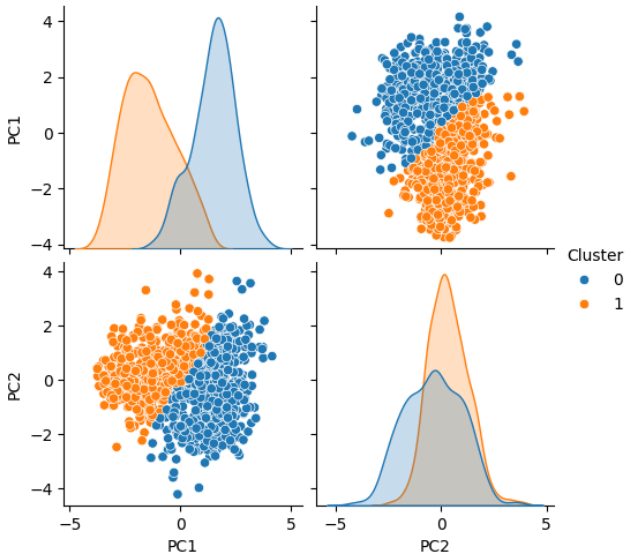


Figure 362: Clusters average correlation

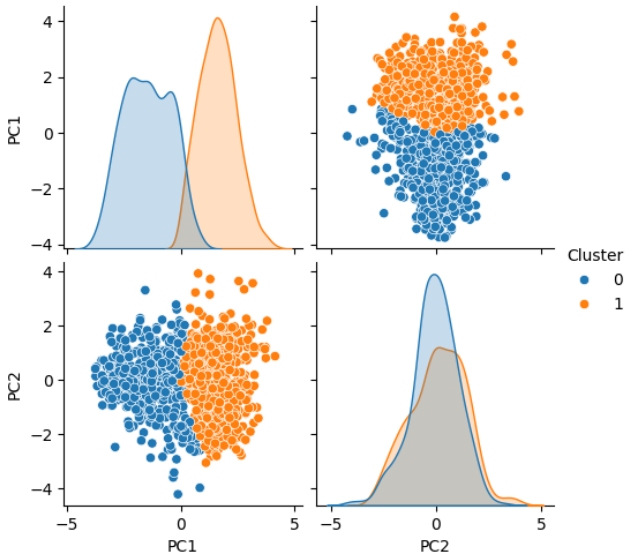


Figure 363: Clusters average cosine

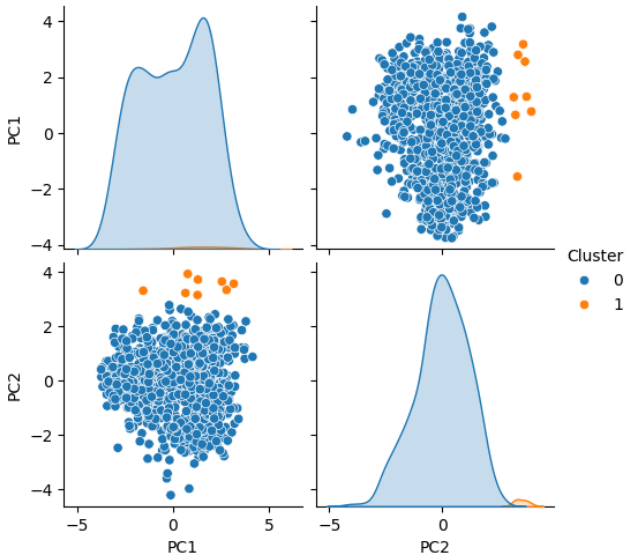


Figure 364: Clusters average euclidean

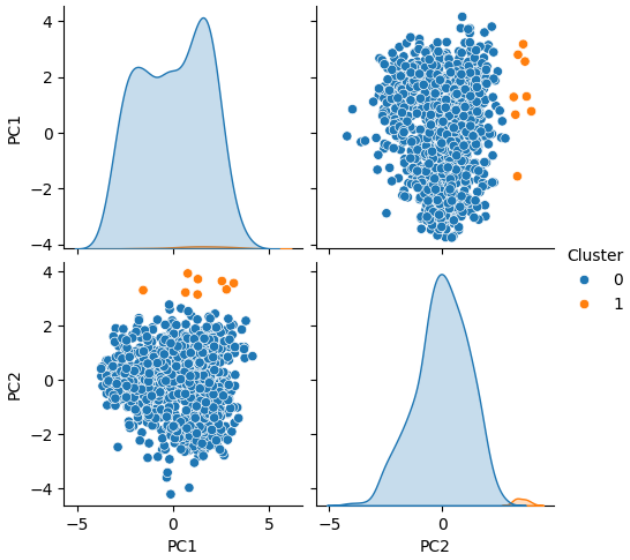


Figure 365: Clusters average manhattan

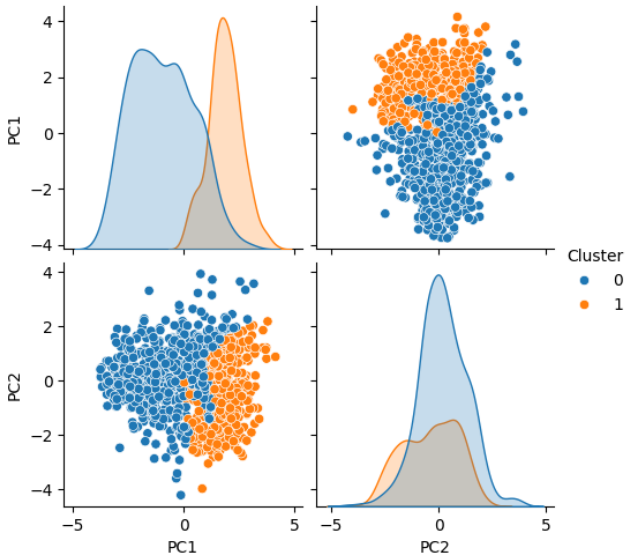


Figure 366: Clusters complete correlation

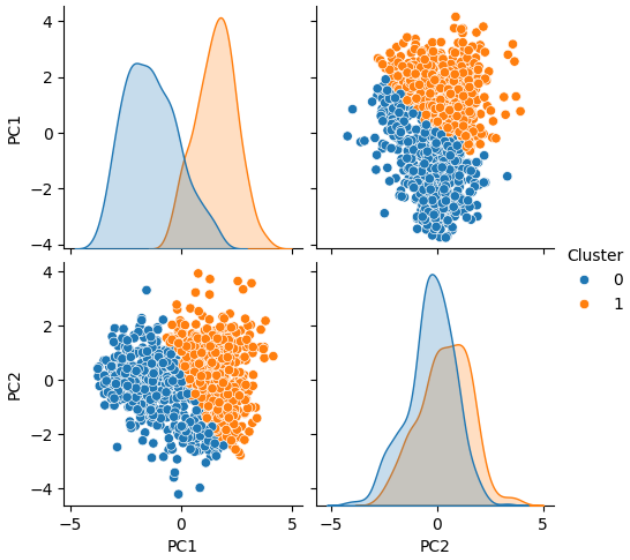


Figure 367: Clusters complete cosine

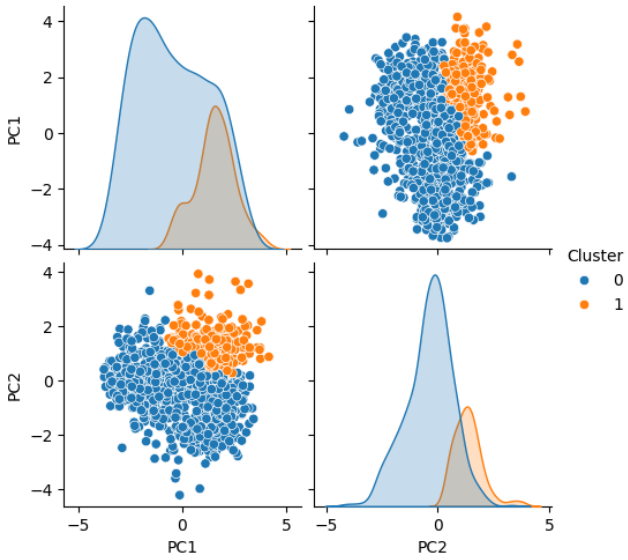


Figure 368: Clusters complete euclidean

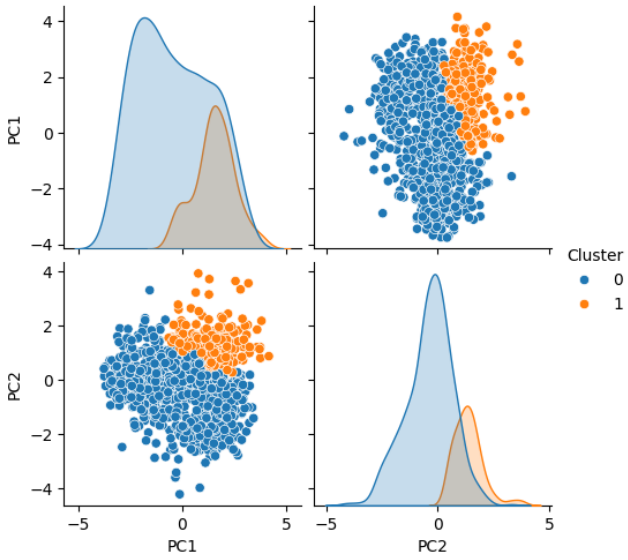


Figure 369: Clusters complete manhattan

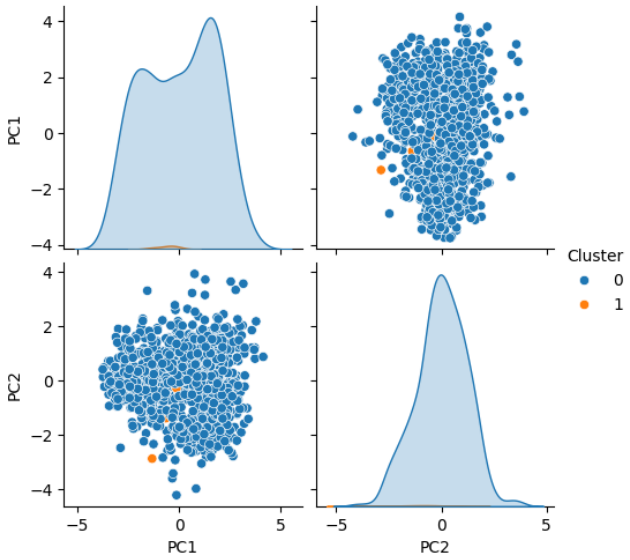


Figure 370: Clusters single correlation

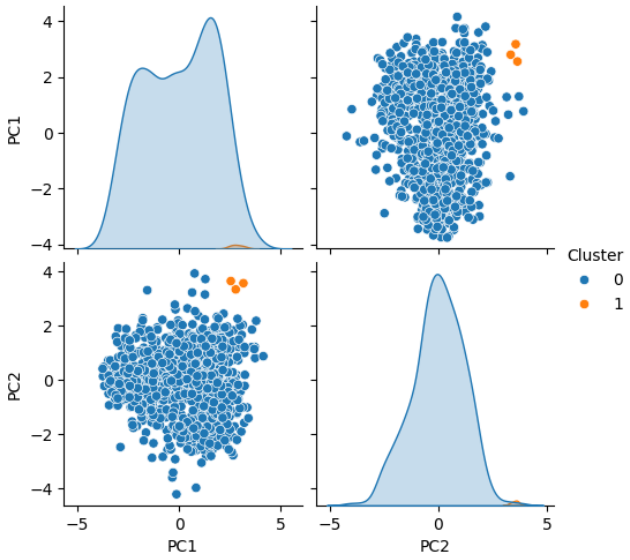


Figure 371: Clusters single cosine

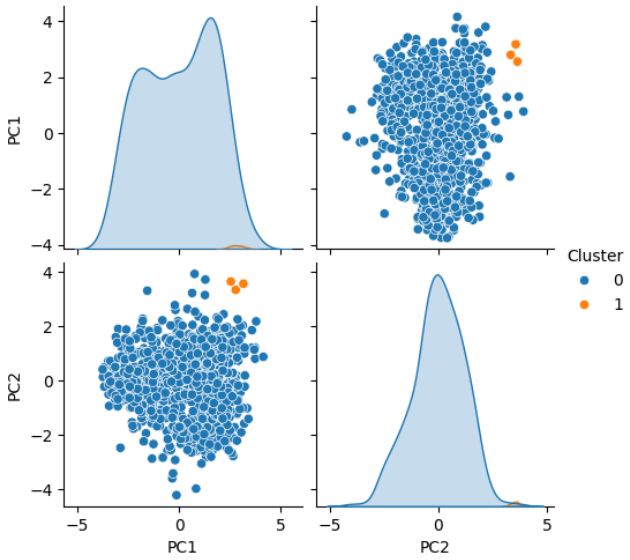


Figure 372: Clusters single euclidean

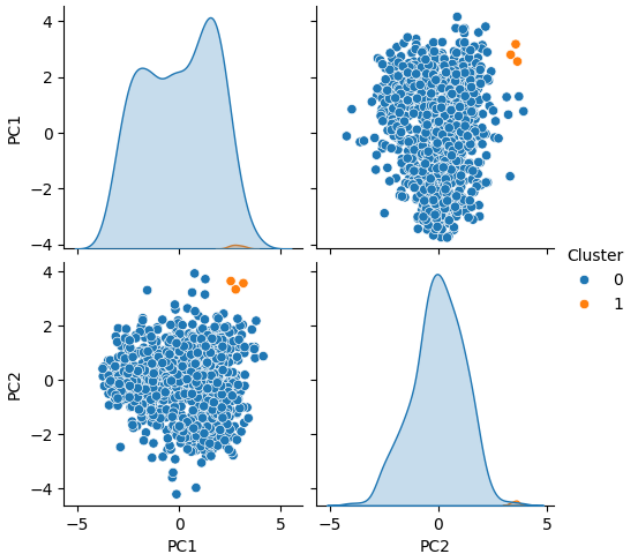


Figure 373: Clusters single manhattan

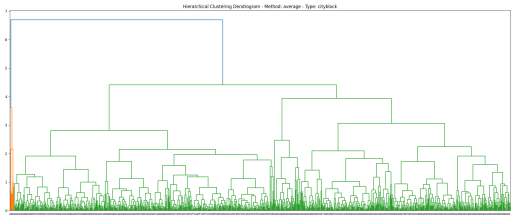


Figure 374: Dendrogram average cityblock

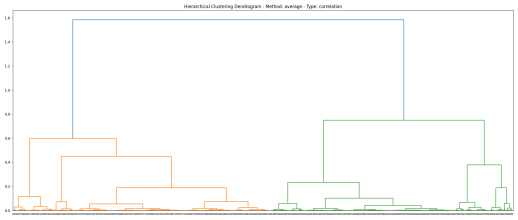


Figure 375: Dendrogram average correlation

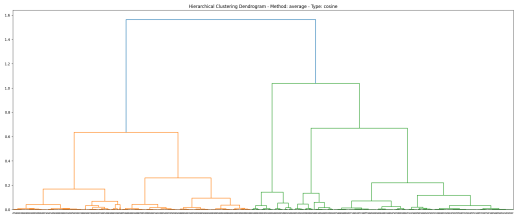


Figure 376: Dendrogram average cosine

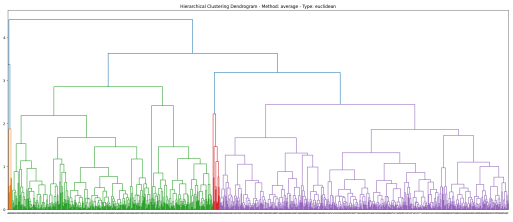


Figure 377: Dendrogram average euclidean

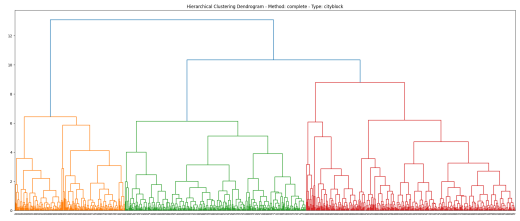


Figure 378: Dendrogram complete cityblock

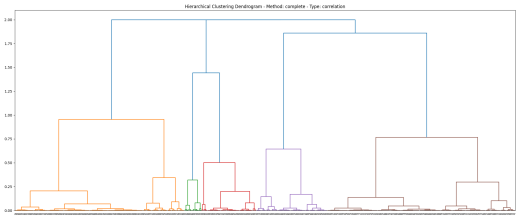


Figure 379: Dendrogram complete correlation

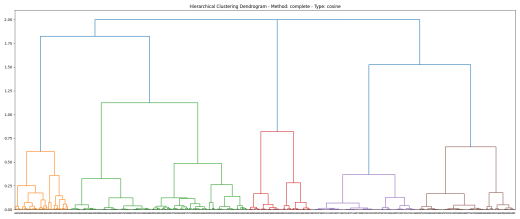


Figure 380: Dendrogram complete cosine

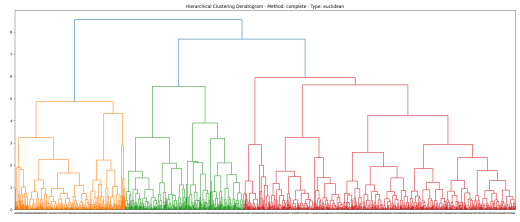


Figure 381: Dendrogram complete euclidean

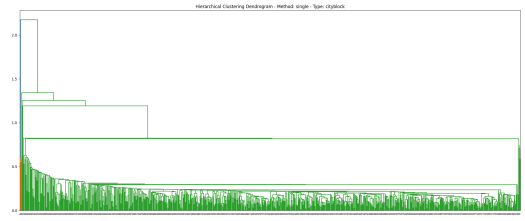


Figure 382: Dendrogram single cityblock

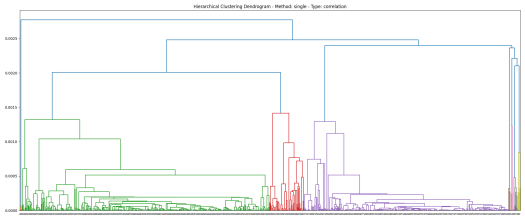


Figure 383: Dendrogram single correlation

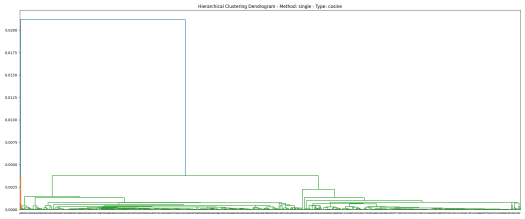


Figure 384: Dendrogram single cosine

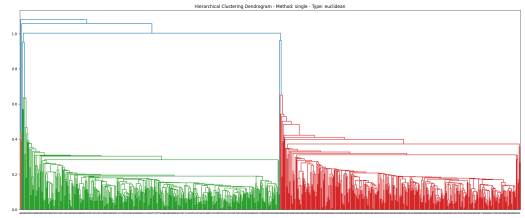


Figure 385: Dendrogram single euclidean

7.2.11 Scenario 6_N

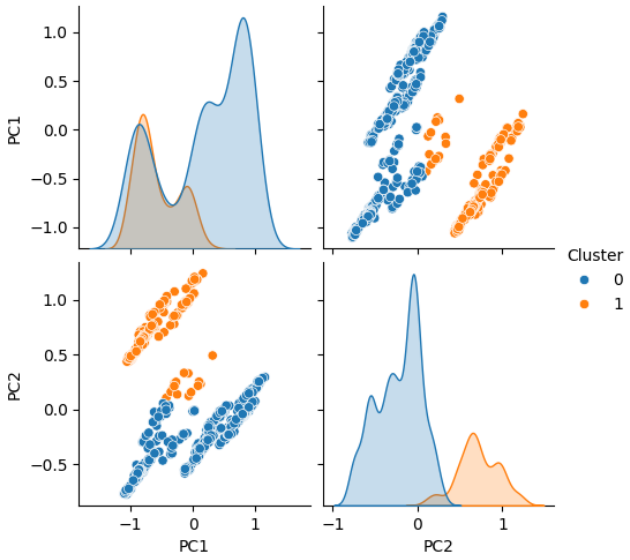


Figure 386: Clusters average correlation

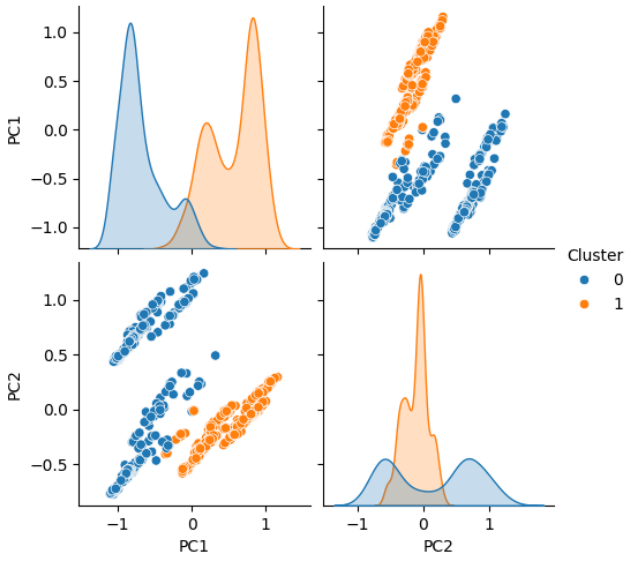


Figure 387: Clusters average cosine

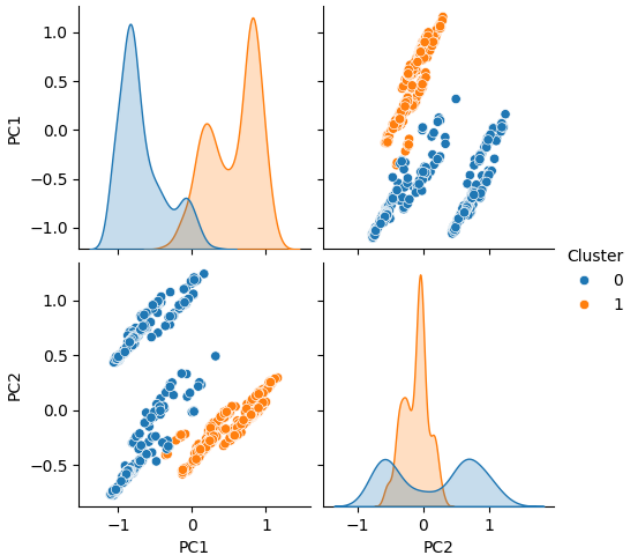


Figure 388: Clusters average euclidean

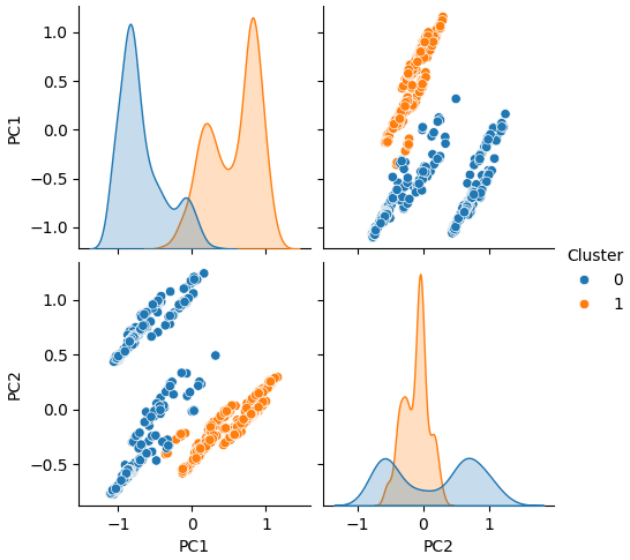


Figure 389: Clusters average manhattan

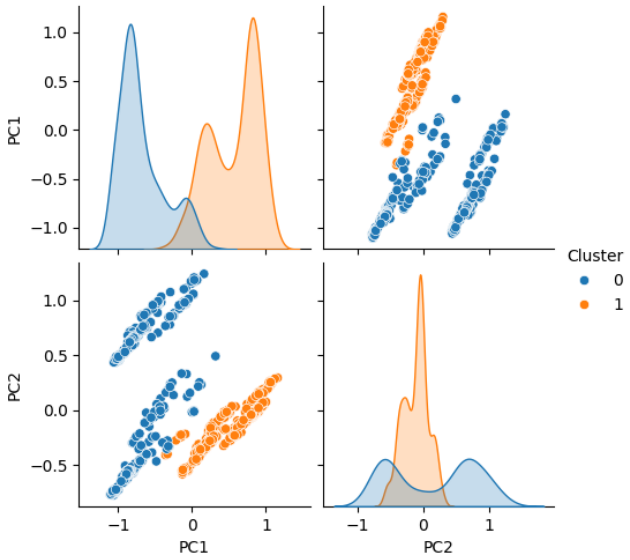


Figure 390: Clusters complete correlation

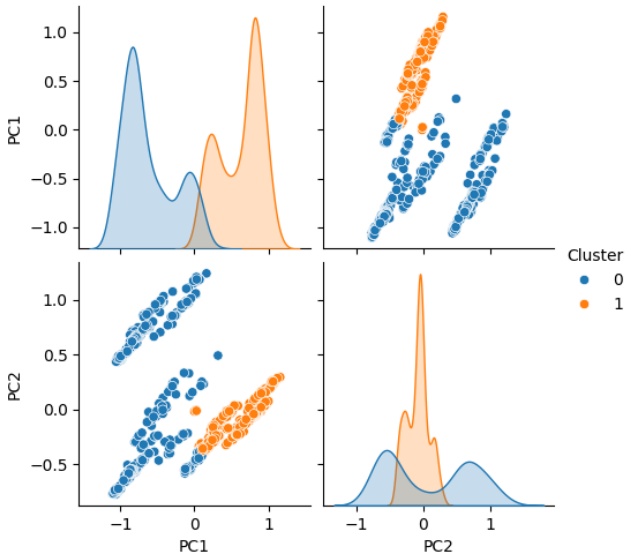


Figure 391: Clusters complete cosine

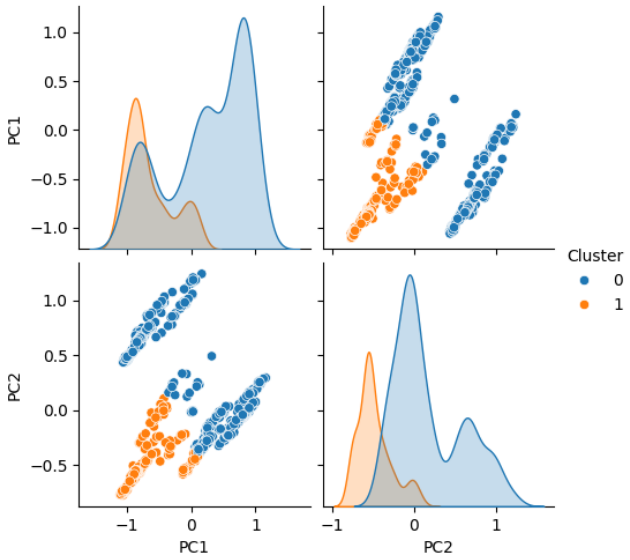


Figure 392: Clusters complete euclidean

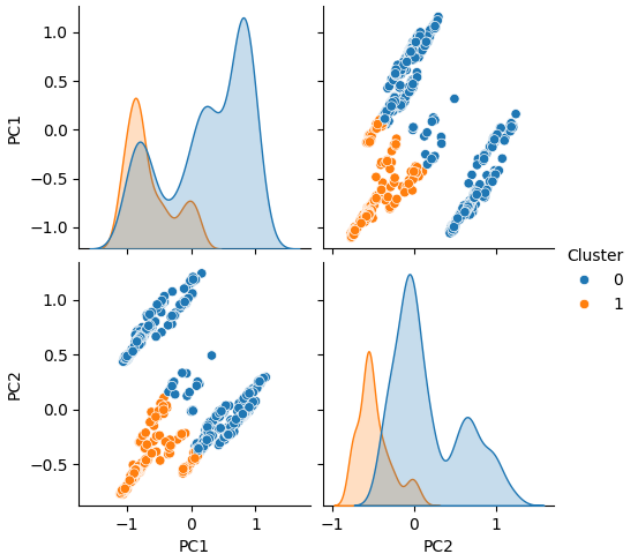


Figure 393: Clusters complete manhattan

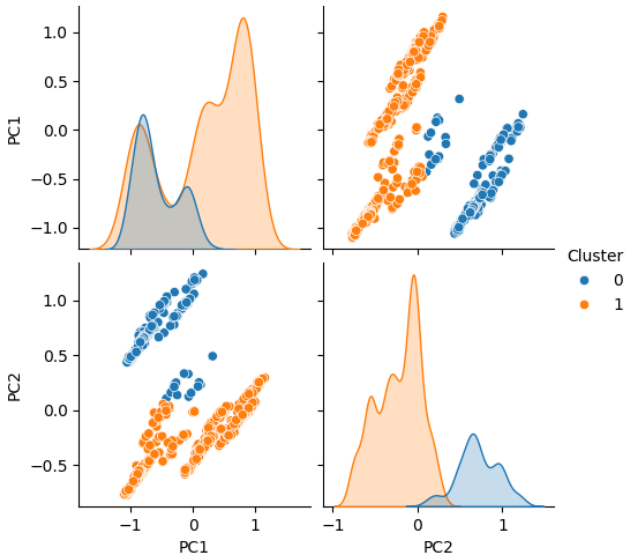


Figure 394: Clusters single correlation

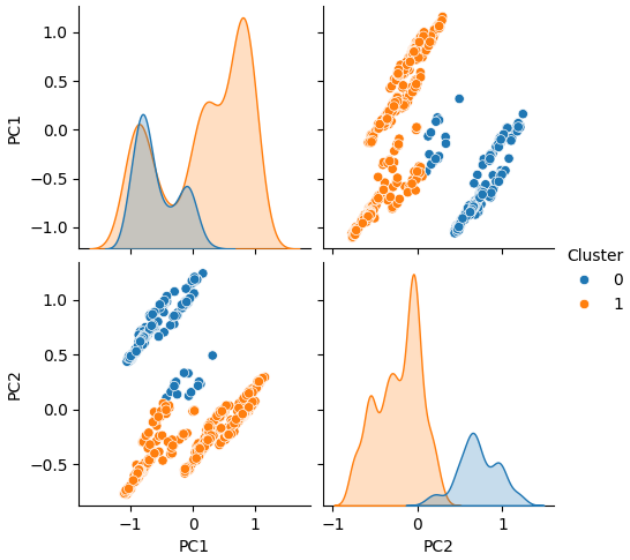


Figure 395: Clusters single cosine

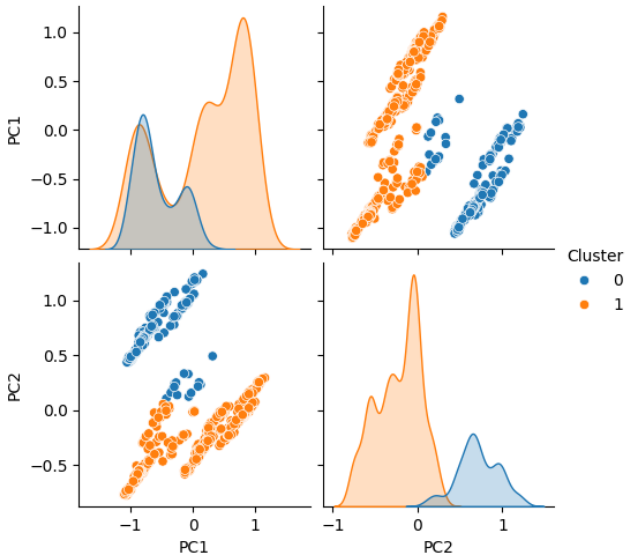


Figure 396: Clusters single euclidean

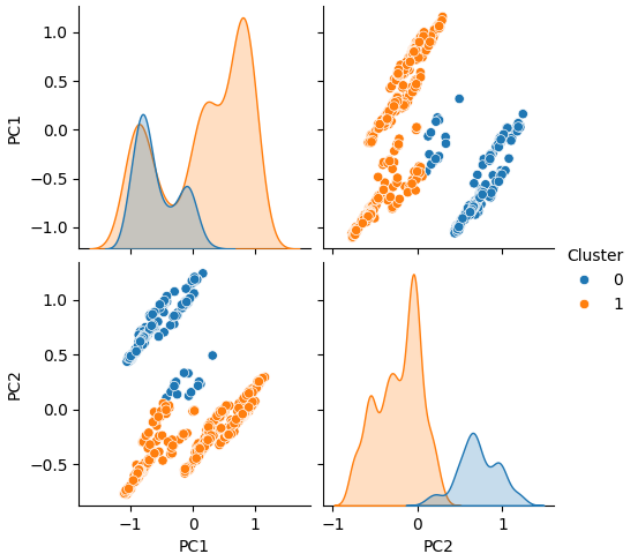


Figure 397: Clusters single manhattan

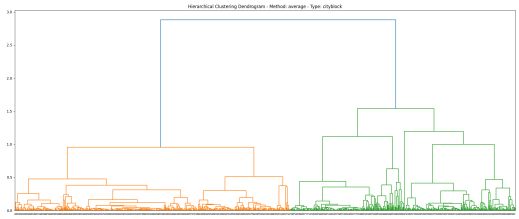


Figure 398: Dendrogram average cityblock

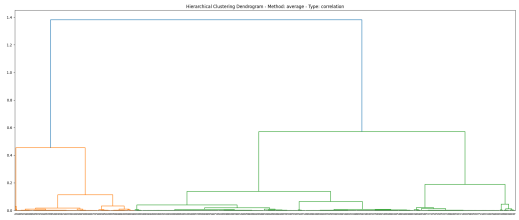


Figure 399: Dendrogram average correlation

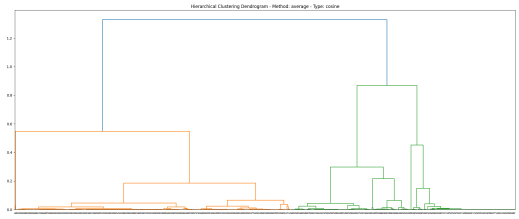


Figure 400: Dendrogram average cosine

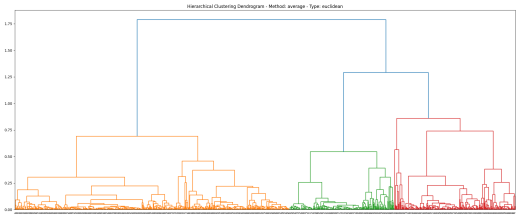


Figure 401: Dendrogram average euclidean

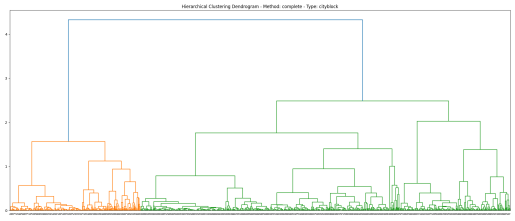


Figure 402: Dendrogram complete cityblock

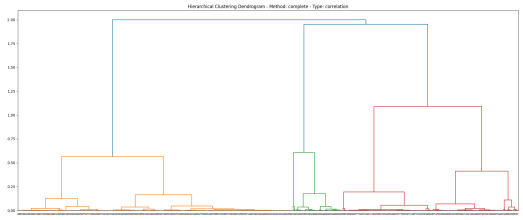


Figure 403: Dendrogram complete correlation

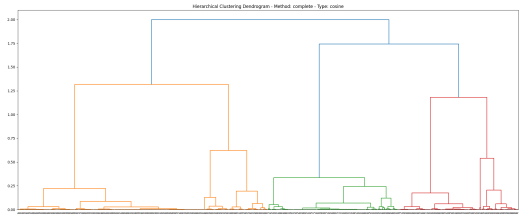


Figure 404: Dendrogram complete cosine

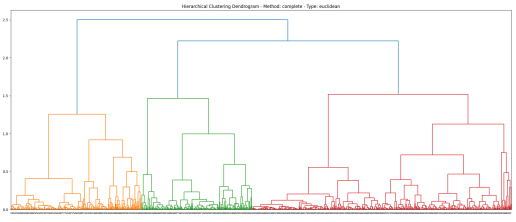


Figure 405: Dendrogram complete euclidean

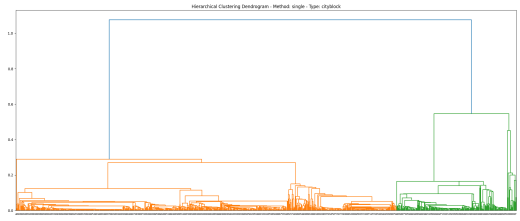


Figure 406: Dendrogram single cityblock

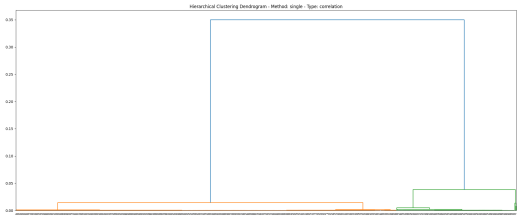


Figure 407: Dendrogram single correlation

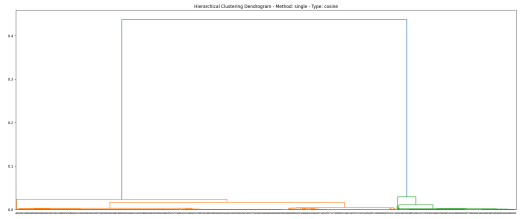


Figure 408: Dendrogram single cosine

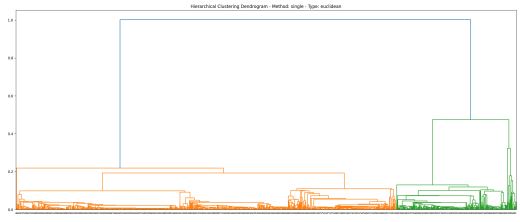


Figure 409: Dendrogram single euclidean

7.2.12 Scenario 6_S

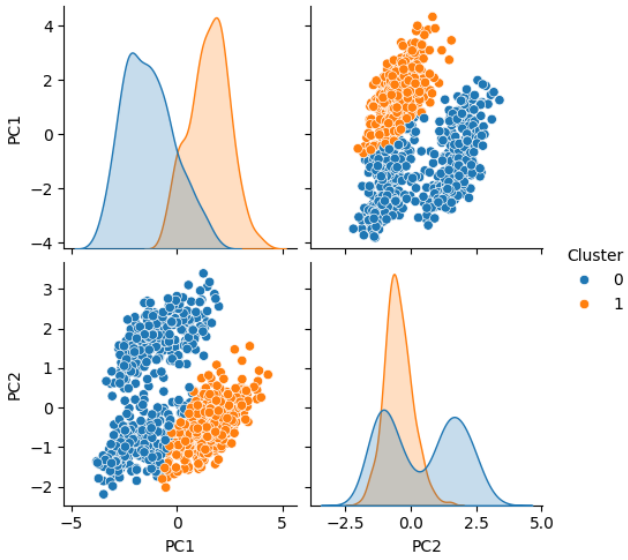


Figure 410: Clusters average correlation

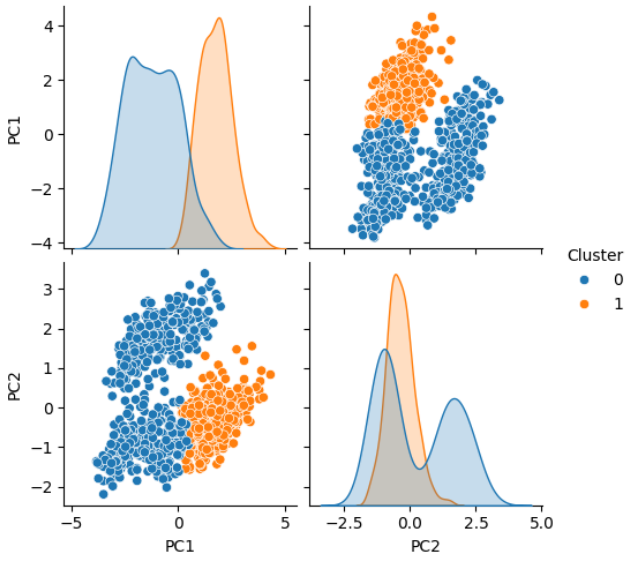


Figure 411: Clusters average cosine

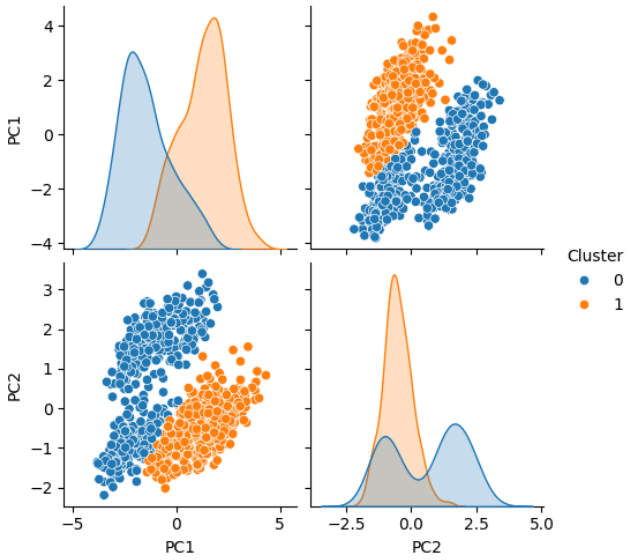


Figure 412: Clusters average euclidean

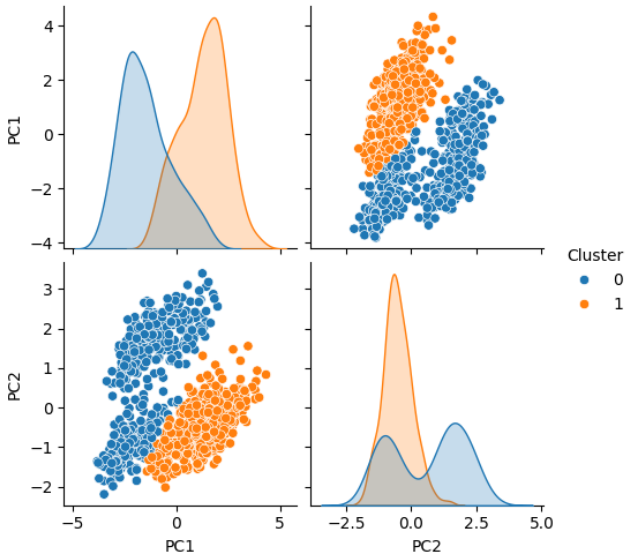


Figure 413: Clusters average manhattan

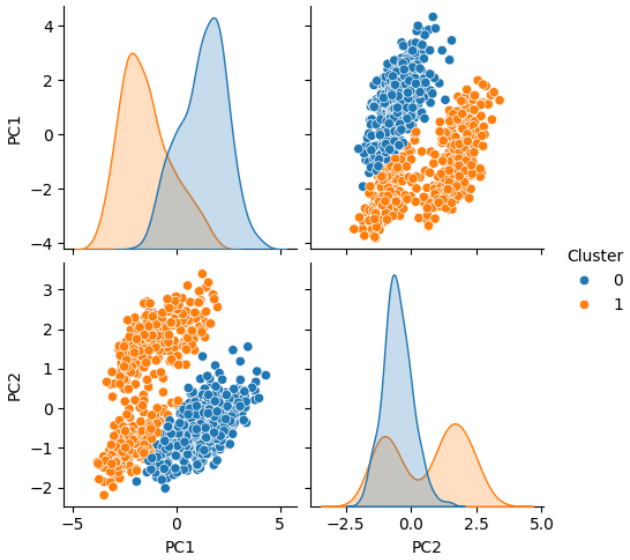


Figure 414: Clusters complete correlation

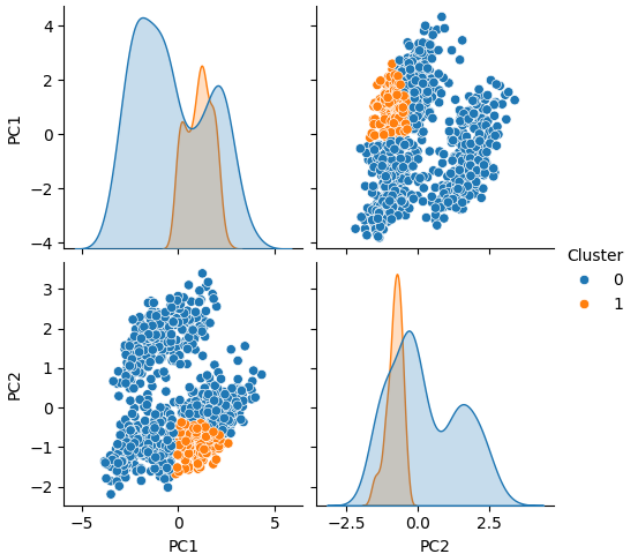


Figure 415: Clusters complete cosine

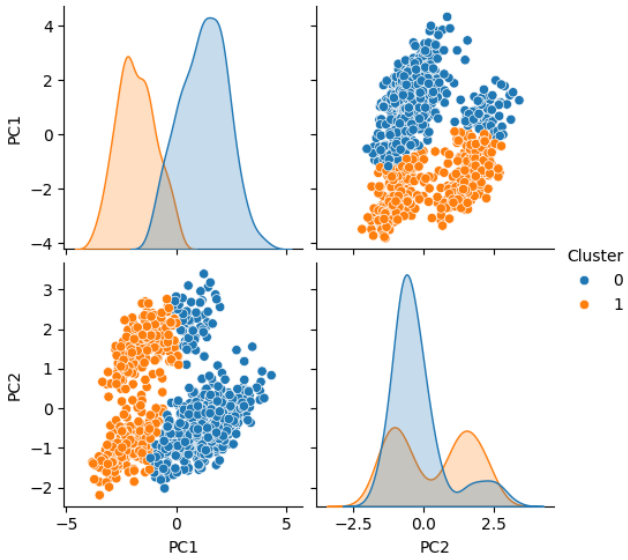


Figure 416: Clusters complete euclidean

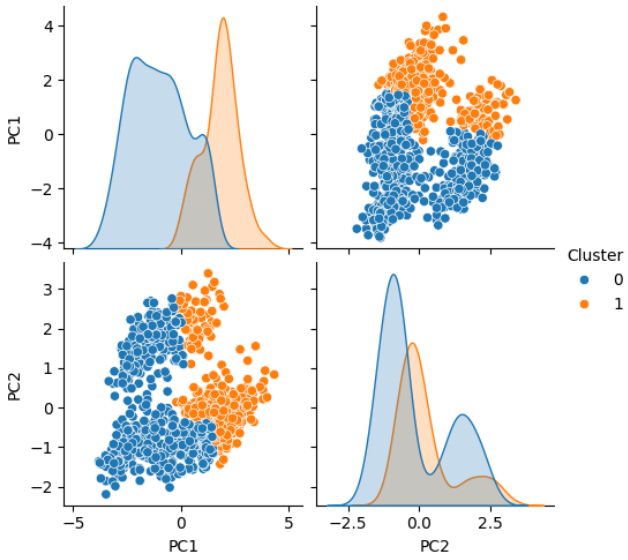


Figure 417: Clusters complete manhattan

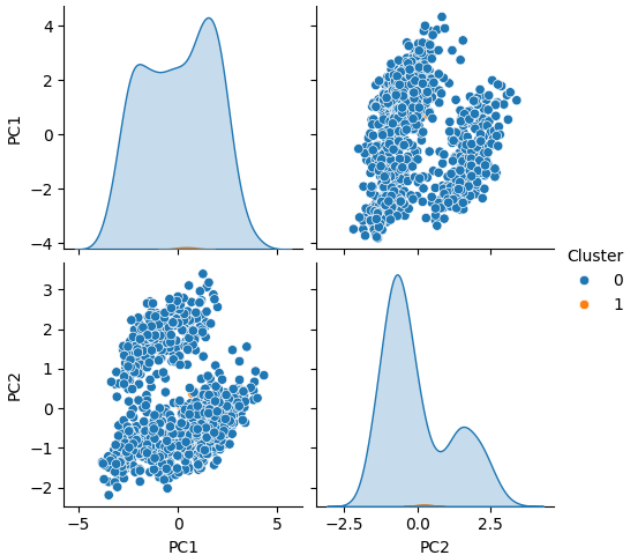


Figure 418: Clusters single correlation

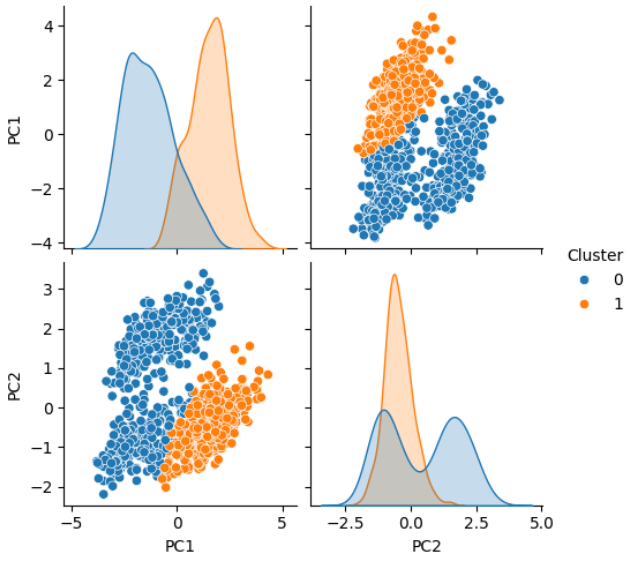


Figure 419: Clusters single cosine

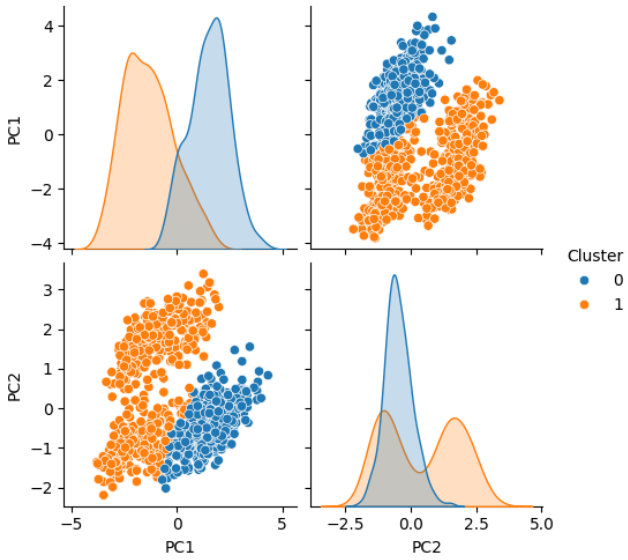


Figure 420: Clusters single euclidean

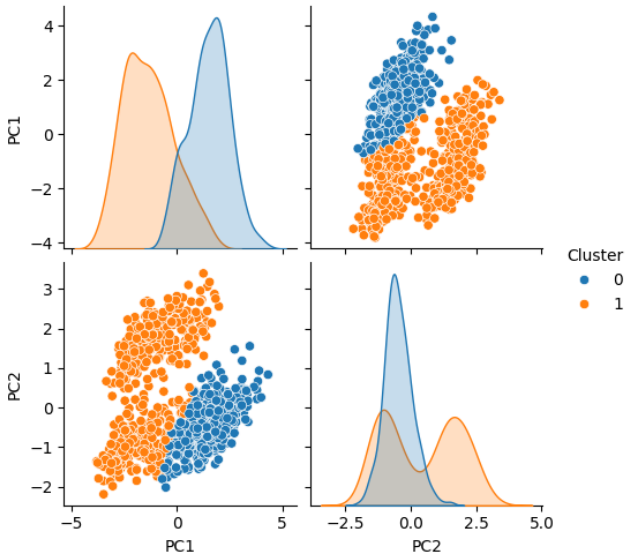


Figure 421: Clusters single manhattan

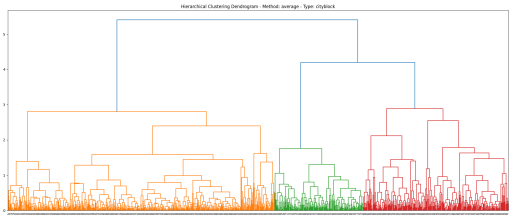


Figure 422: Dendrogram average cityblock

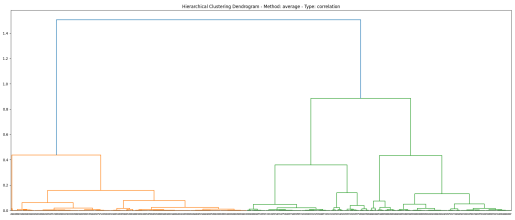


Figure 423: Dendrogram average correlation

Figure 424: Dendrogram average cosine

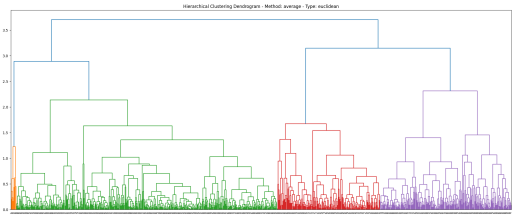


Figure 425: Dendrogram average euclidean

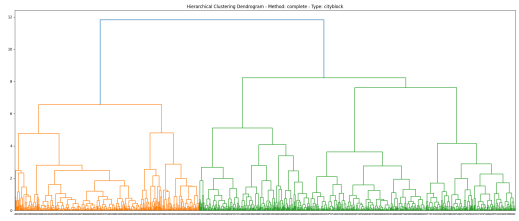


Figure 426: Dendrogram complete cityblock

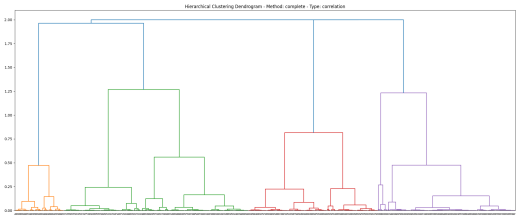


Figure 427: Dendrogram complete correlation

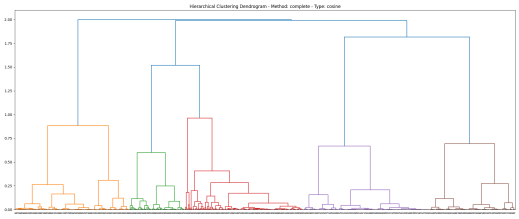


Figure 428: Dendrogram complete cosine

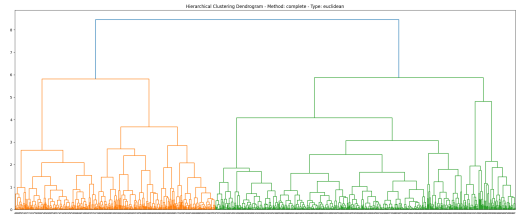


Figure 429: Dendrogram complete euclidean

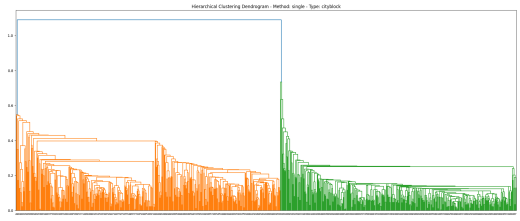


Figure 430: Dendrogram single cityblock

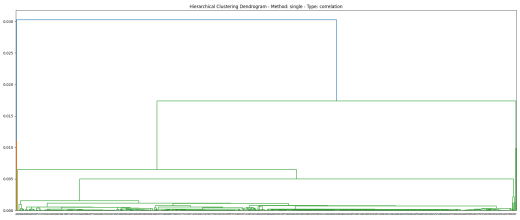


Figure 431: Dendrogram single correlation

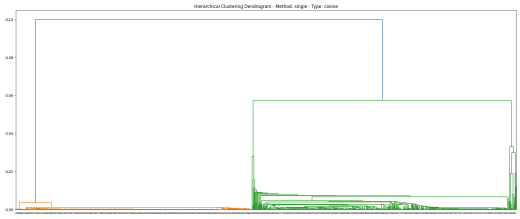


Figure 432: Dendrogram single cosine

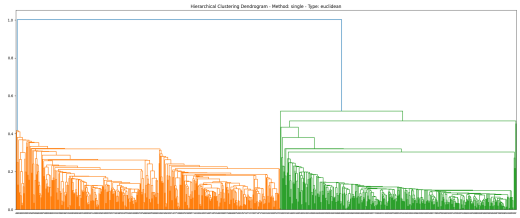


Figure 433: Dendrogram single euclidean

7.2.13 Scenario 7_N

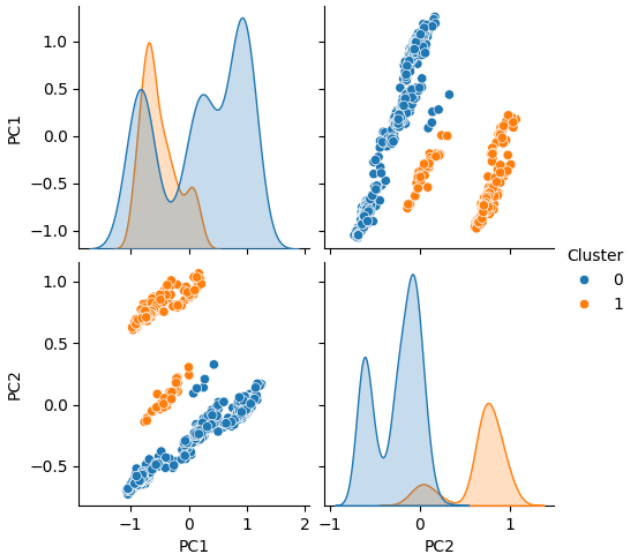


Figure 434: Clusters average correlation

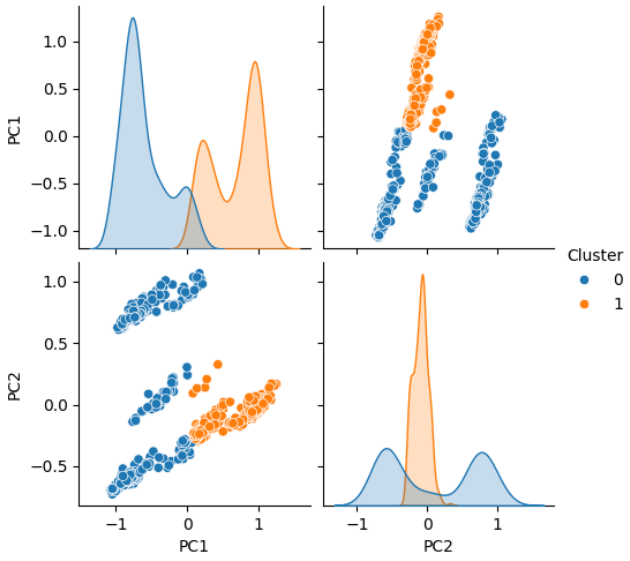


Figure 435: Clusters average cosine

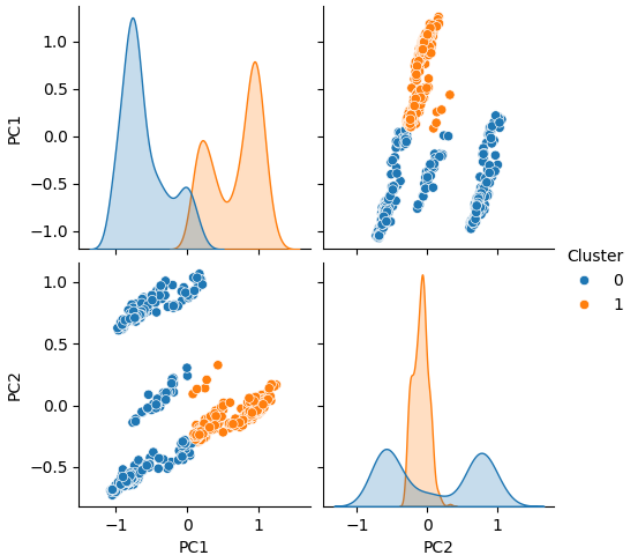


Figure 436: Clusters average euclidean

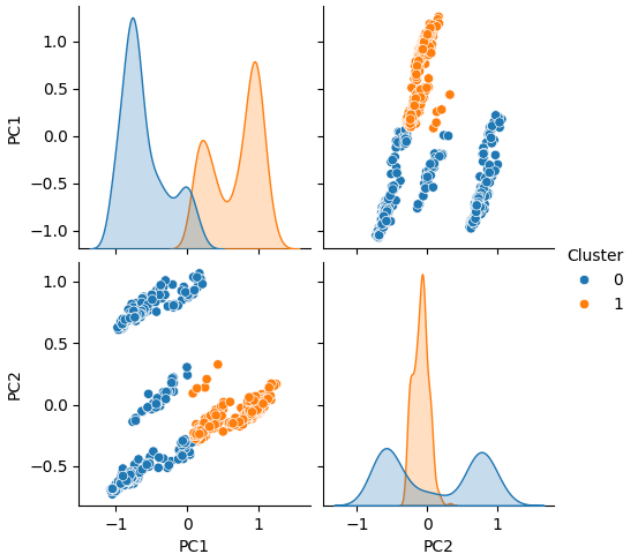


Figure 437: Clusters average manhattan

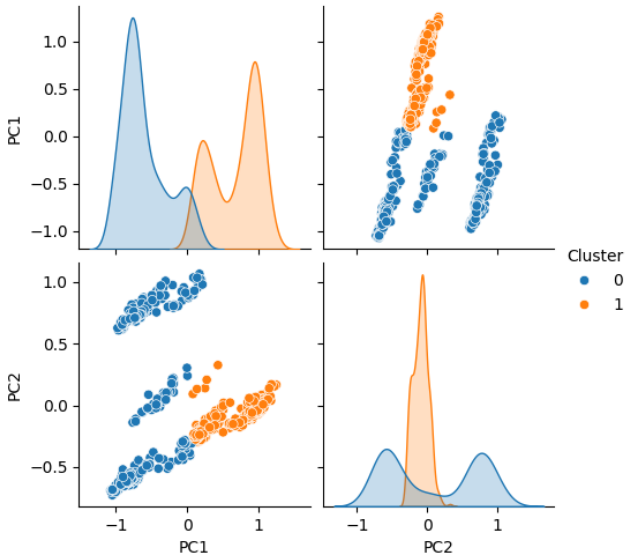


Figure 438: Clusters complete correlation

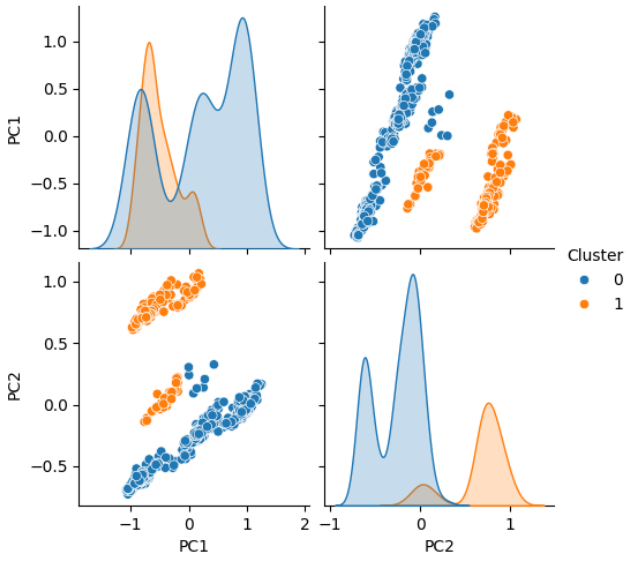


Figure 439: Clusters complete cosine

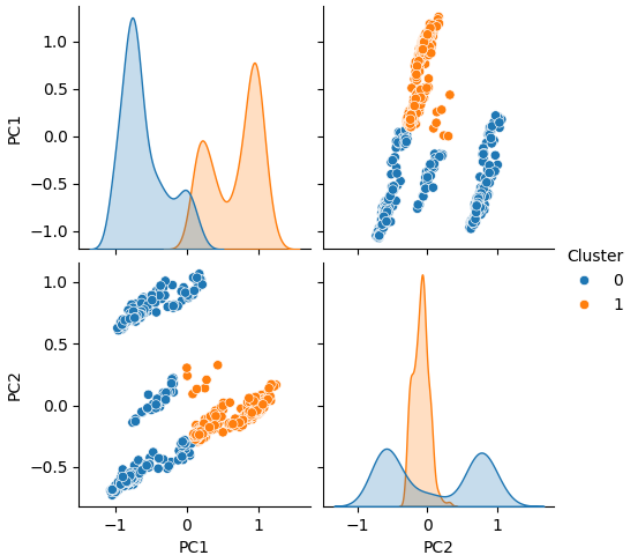


Figure 440: Clusters complete euclidean

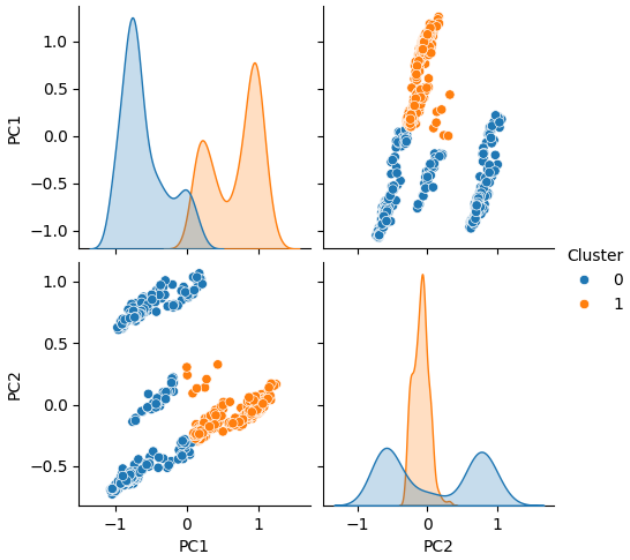


Figure 441: Clusters complete manhattan

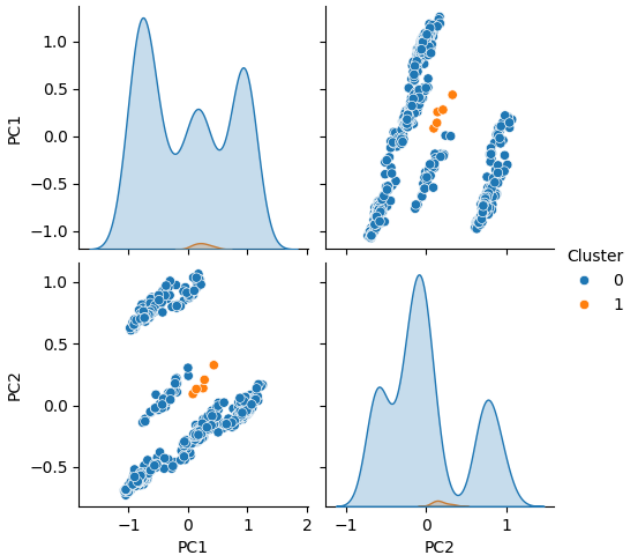


Figure 442: Clusters single correlation

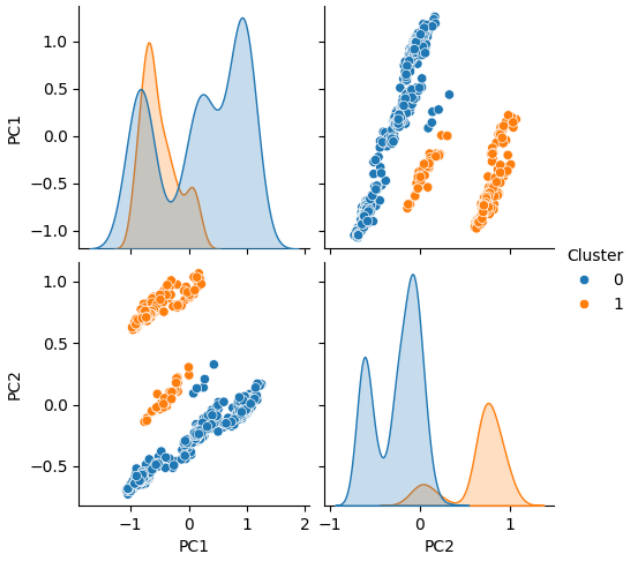


Figure 443: Clusters single cosine

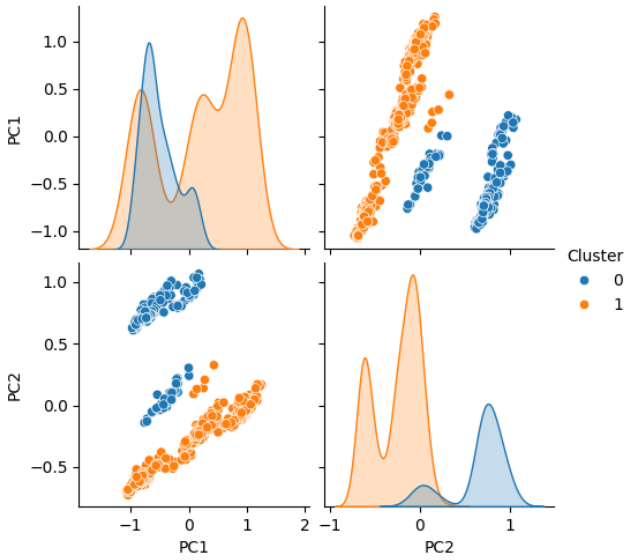


Figure 444: Clusters single euclidean

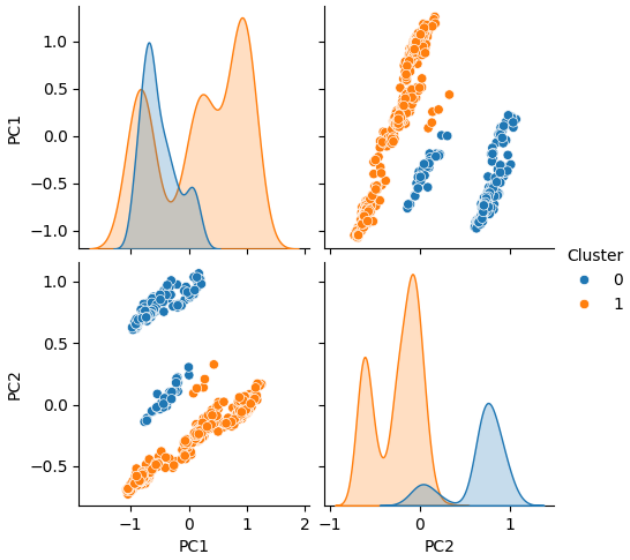


Figure 445: Clusters single manhattan

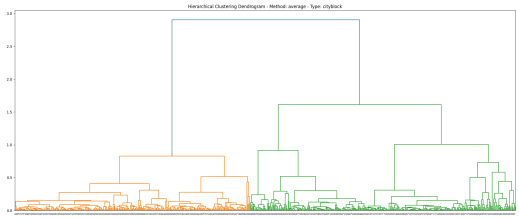


Figure 446: Dendrogram average cityblock

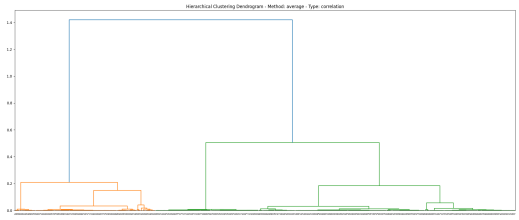


Figure 447: Dendrogram average correlation

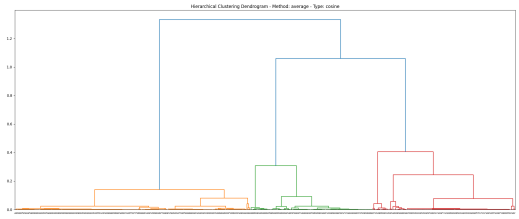


Figure 448: Dendrogram average cosine

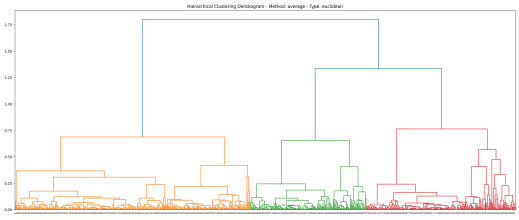


Figure 449: Dendrogram average euclidean

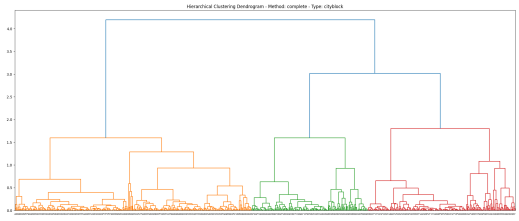


Figure 450: Dendrogram complete cityblock

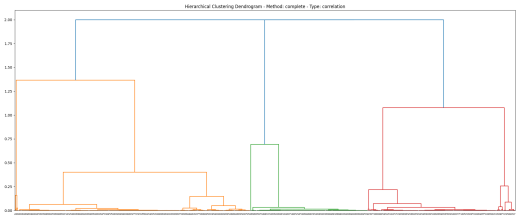


Figure 451: Dendrogram complete correlation

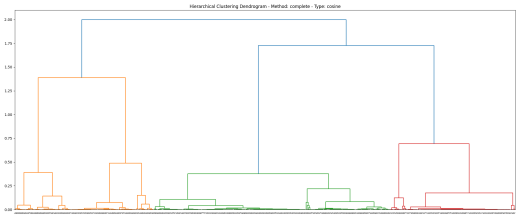


Figure 452: Dendrogram complete cosine

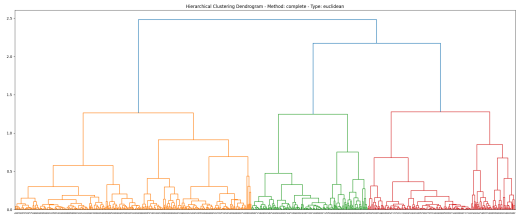


Figure 453: Dendrogram complete euclidean

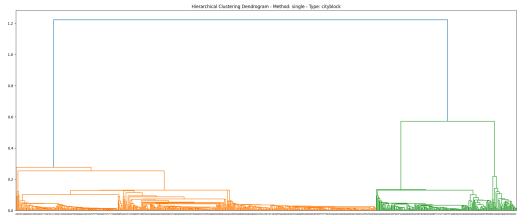


Figure 454: Dendrogram single cityblock

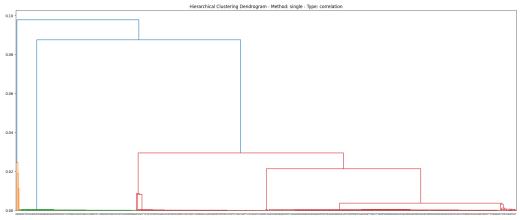


Figure 455: Dendrogram single correlation

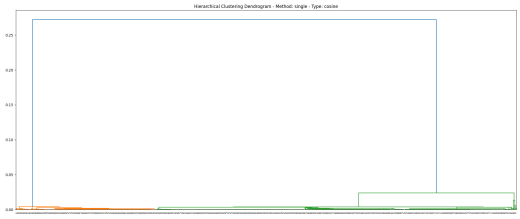


Figure 456: Dendrogram single cosine

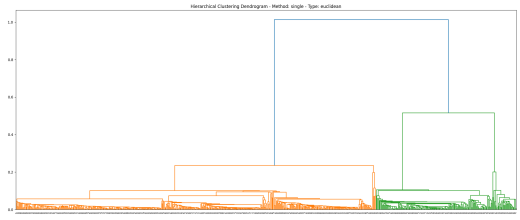


Figure 457: Dendrogram single euclidean

7.2.14 Scenario 7_S

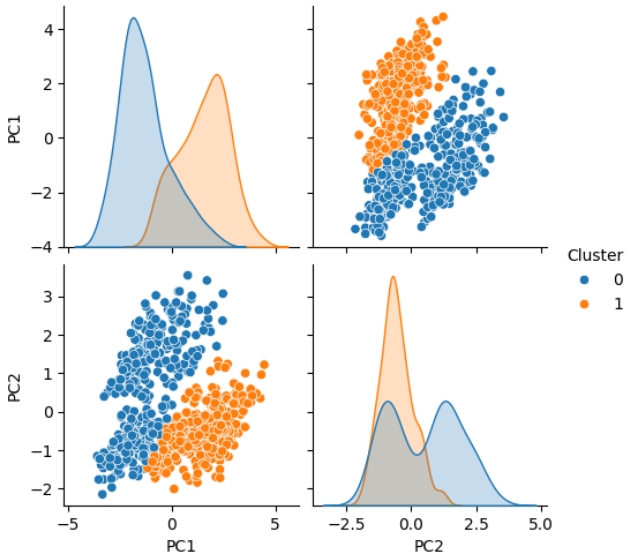


Figure 458: Clusters average correlation

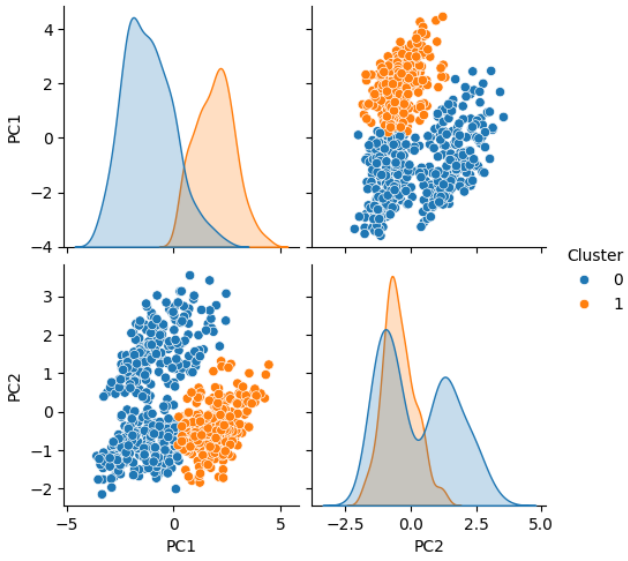


Figure 459: Clusters average cosine

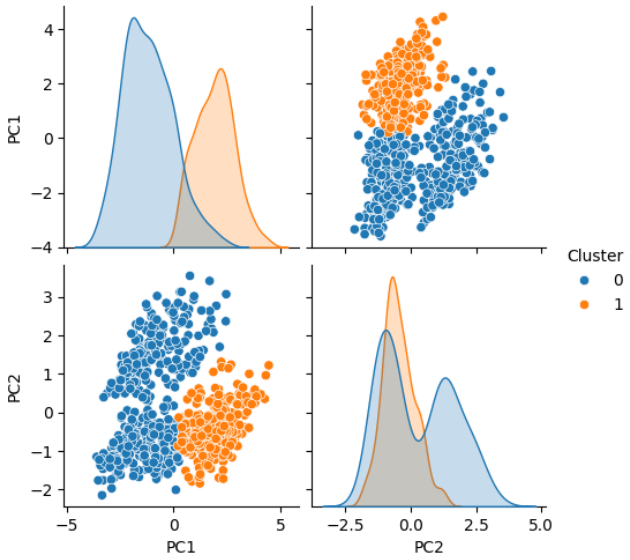


Figure 460: Clusters average euclidean

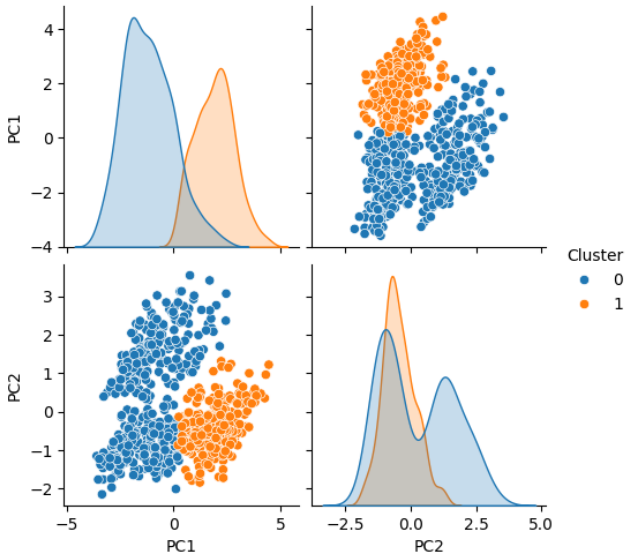


Figure 461: Clusters average manhattan

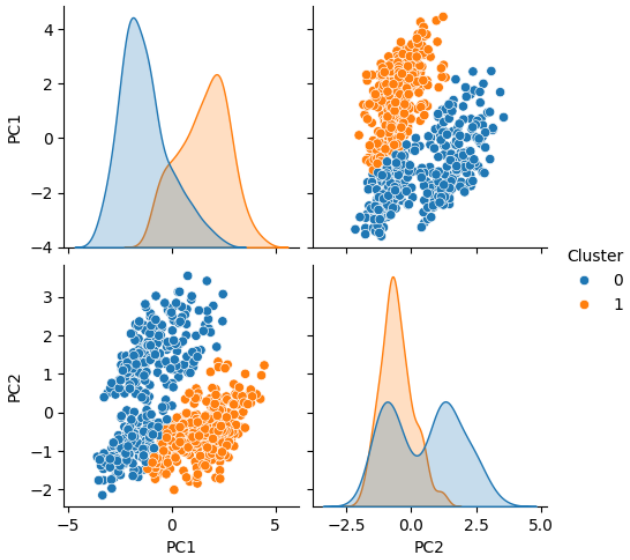


Figure 462: Clusters complete correlation

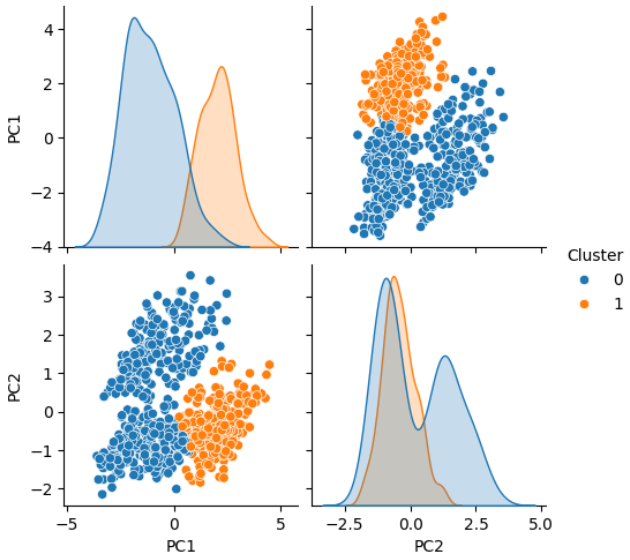


Figure 463: Clusters complete cosine

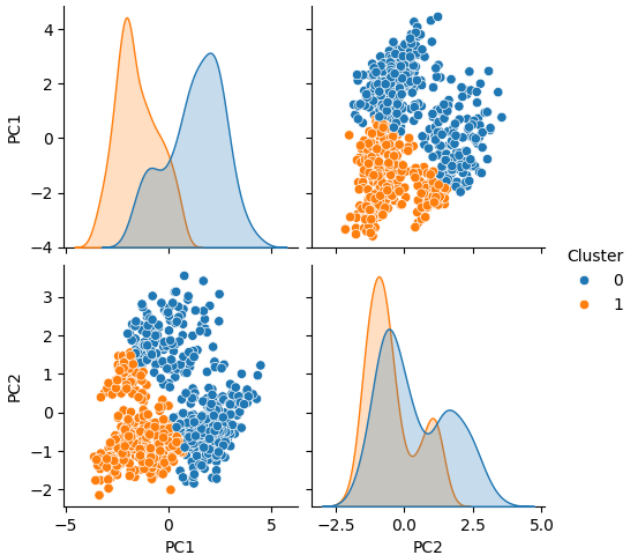


Figure 464: Clusters complete euclidean

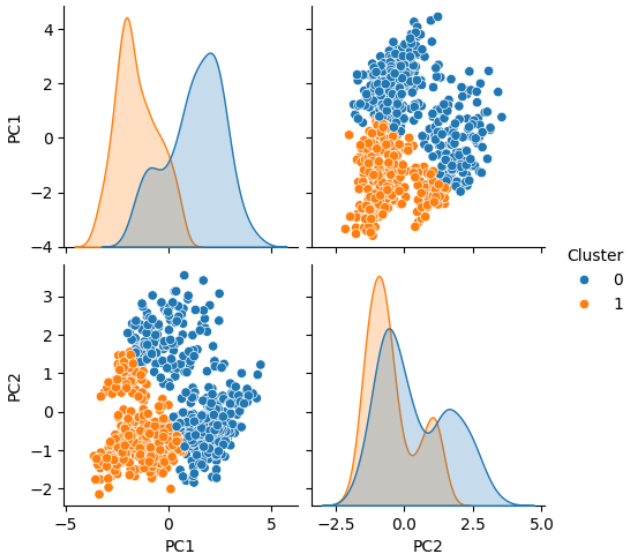


Figure 465: Clusters complete manhattan

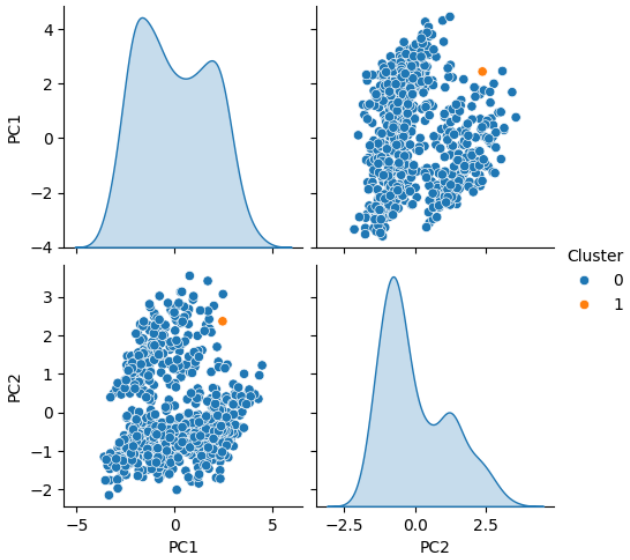


Figure 466: Clusters single correlation

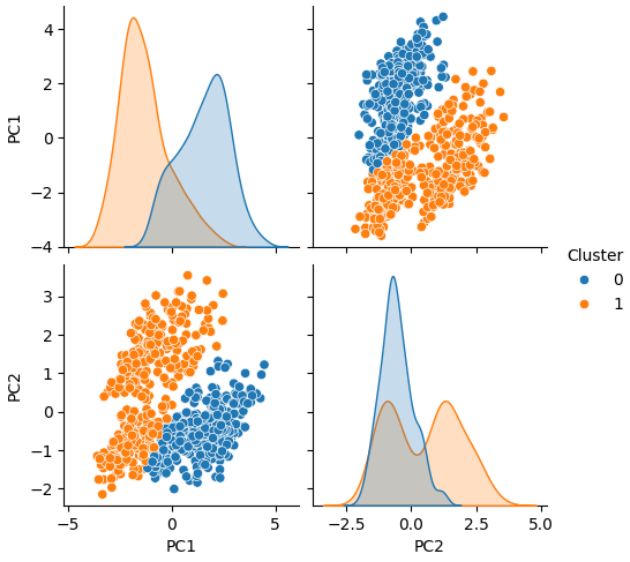


Figure 467: Clusters single cosine

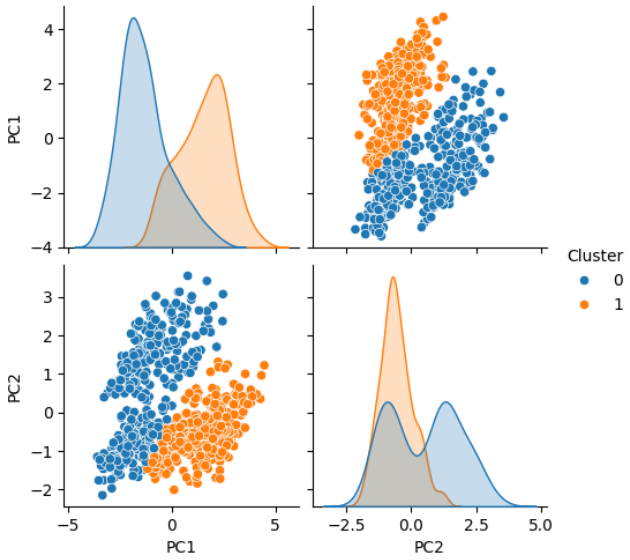


Figure 468: Clusters single euclidean

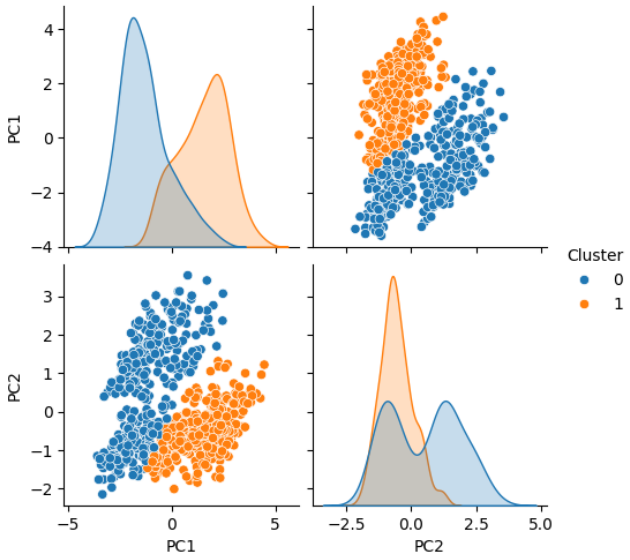


Figure 469: Clusters single manhattan

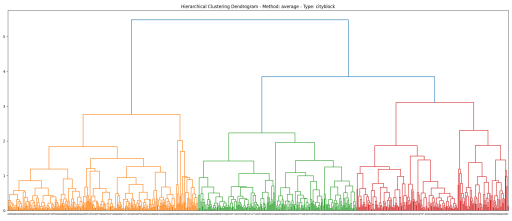


Figure 470: Dendrogram average cityblock

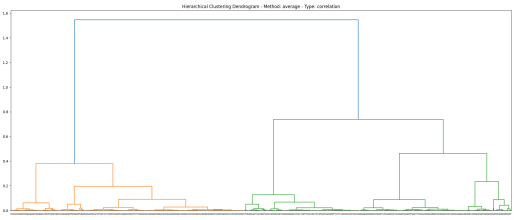


Figure 471: Dendrogram average correlation

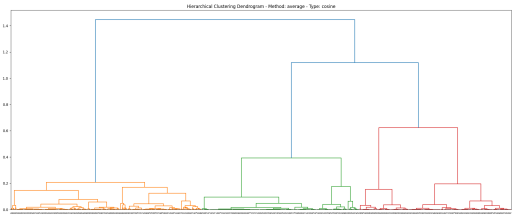


Figure 472: Dendrogram average cosine

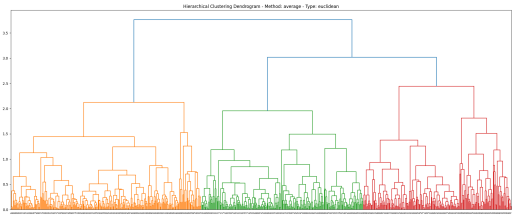


Figure 473: Dendrogram average euclidean

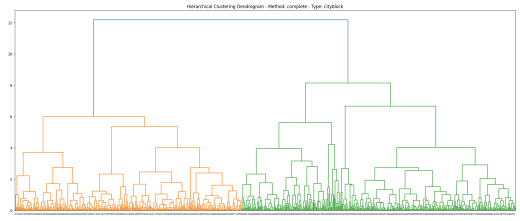


Figure 474: Dendrogram complete cityblock

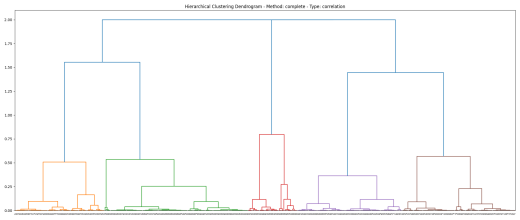


Figure 475: Dendrogram complete correlation

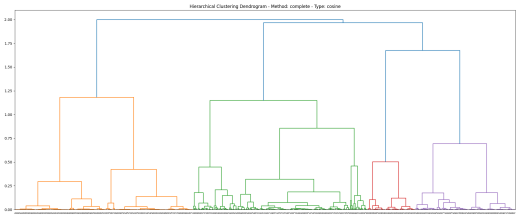


Figure 476: Dendrogram complete cosine

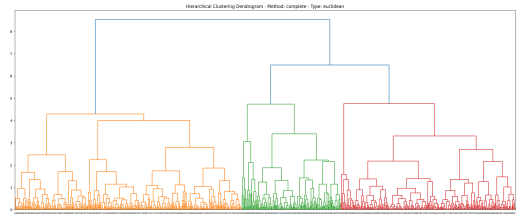


Figure 477: Dendrogram complete euclidean

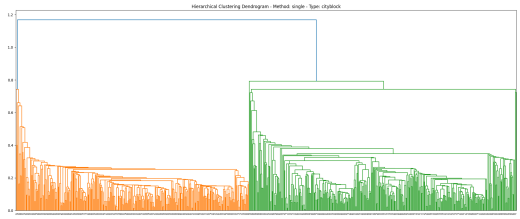


Figure 478: Dendrogram single cityblock

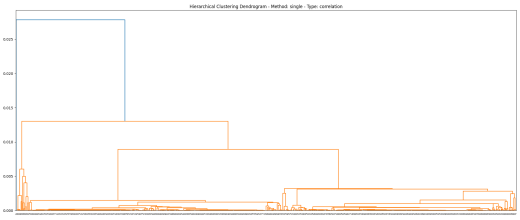


Figure 479: Dendrogram single correlation

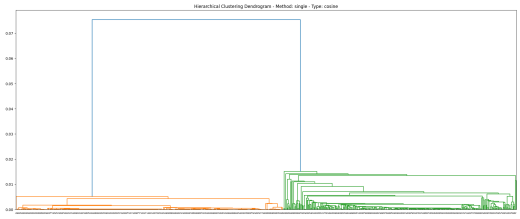


Figure 480: Dendrogram single cosine

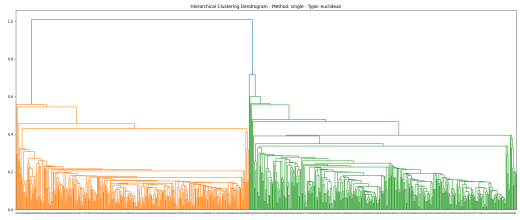


Figure 481: Dendrogram single euclidean

7.2.15 Scenario 8_N

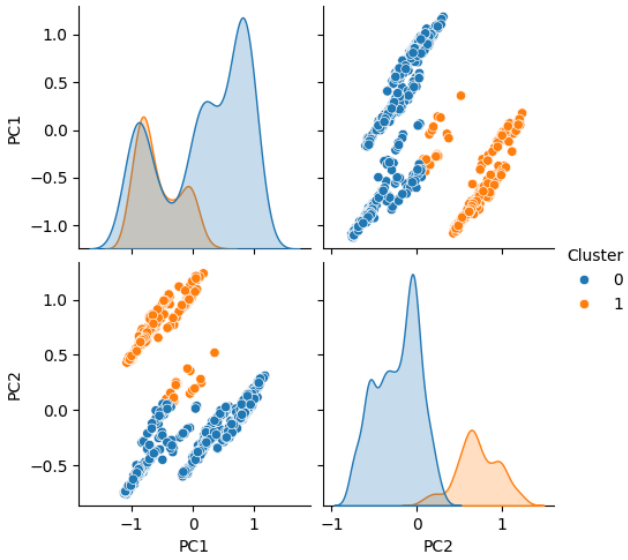


Figure 482: Clusters average correlation

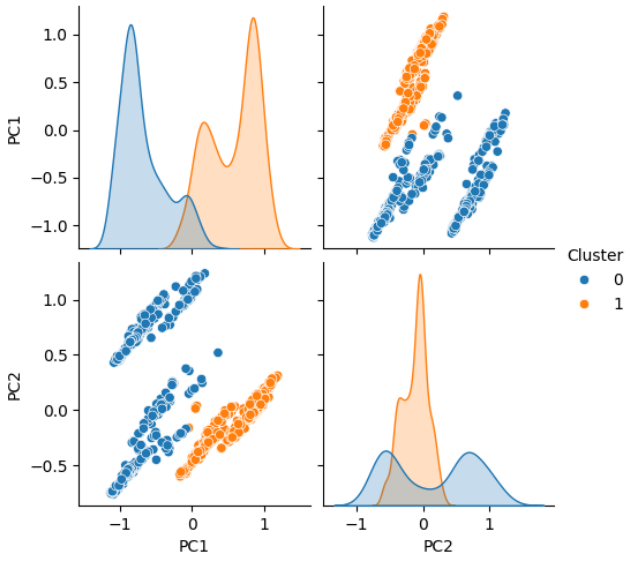


Figure 483: Clusters average cosine

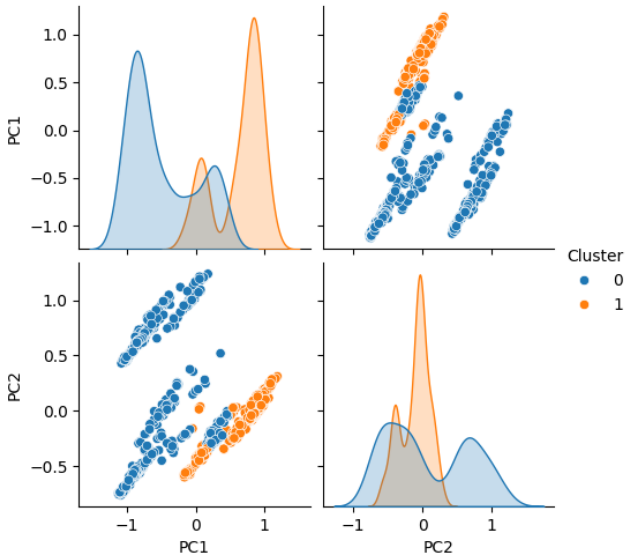


Figure 484: Clusters average euclidean

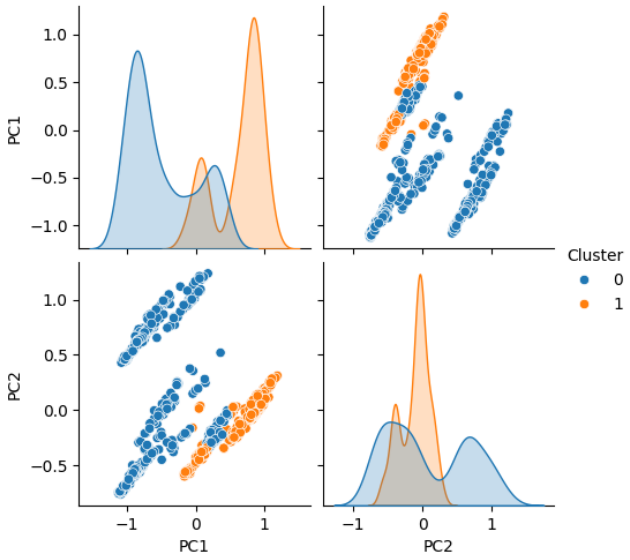


Figure 485: Clusters average manhattan

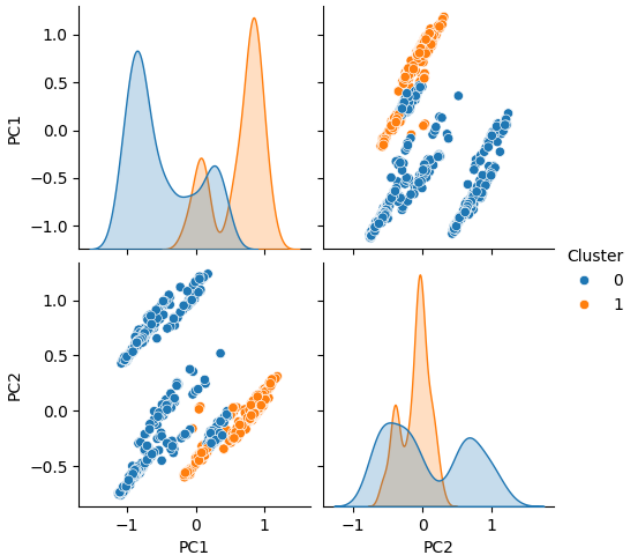


Figure 486: Clusters complete correlation

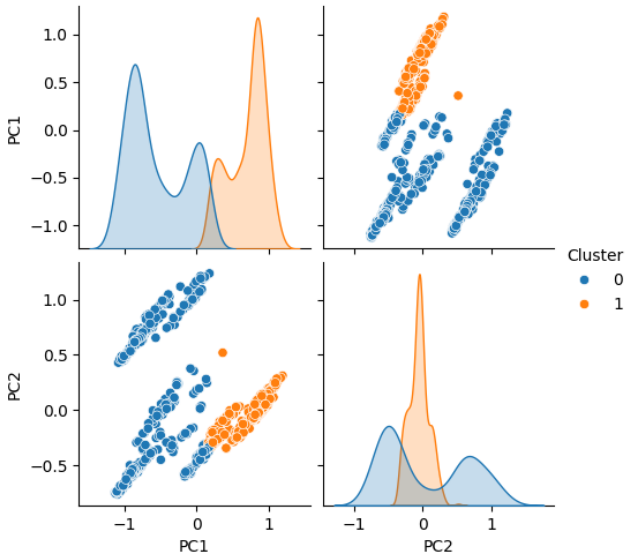


Figure 487: Clusters complete cosine

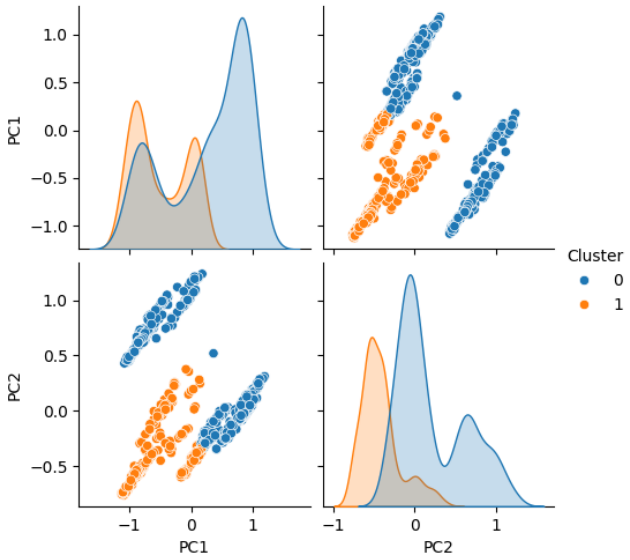


Figure 488: Clusters complete euclidean

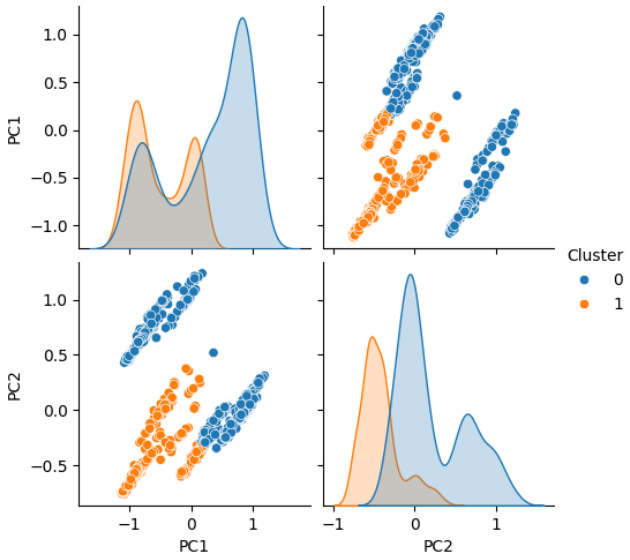


Figure 489: Clusters complete manhattan

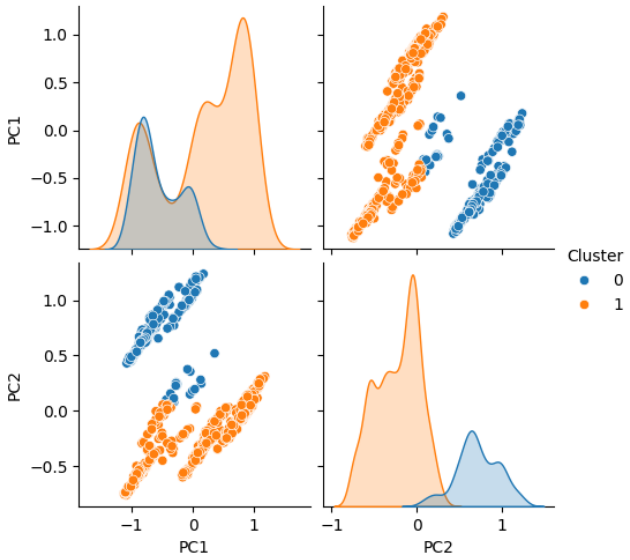


Figure 490: Clusters single correlation

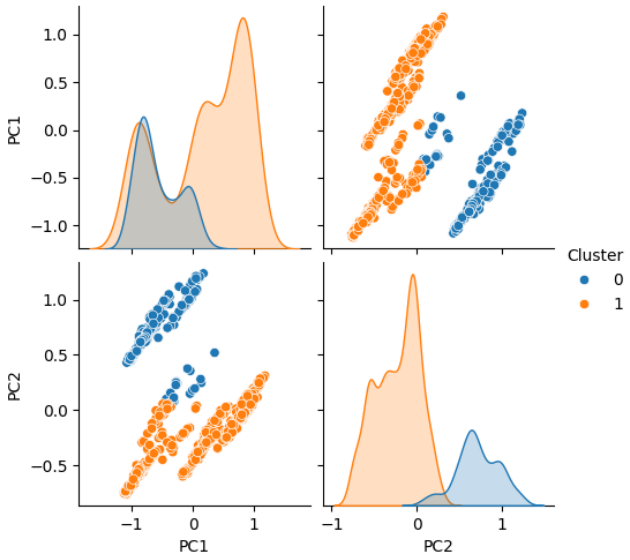


Figure 491: Clusters single cosine

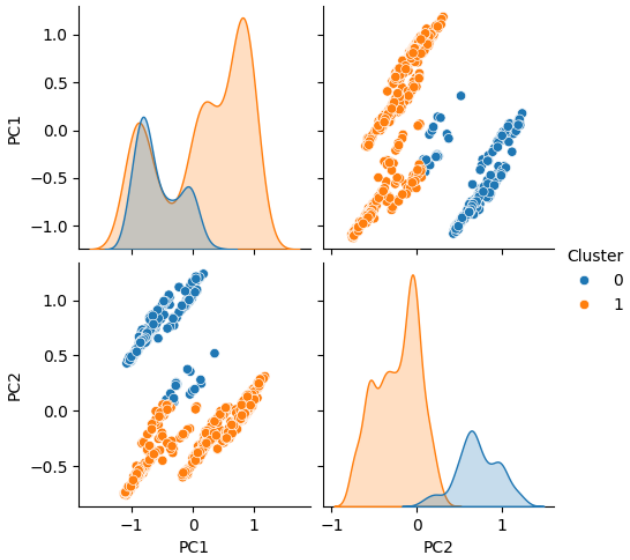


Figure 492: Clusters single euclidean

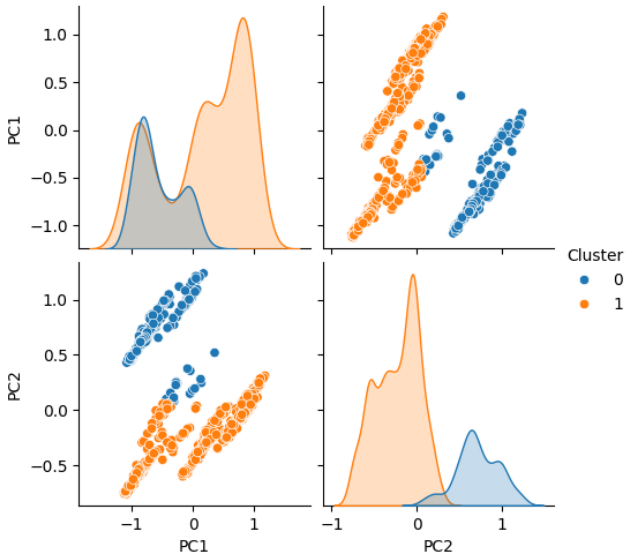


Figure 493: Clusters single manhattan

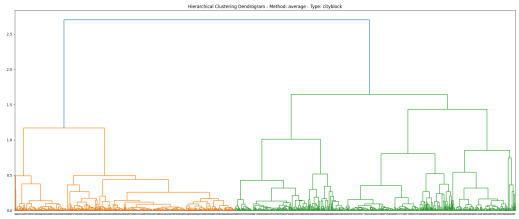


Figure 494: Dendrogram average cityblock

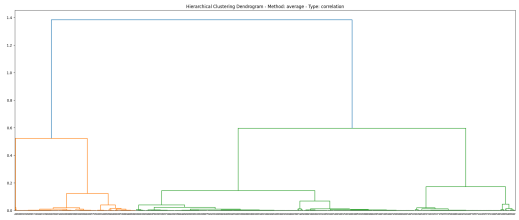


Figure 495: Dendrogram average correlation

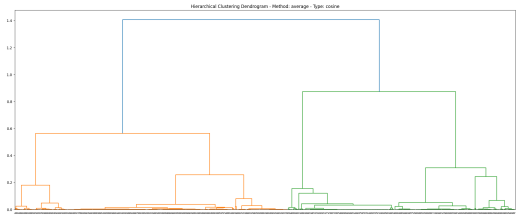


Figure 496: Dendrogram average cosine

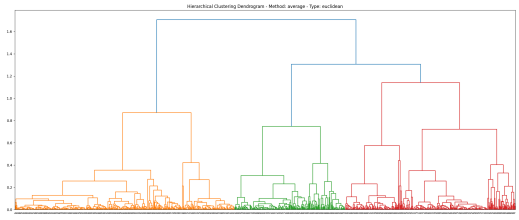


Figure 497: Dendrogram average euclidean

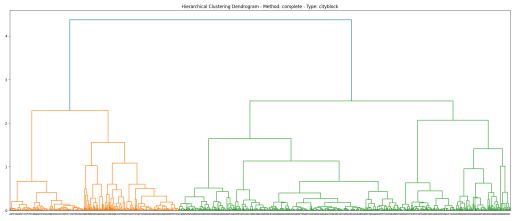


Figure 498: Dendrogram complete cityblock

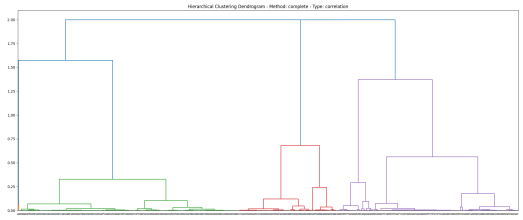


Figure 499: Dendrogram complete correlation

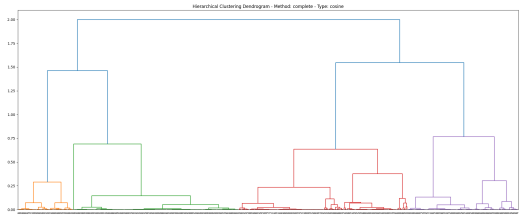


Figure 500: Dendrogram complete cosine

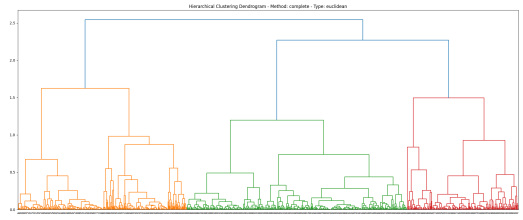


Figure 501: Dendrogram complete euclidean

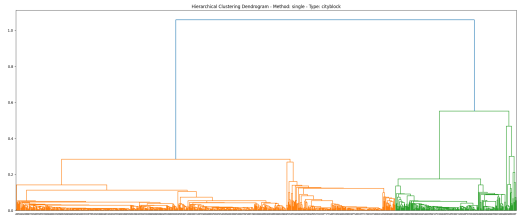


Figure 502: Dendrogram single cityblock

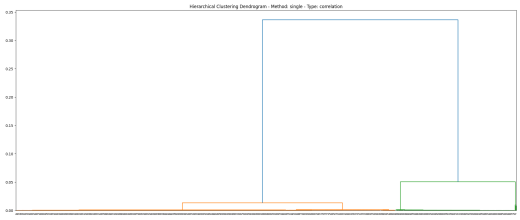


Figure 503: Dendrogram single correlation

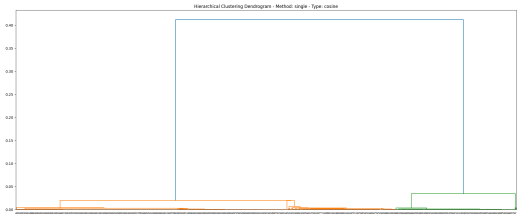


Figure 504: Dendrogram single cosine

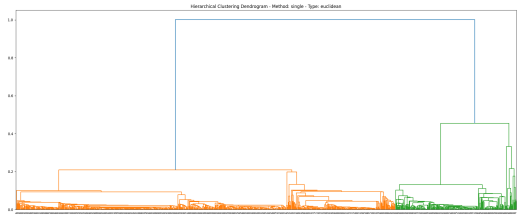


Figure 505: Dendrogram single euclidean

7.2.16 Scenario 8_S

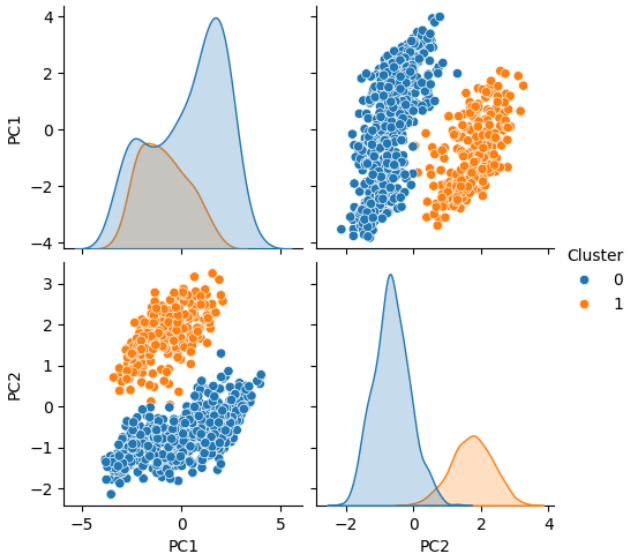


Figure 506: Clusters average correlation

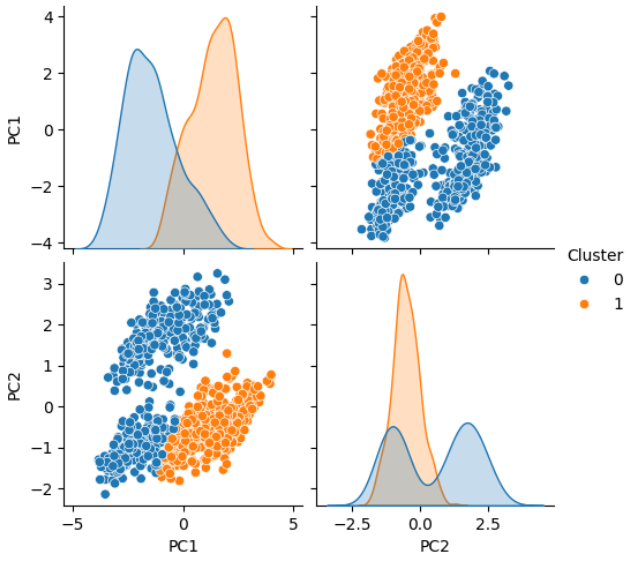


Figure 507: Clusters average cosine

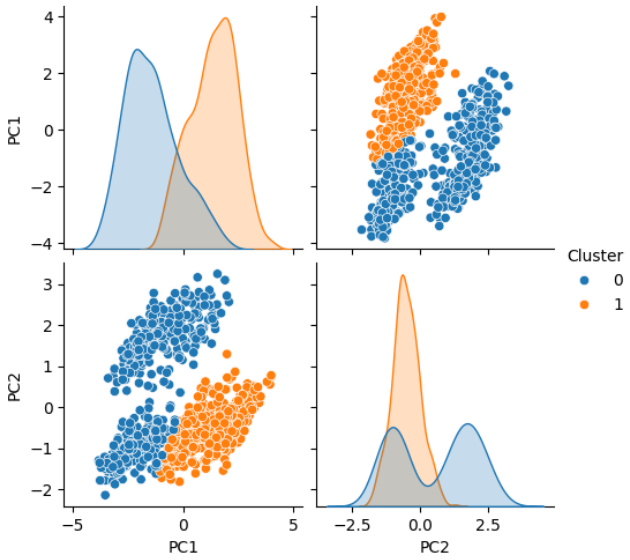


Figure 508: Clusters average euclidean

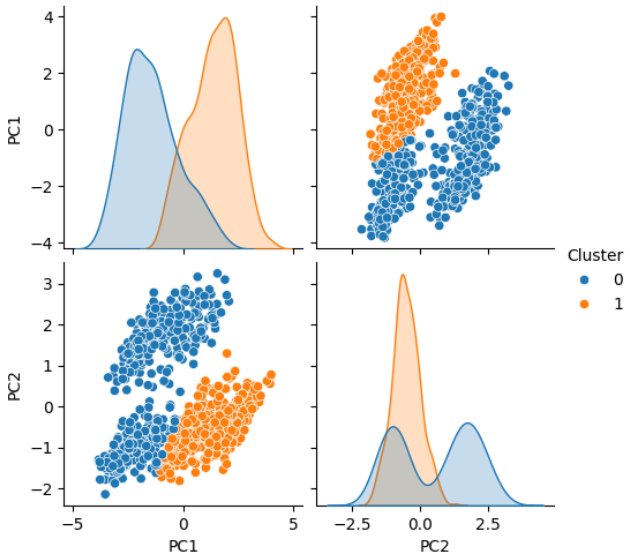


Figure 509: Clusters average manhattan

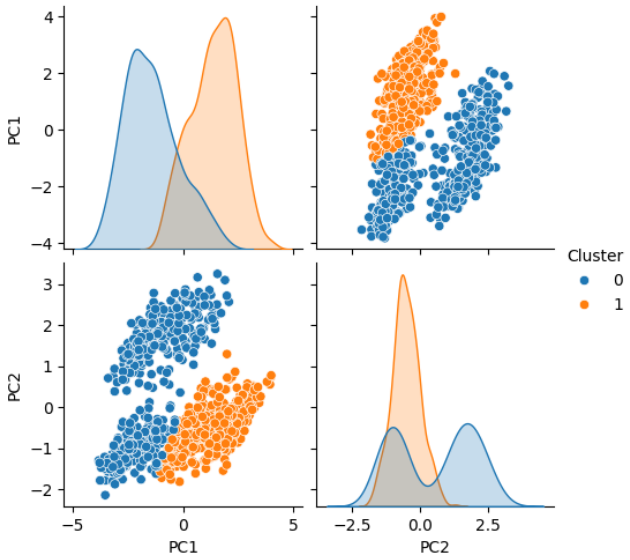


Figure 510: Clusters complete correlation

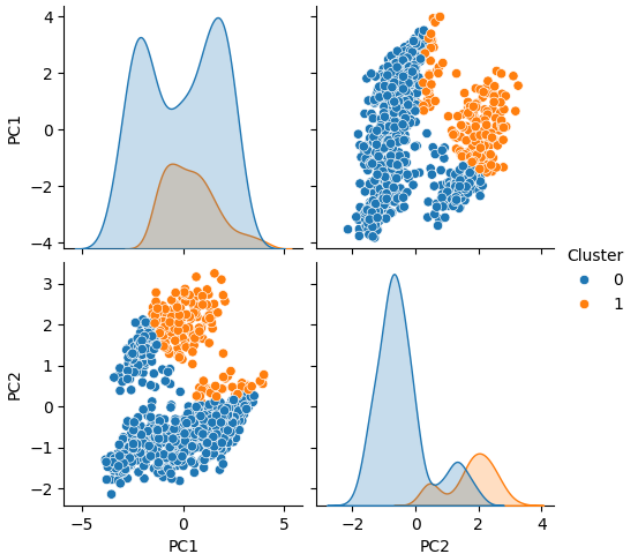


Figure 511: Clusters complete cosine

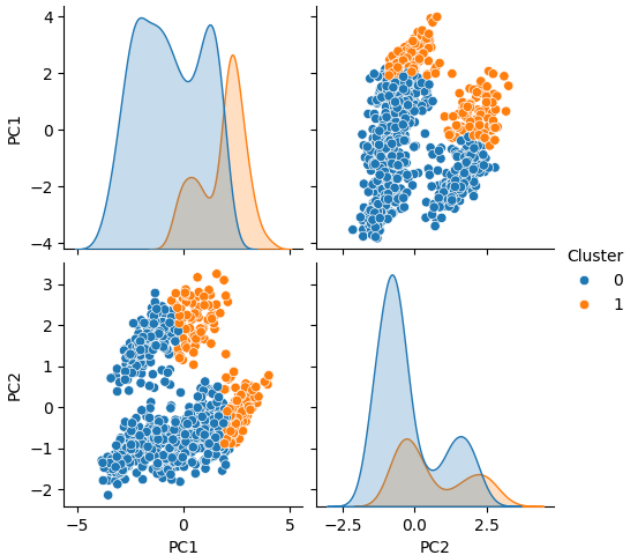


Figure 512: Clusters complete euclidean

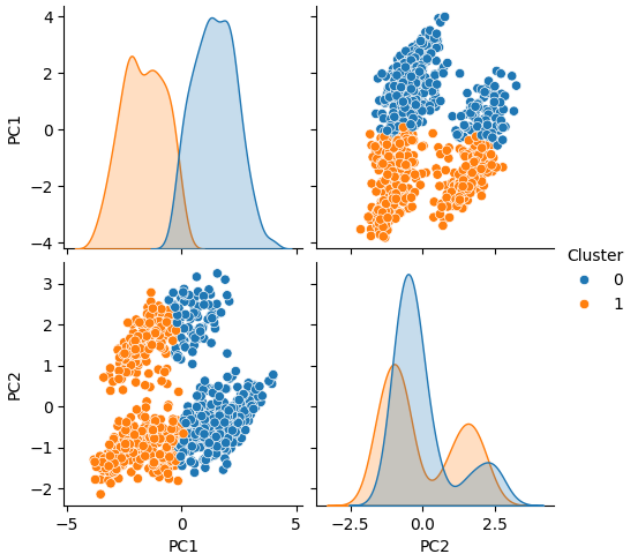


Figure 513: Clusters complete manhattan

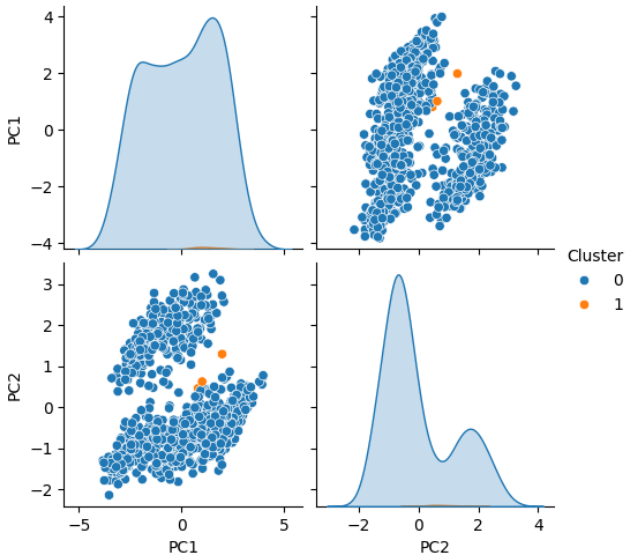


Figure 514: Clusters single correlation

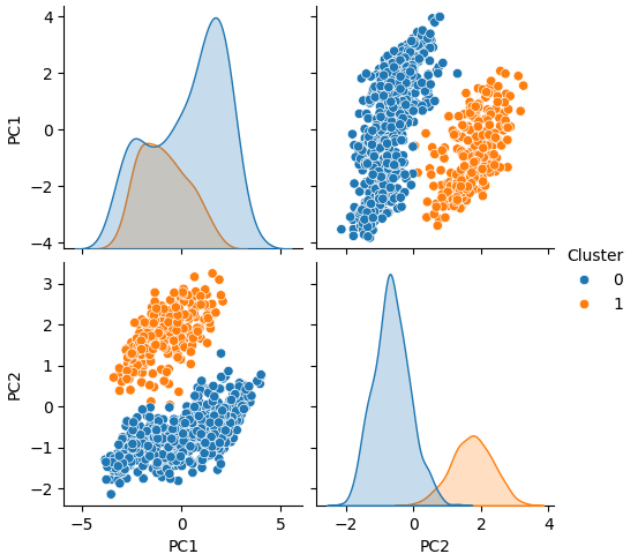


Figure 515: Clusters single cosine

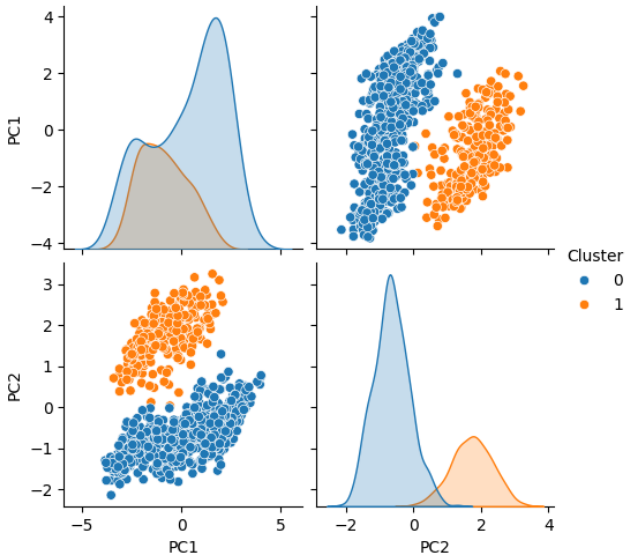


Figure 516: Clusters single euclidean

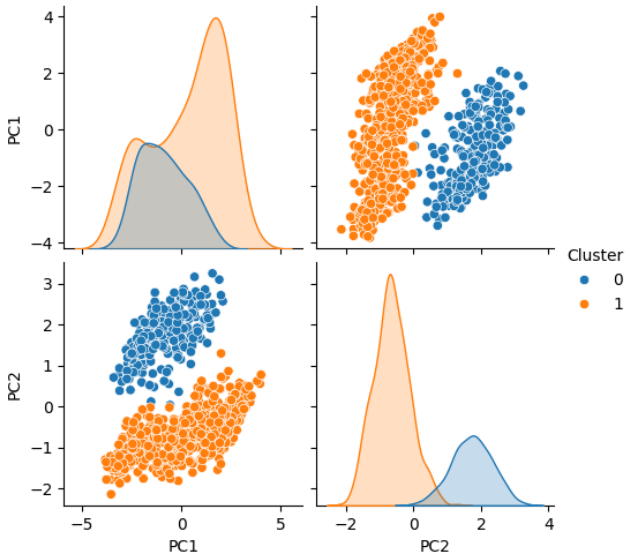


Figure 517: Clusters single manhattan

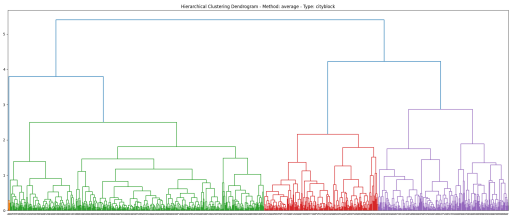


Figure 518: Dendrogram average cityblock

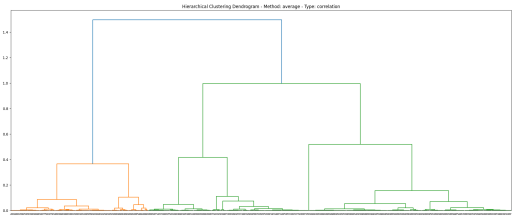


Figure 519: Dendrogram average correlation

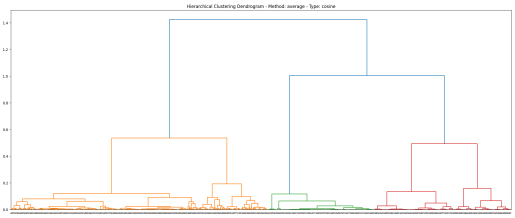


Figure 520: Dendrogram average cosine

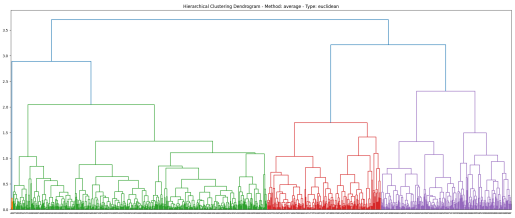


Figure 521: Dendrogram average euclidean

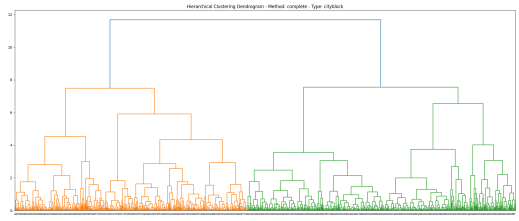


Figure 522: Dendrogram complete cityblock

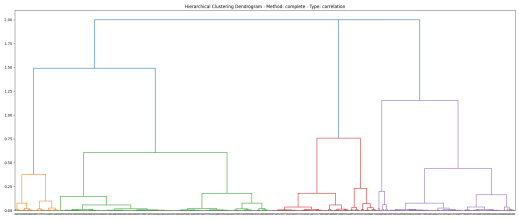


Figure 523: Dendrogram complete correlation

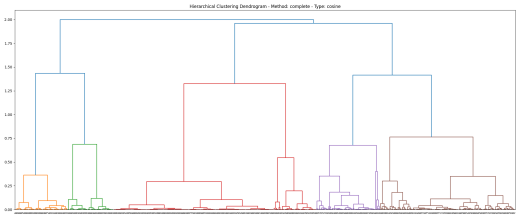


Figure 524: Dendrogram complete cosine

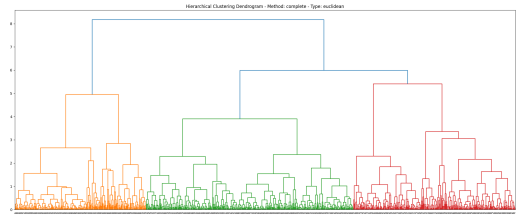


Figure 525: Dendrogram complete euclidean

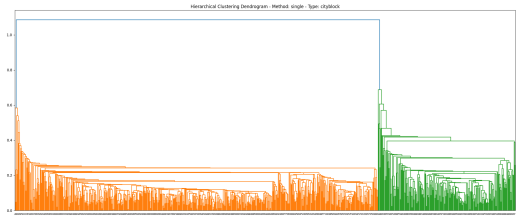


Figure 526: Dendrogram single cityblock

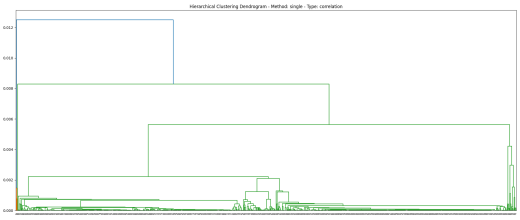


Figure 527: Dendrogram single correlation

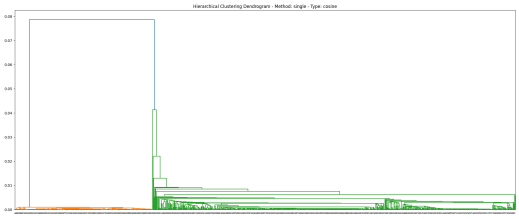


Figure 528: Dendrogram single cosine

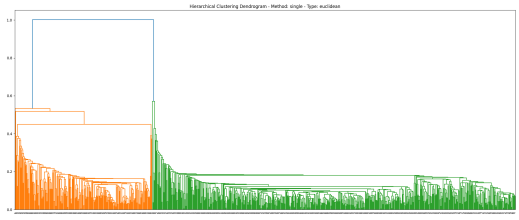


Figure 529: Dendrogram single euclidean

7.3 Alignment with PCA and Classifier Results

7.3.1 Accuracy with PCA 2 dimensions

Table 6: Scenario 1_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2179
complete	manhattan	0.2179
complete	cosine	0.5893
complete	correlation	0.2669
average	euclidean	0.7331
average	manhattan	0.7331
average	cosine	0.7331
average	correlation	0.7331
single	euclidean	0.2669
single	manhattan	0.2669
single	cosine	0.2669
single	correlation	0.2691

Table 7: Scenario 1_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.6656
complete	manhattan	0.6656
complete	cosine	0.2996
complete	correlation	0.2527
average	euclidean	0.2407
average	manhattan	0.7593
average	cosine	0.8192
average	correlation	0.8192
single	euclidean	0.4455
single	manhattan	0.4455
single	cosine	0.4455
single	correlation	0.2516

Table 8: Scenario 2_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2669
complete	manhattan	0.2669
complete	cosine	0.4521
complete	correlation	0.7277
average	euclidean	0.7277
average	manhattan	0.7277
average	cosine	0.7331
average	correlation	0.7298
single	euclidean	0.7298
single	manhattan	0.7298
single	cosine	0.2702
single	correlation	0.5381

Table 9: Scenario 2_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2168
complete	manhattan	0.2168
complete	cosine	0.7941
complete	correlation	0.8050
average	euclidean	0.5458
average	manhattan	0.5458
average	cosine	0.2190
average	correlation	0.6786
single	euclidean	0.4455
single	manhattan	0.4455
single	cosine	0.4455
single	correlation	0.2081

Table 10: Scenario 3_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2251
complete	manhattan	0.2251
complete	cosine	0.7707
complete	correlation	0.3775
average	euclidean	0.7778
average	manhattan	0.2222
average	cosine	0.2222
average	correlation	0.2222
single	euclidean	0.2222
single	manhattan	0.2222
single	cosine	0.2222
single	correlation	0.2222

Table 11: Scenario 3_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.4274
complete	manhattan	0.4274
complete	cosine	0.8319
complete	correlation	0.1895
average	euclidean	0.5342
average	manhattan	0.2009
average	cosine	0.8162
average	correlation	0.8177
single	euclidean	0.5385
single	manhattan	0.5385
single	cosine	0.5385
single	correlation	0.8120

Table 12: Scenario 4_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.7037
complete	manhattan	0.7037
complete	cosine	0.7342
complete	correlation	0.4935
average	euclidean	0.4935
average	manhattan	0.4935
average	cosine	0.7309
average	correlation	0.7386
single	euclidean	0.7386
single	manhattan	0.7386
single	cosine	0.7386
single	correlation	0.4466

Table 13: Scenario 4_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.3355
complete	manhattan	0.3355
complete	cosine	0.6765
complete	correlation	0.7789
average	euclidean	0.4444
average	manhattan	0.4521
average	cosine	0.7843
average	correlation	0.7081
single	euclidean	0.4477
single	manhattan	0.4477
single	cosine	0.4477
single	correlation	0.5316

Table 14: Scenario 5_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2647
complete	manhattan	0.2647
complete	cosine	0.3834
complete	correlation	0.7974
average	euclidean	0.7974
average	manhattan	0.7974
average	cosine	0.7974
average	correlation	0.3834
single	euclidean	0.6166
single	manhattan	0.6166
single	cosine	0.3834
single	correlation	0.4477

Table 15: Scenario 5_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.5970
complete	manhattan	0.5970
complete	cosine	0.7756
complete	correlation	0.7418
average	euclidean	0.4510
average	manhattan	0.4510
average	cosine	0.8094
average	correlation	0.1950
single	euclidean	0.4499
single	manhattan	0.4499
single	cosine	0.4499
single	correlation	0.4510

Table 16: Scenario 6_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2810
complete	manhattan	0.2810
complete	cosine	0.7832
complete	correlation	0.7647
average	euclidean	0.7647
average	manhattan	0.7647
average	cosine	0.7669
average	correlation	0.3834
single	euclidean	0.6166
single	manhattan	0.6166
single	cosine	0.6166
single	correlation	0.6166

Table 17: Scenario 6_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.1808
complete	manhattan	0.7429
complete	cosine	0.5871
complete	correlation	0.2200
average	euclidean	0.7810
average	manhattan	0.7810
average	cosine	0.7843
average	correlation	0.7854
single	euclidean	0.2146
single	manhattan	0.2146
single	cosine	0.7854
single	correlation	0.4488

Table 18: Scenario 7_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.7792
complete	manhattan	0.7792
complete	cosine	0.4160
complete	correlation	0.7792
average	euclidean	0.7792
average	manhattan	0.7792
average	cosine	0.7792
average	correlation	0.4160
single	euclidean	0.5840
single	manhattan	0.5840
single	cosine	0.4160
single	correlation	0.5356

Table 19: Scenario 7_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2179
complete	manhattan	0.2179
complete	cosine	0.8134
complete	correlation	0.7835
average	euclidean	0.8120
average	manhattan	0.8120
average	cosine	0.8120
average	correlation	0.7835
single	euclidean	0.7835
single	manhattan	0.7835
single	cosine	0.2165
single	correlation	0.5385

Table 20: Scenario 8_N Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.2680
complete	manhattan	0.2680
complete	cosine	0.7963
complete	correlation	0.6830
average	euclidean	0.6830
average	manhattan	0.6830
average	cosine	0.7647
average	correlation	0.3856
single	euclidean	0.6144
single	manhattan	0.6144
single	cosine	0.6144
single	correlation	0.6144

Table 21: Scenario 8_S Clustering Accuracy Results

Method	Type	Accuracy
complete	euclidean	0.6416
complete	manhattan	0.1732
complete	cosine	0.5
complete	correlation	0.7799
average	euclidean	0.7799
average	manhattan	0.7799
average	cosine	0.7799
average	correlation	0.3769
single	euclidean	0.3769
single	manhattan	0.6230
single	cosine	0.3769
single	correlation	0.4498

8 Results and Discussion

8.1 Summary of Model Performances

The study involved evaluating multiple classification models, namely **Naive Bayes**, **SVM**, **KNN**, and **Decision Tree**, across a variety of preprocessing scenarios and optimization methods. The models were compared based on their classification accuracy and fine-tuned hyperparameters. Below is a summary of the findings:

1. **Performance Across Scenarios**:
   - Each model's performance was measured across scenarios that differed in data preprocessing techniques, such as handling outliers, scaling methods (MinMax and Standard), and encoding approaches (Label Encoding and One-Hot Encoding).
   - Scenario 6, where outliers were not removed but errors were fixed, consistently provided a good balance for several models, especially Naive Bayes and Decision Tree.
2. **Optimization Methods**:
   - **Grid Search** provided a comprehensive search of hyperparameter combinations, leading to consistent but computationally expensive results.
   - **Random Search** achieved comparable accuracy with fewer computational resources, making it efficient for high-dimensional hyperparameter spaces.
   - **Optuna** performed exceptionally well, dynamically exploring the hyperparameter space and yielding the highest accuracy for several models.
3. **Best Model Performances**:
   - The **SVM KNN** model optimized with Optuna demonstrated the highest classification accuracy (~ 0.90), with Scenario 1.
4. **Cluster Analysis**:
   - The hierarchical clustering analysis using dendrograms provided insights into the natural grouping of data points. This aligned well with the PCA-based visualizations, showing distinct separations in high-performing scenarios.

8.2 Challenges Faced and Solutions

1. Data Quality Issues **Challenge:** The dataset contained anomalies, such as zero values for cholesterol and negative values for *Oldpeak*, which are biologically implausible. **Solution:**

- Implemented a `fix_blunder` function to replace zero values in numerical features (e.g., cholesterol) with their mean.
- Applied absolute values to rectify negative *Oldpeak* values, ensuring consistency in data representation.

2. Handling Outliers **Challenge:** Numerous outliers were identified in features like age, cholesterol, and blood pressure, which could skew model training. **Solution:**

- Developed an IQR-based method to detect and remove outliers.
- For specific scenarios, replaced outliers with the feature mean to retain data size while minimizing skew.

3. Preprocessing and Scaling Variations **Challenge:** Determining the optimal preprocessing approach (e.g., scaling, encoding) for different models was non-trivial. **Solution:**

- Created multiple preprocessing scenarios, testing MinMax Scaling, Standard Scaling, and label/one-hot encoding combinations.
- Evaluated model performances under all scenarios to identify the most effective preprocessing pipelines for each model.

4. Hyperparameter Optimization **Challenge:** Exhaustively searching the hyperparameter space for optimal configurations was computationally intensive. **Solution:**

- Used Grid Search and Randomized Search for simpler models.
- Leveraged `Optuna`, a Bayesian optimization library, to dynamically explore hyperparameter space, significantly reducing computation time while maintaining high accuracy.

5. Cluster Validation **Challenge:** Validating hierarchical clustering against ground truth labels (e.g., *HeartDisease*) was difficult due to the unsupervised nature of clustering. **Solution:**

- Applied hierarchical clustering with various proximity measures (e.g., Euclidean, Manhattan) and linkage methods (e.g., single, complete).
- Calculated accuracy scores between cluster assignments and ground truth labels to assess clustering alignment.

6. Visualization Challenges **Challenge:** High-dimensional data made visualization and interpretation challenging, particularly for clustering and classification boundaries. **Solution:**

- Applied PCA to reduce dimensions, allowing for 2D/3D scatter plots and dendrograms to visualize data separations.
- Created clear visual outputs (e.g., pair plots, confusion matrices) for enhanced interpretability.

7. Model Comparisons **Challenge:** Comparing models across varied preprocessing scenarios and optimization methods required consistent and detailed metrics. **Solution:**

- Designed a unified evaluation framework to calculate and log metrics (accuracy, precision, recall, F1 score) for each model and scenario.
- Plotted accuracy comparisons across models and methods for clear performance benchmarking.

8. Computational Constraints **Challenge:** Large datasets and multiple scenarios led to significant computational overhead. **Solution:**

- Utilized parallel processing for hyperparameter tuning (e.g., cross-validation with `n_jobs=-1`).
- Implemented efficient data processing pipelines and scenario-specific optimizations to reduce computation time.

9. Scenario Alignment with Ground Truth **Challenge:** Ensuring that the preprocessing and clustering results align with the actual *HeartDisease* labels. **Solution:**

- Evaluated the alignment of clusters and ground truth through accuracy and confusion matrices.
- Optimized scenarios iteratively to refine clustering and classification outcomes.

By addressing these challenges systematically, the study ensured robust preprocessing, efficient model training, and reliable clustering analyses. These solutions contributed to achieving high accuracy across multiple scenarios and optimization methods.

8.3 Key Insights

1. Data Characteristics and Preprocessing

- The dataset exhibited several inconsistencies, such as implausible values (e.g., zero cholesterol, negative Oldpeak) and the presence of outliers. These were systematically addressed using techniques like replacing zeros with means, applying absolute values, and removing outliers based on the IQR method.
- A variety of preprocessing pipelines, including label encoding, one-hot encoding, and scaling (MinMax and Standard Scaling), were explored. Each preprocessing method demonstrated varying impacts on the performance of clustering and classification models, highlighting the importance of data preparation in machine learning workflows.

2. Clustering Analysis

- Hierarchical clustering with diverse proximity measures (Euclidean, Manhattan, Cosine, Correlation) and linkage methods (Single, Complete, Average) revealed distinct data groupings.
- The dendrogram analysis provided insights into natural clusters within the dataset, some of which aligned with the ground truth labels (HeartDisease). This validated the dataset's inherent separability for certain preprocessing and clustering scenarios.

3. Dimensionality Reduction (PCA)

- PCA effectively reduced the dataset's dimensions to 2D/3D, enabling visualization of data separations. Key patterns emerged, with clusters showing partial alignment with "HeartDisease" labels, particularly under scenarios with outlier removal or corrected data.

4. Model Performance and Comparisons

- **Naive Bayes**: Performed well in scenarios with minimal preprocessing, suggesting its robustness to simpler feature representations.
- **SVM**: Achieved high accuracy with fine-tuned hyperparameters, excelling in scenarios with well-scaled and clean data.
- **KNN**: Showed sensitivity to the choice of neighbors and distance metrics, performing better with outlier removal and scaling.
- **Decision Trees**: Demonstrated flexibility in capturing complex patterns, especially after handling outliers and blunders.

5. Impact of Hyperparameter Optimization

- Bayesian Optimization (Optuna) consistently outperformed traditional methods (Grid Search, Randomized Search) in finding optimal hyperparameters, especially for models like SVM and Decision Trees.
- Optuna reduced computation time while maintaining or improving model performance, underscoring the efficiency of adaptive search methods.

6. Validation Through Metrics

- Cross-validation across scenarios revealed high consistency in accuracy, precision, recall, and F1 scores, particularly for SVM and KNN models.
- Confusion matrices highlighted areas where models (Naive Bayes and Decision Trees) struggled, such as distinguishing borderline cases in imbalanced clusters.

7. Insights from Visualization

- Visual tools like dendrograms, scatter plots, and confusion matrices provided a deeper understanding of data separations, model behaviors, and misclassification patterns.
- PCA visualizations demonstrated clear separations in some scenarios, correlating strongly with "HeartDisease" labels.

8. General Observations

- Data preprocessing (e.g., outlier handling, scaling) played a crucial role in model and clustering performance.
- Clustering insights aligned partially with classification results, affirming the dataset's structure and separability.
- Combining domain knowledge with systematic preprocessing and optimization led to significant performance improvements across models.

These insights underline the critical role of preprocessing, model selection, and hyperparameter tuning in extracting meaningful patterns and achieving high performance in classification and clustering tasks.

9 Conclusion

The Heart Failure Classification project represents a comprehensive application of machine learning to address a critical healthcare challenge. Using the Heart Failure Prediction Dataset, the study meticulously explored clinical and demographic attributes to predict heart disease presence. By incorporating essential preprocessing steps such as outlier removal, missing value handling, and data scaling, the project ensured a robust foundation for building reliable predictive models. These steps addressed data quality issues while aligning the dataset with the requirements of various machine learning algorithms.

The implementation of classical models, including Naive Bayes, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Decision Trees, highlighted the effectiveness of these techniques in solving classification problems. Extensive hyperparameter tuning through methods like Grid Search, Random Search, and Bayesian Optimization (Optuna) further refined the models' performance. The analysis of results revealed significant differences in model behaviors under varying preprocessing conditions, demonstrating the importance of tailoring methods to data characteristics.

Principal Component Analysis (PCA) and hierarchical clustering were key components of the study, offering valuable insights into data patterns and relationships. PCA enabled effective dimensionality reduction and visualization, providing clarity on feature separability in a 2D space. The dendrogram analysis further enhanced understanding by uncovering natural groupings within the data, which is particularly relevant for patient stratification and targeted interventions.

The project's structured approach ensured transparency and reproducibility, with all preprocessing scenarios, models, and results comprehensively documented. Model performance was evaluated using metrics like precision, recall, F1-score, and confusion matrices, allowing detailed comparisons and actionable insights. Naive Bayes and Decision Trees emerged as robust performers across preprocessing scenarios, while SVM and KNN demonstrated sensitivity to data scaling, emphasizing the critical role of preprocessing.

In conclusion, this project successfully leveraged machine learning to predict heart failure risks, delivering actionable insights and a scalable methodology. The integration of diverse techniques, from preprocessing to advanced model optimization, highlights its contribution to advancing healthcare analytics. This work not only underscores the transformative potential of machine learning in healthcare but also provides a robust framework for tackling similar challenges in other domains.