Exercise Report: Units 1–8

Joan Fernández Navarro

STATISTICAL LEARNING
FOR DATA SCIENCE

Master's Degree in Computational Engineering and Industrial Mathematics

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Contents

List of Figures

List of Tables

List of Algorithms

The code developed to complete all the proposed exercises for this report is available in the online GitHub repository: https://github.com/JoanFer030/statistical_exercises

1 Unit 1. Ensemble Learning

1.1 Bias and Variance

In the field of machine learning, the trade-off between bias and variance is one of the fundamental concepts for understanding the behavior of predictive models. This section presents an experimental analysis of this phenomenon through the evaluation of polynomial regression models applied to data with nonlinear patterns, with the aim of illustrating how different levels of complexity model the relationship between variables differently.

- **Bias** Represents the inherent error that occurs when an overly simplified model attempts to approximate complex relationships present in the data. A model with high bias tends to make systematic errors, as it fails to adequately capture the patterns existing in the training data. This behavior is known as underfitting. In these cases, the simplicity of the model prevents it from reflecting the actual complexity, resulting in inaccurate predictions and poor generalizations.
- **Variance** Measures the sensitivity of the model to small variations in the training data. A model with high variance tends to overfit the patterns in the training set, including noise or outliers. This behavior is known as overfitting, in which the model achieves a very low error rate in the training data but performs poorly when evaluated on new data. In these cases, the excessive complexity of the model causes a loss of generalization ability.

To illustrate the concepts of bias and variance, an experiment was developed using synthetic data with a nonlinear relationship between the independent variable X and the dependent variable y. The data were generated according to the following expressions:

$$\begin{aligned} X &= [-2, 2]; \quad \text{divided into 100 equal subintervals} \\ y &= sin(2 \cdot X) + \varepsilon; \quad \varepsilon \sim N(0, 0.4^2) \end{aligned}$$

where ε represents a Gaussian noise term with a mean of zero and a standard deviation of 0.4. This dataset allows us to observe how different levels of model complexity affect fit and generalization.
Next, three polynomial regression models with different degrees of complexity (degrees 1, 3, and 20) were trained for the purpose of comparatively analysing the effects of underfitting and overfitting as a function of the degree of the polynomial.
As shown in Figure 1, the first-degree polynomial model (linear regression) exhibits a high level of underfitting. This model fails to capture any patterns in the training data, showing virtually no generalization ability. Numerically, it achieves a mean square error (MSE) of 0.6526 in the training set and 0.6287 in the validation set, which highlights its excessive simplicity and limited ability to represent the nonlinear relationship between

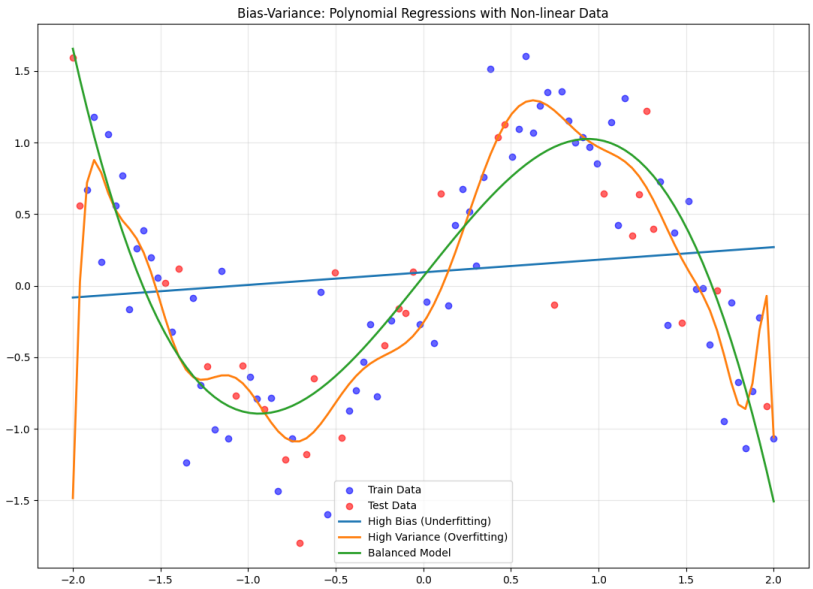


Figure 1: Three polynomial regressions of different degrees with non-linear data ($y = sin(2X) + \varepsilon$).

the variables.

At the opposite extreme, the 30th degree polynomial model shows a clear case of overfitting. As can be seen in the figure, the model reproduces the behavior of the training set almost exactly, even following the data pattern point by point. Numerically, it has an MSE of 0.1092 in training (notably low), but an MSE of 0.5336 in validation, which is considerably higher. This difference reflects the excessive complexity captured by the model, which is why it ends up adjusting to the noise in the training set and losing virtually all of its generalization capacity.

Finally, the third-degree polynomial model achieves an adequate balance between bias and variance. This model manages to appropriately represent the general trend of the training data without overfitting to it, which translates into good performance on the validation set. The MSE values are relatively similar in both cases, 0.1679 for training and 0.2236 for validation, confirming its ability to correctly generalize the behavior of the data.

1.2 Ensemble Learning

Ensemble methods have become essential tools for improving the performance and robustness of predictive models. These approaches combine multiple individual models to produce a final prediction that is typically more accurate and stable than that of any single model. In this section, an experiment is conducted using several individual models and one ensemble model, with the goal of analysing how model diversity affects prediction accuracy and generalization capability.

- **Bagging (Bootstrap Aggregating)** is an ensemble technique designed to reduce variance and improve model stability. It works by generating multiple versions of a training dataset through bootstrap resampling (randomly sampling with

replacement) and training a separate model on each of these samples. The final prediction is obtained by aggregating the outputs of all models, typically through averaging in regression tasks or majority voting in classification tasks. Bagging reduces overfitting by smoothing out the fluctuations caused by high-variance models, such as decision trees.

- **Boosting** unlike bagging, it aims to reduce bias by sequentially training models in a way that each new model focuses on correcting the errors made by the previous ones. In boosting, models are added iteratively, and the predictions are combined through a weighted sum, where the weights depend on each model's accuracy. Boosting transforms a collection of weak learners, that are models that perform only slightly better than random guessing, into a strong learner with significantly improved predictive performance.

1.2.1 Model Diversity

Model diversity refers to the degree of difference between the individual models that make up an ensemble. In other words, it measures how independently the models learn and make their predictions. Theoretically, high diversity implies that the models commit different types of error on the data, which allows the ensemble to combine their outputs in a complementary way. Therefore, the main goal of a high model diversity is to reduce the overall generalization error.
To analyse this effect, a synthetic dataset was generated consisting of 800 samples distributed in two distinct clusters. This setup allows us to evaluate how various models (both individual and ensemble) capture the underlying structure of the data and how model diversity contributes to improving predictive accuracy and generalization.
The models used in this experiment were a Logistic Regression, a Decision Tree, a Support Vector Machine (SVM) with an RBF kernel, and a k-Nearest Neighbors (kNN) classifier with $k = 5$ neighbors. Each of these models was trained and evaluated individually. Subsequently, they were combined into an ensemble model using a majority voting scheme to analyse how the aggregation of diverse classifiers affects predictive performance.

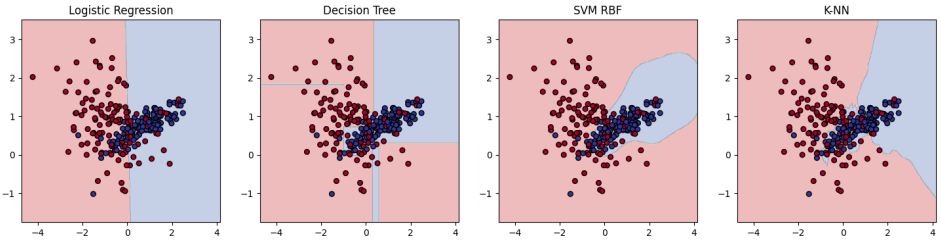


Figure 2: Decision boundaries of the individual models showing varying complexity and adaptation to the data.

As shown in Figure 2, the decision boundaries of the individual models vary considerably. In the cases of Logistic Regression and Decision Tree, the boundaries are more

linear, which is consistent with the inherent characteristics of these algorithms. Consequently, both models achieved lower accuracy scores, 0.7750 and 0.8125, respectively. On the other hand, the SVM and kNN models produced decision boundaries that more closely follow the non-linear shape of each cluster, resulting in higher accuracies of 0.8167 for SVM and 0.8333 for kNN.

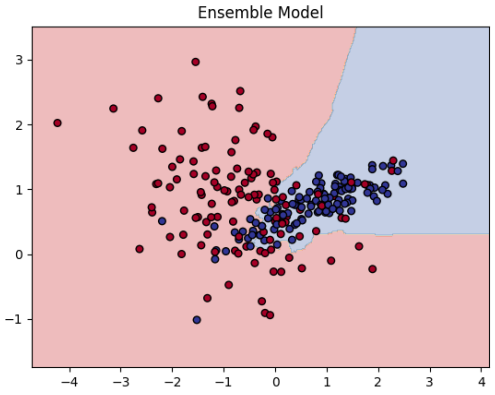


Figure 3: Decision boundary of the ensemble model illustrating improved separation and generalization.

Figure 3 illustrates the decision boundary of the ensemble model, which forms a more precise contour around both clusters, particularly around the second (blue) cluster. This boundary successfully captures the underlying structure of the data with greater accuracy, as reflected by the ensemble's overall performance, achieving an accuracy of 0.8392. These results demonstrate how combining diverse models through majority voting can enhance generalization and yield more robust predictions compared to individual classifiers.

1.3 Voting Mechanisms

Within bagging-based ensemble models, different voting strategies can be employed to combine the predictions of individual classifiers. These voting methods determine how the final class label is assigned based on the outputs of all base models. In this section, three common approaches are analyzed: hard voting, soft voting, and weighted voting.

- **Hard voting** assigns the final prediction based on the majority class selected by the individual models. Each model contributes one vote, and the class receiving the highest number of votes becomes the ensemble's output. This method is simple and effective when all models have similar performance, but it may be suboptimal if some classifiers are significantly more accurate than others.

- **Soft voting** takes into account the predicted class probabilities rather than just the final class labels. The probabilities produced by each model for every class are averaged, and the class with the highest aggregated probability is chosen. This approach often leads to better performance, as it considers the confidence of each prediction rather than treating all votes equally.

- **Weighted voting** extends the idea of soft voting by assigning different weights to the models according to their performance, typically based on validation accuracy or another evaluation metric. Models with higher accuracy have greater influence on the final decision.

Following the same experimental framework described earlier, an ensemble was trained using the same four classifiers, but this time applying each of the three voting strategies independently. A new synthetic dataset was generated for this purpose, allowing for a comparative analysis of how different voting methods affect classification accuracy and generalization performance.

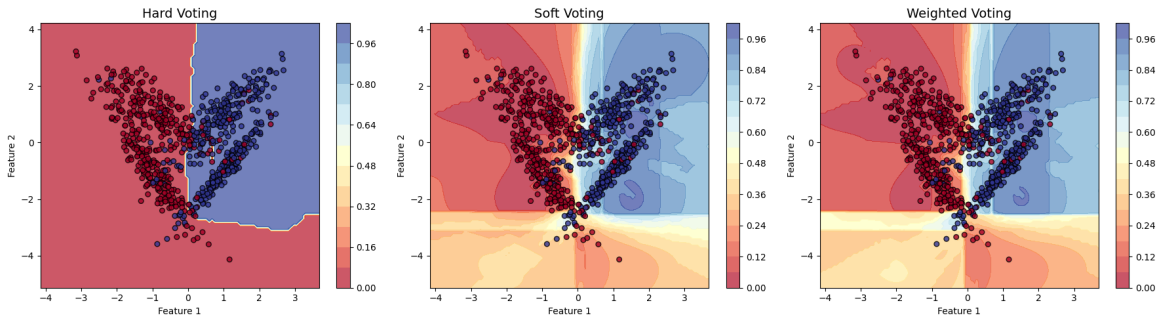


Figure 4: Comparison of decision boundaries using hard, soft, and weighted voting ensembles, illustrating differences between binary and probabilistic cluster assignments.

As shown in Figure 4, the decision boundaries exhibit a similar overall pattern across all four voting mechanisms, forming a quadrant in the upper right region where samples of the second cluster (blue) are classified. In the case of hard voting, the cluster assignments are strictly binary: each sample either belongs to the cluster or it does not, resulting in decision boundaries drawn as a single line.
In contrast, the decision boundaries produced by soft and weighted voting are probabilistic rather than fixed. Each sample is assigned a probability of belonging to one cluster or another, with the probability decreasing as the sample moves away from the cluster until reaching the boundary where both classes have equal probability. This probabilistic representation allows for a smoother separation between clusters.
Numerical results confirm the benefits of these probabilistic boundaries. The ensemble using soft voting achieved an accuracy of 0.9137, and the weighted voting ensemble reached 0.9150, both outperforming the hard voting ensemble, which obtained an accuracy of 0.9012. These results demonstrate that incorporating probability information and model weighting can enhance the ensemble's predictive performance and generalization.

1.4 Stacking

Stacking is another ensemble learning technique that aims to combine the strengths of diverse base models by learning how to best integrate their predictions. The main difference between stacking and the other two techniques (bagging and boosting) lies in the way the final prediction is made.

In stacking, multiple base models (also called level-0 models) are trained independently on the same dataset. The predictions of these base models are then used as input features for a higher-level model, known as the meta-learner or level-1 model. The meta-learner is responsible for learning the optimal way to combine the outputs of the base models, effectively correcting their individual weaknesses and producing a final prediction that is typically more accurate than any single base model.
Unlike bagging and boosting, which usually combine models using simple aggregation rules such as averaging or weighted voting, stacking leverages the predictive power of a machine learning model to determine how to merge the predictions.
Following the experimental framework described in the previous sections, a new synthetic dataset was generated. Using this data set, four base learners, the same models used previously, were trained independently. A Logistic Regression model was then employed as the meta-learner, which takes the predictions of the base learners as input features to produce the final ensemble predictions. This approach allows the meta-learner to learn how to optimally combine the outputs of the diverse base models.

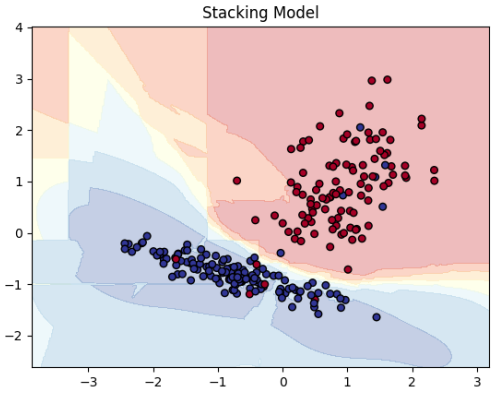


Figure 5: Decision boundaries of the stacking ensemble using Logistic Regression as the meta-learner, showing probabilistic cluster assignments.

As shown in Figure 5, one of the most notable characteristics of the stacking ensemble is the presence of probabilistic decision boundaries. By using Logistic Regression as the meta-learner, the model produces soft, rather than binary, cluster assignments, resulting in more nuanced and flexible boundaries. The ensemble effectively captures the underlying patterns of each cluster, as illustrated in the figure. This improved representation is also reflected in the numerical results, with the stacking model achieving an accuracy of 0.9458, outperforming the individual base learners and previous ensemble methods.

2 Unit 2. Bagging Techniques - RF and ET

2.1 Bootstrap

In this section, we introduce and analyze the bootstrap resampling method, a non-parametric technique widely used in ensemble learning. The concept of bootstrapping plays a central role, particularly in methods based on bagging (Bootstrap Aggregating). Bootstrapping in this context refers to the process of creating multiple training datasets by sampling with replacement from the original dataset. Each of these samples, called bootstrap samples, typically has the same size as the original data, but contains repeated observations and leaves out others.
The main goal of this resampling procedure is to introduce diversity among the base models. Since each model in the ensemble is trained on a slightly different subset of the data, their individual errors tend to be less correlated. When the predictions of all base learners are later aggregated, this diversity reduces the overall variance of the model and leads to better generalization performance. In essence, bootstrapping allows bagging to stabilize models that are highly sensitive to small changes in the training data, without increasing bias significantly.

Central Limit Theorem (CLT)
The Central Limit Theorem (CLT) establish that the sampling distribution of the sample mean tends to follow a normal distribution as the sample size increases, regardless of the shape of the original population distribution, provided that the samples are independent and identically distributed. Mathematically, if $X_1, X_2, ..., X_n$ are random samples from a population with mean μ and finite variance σ^2, then the standardized sample mean

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

approaches a standard normal distribution as $n \to \infty$.
In the context of bootstrapping, the CLT provides the theoretical foundation for estimating the sampling distribution of a statistic when only one sample is available. By repeatedly resampling (with replacement) from the observed data and computing the statistic of interest (such as the mean or standard deviation) for each resample, the resulting distribution of these bootstrap estimates approximates the true sampling distribution. Thanks to the CLT, as the number of bootstrap samples grows, this empirical distribution converges toward a normal distribution centered around the population parameter.
This connection allows bootstrapping to be used as a practical method for assessing the variability, bias, and confidence intervals of estimators and also in ensemble methods like bagging, where resampling is used to stabilize model predictions.

To illustrate these concepts empirically, we generated a large synthetic population and performed a bootstrap experiment. The population was sampled from a Normal distribution with mean 30 and standard deviation 5, consisting of $50,000$ observations:

$$X_i \sim N(30, 5^2); \quad i = 1, ..., 50000$$

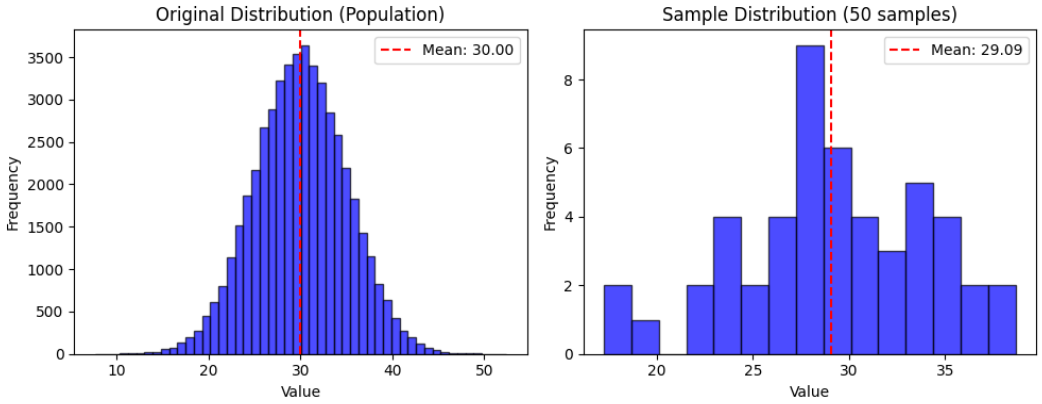


Figure 6: Distribution comparison between the full population and a small sample of 50 elements.

As shown in Figure 6, the generated population follows a clear normal distribution centered around a mean value of 30. However, when a small sample of only 50 elements is drawn from this population, the distribution of the sample no longer appears perfectly normal, and its mean slightly deviates from the population mean, taking a value of 29.09. This illustrates how limited samples can introduce sampling variability and deviate from the theoretical properties of the underlying distribution, a concept that bootstrap resampling seeks to mitigate by repeatedly drawing new samples from the available data, as will be shown in the following example.

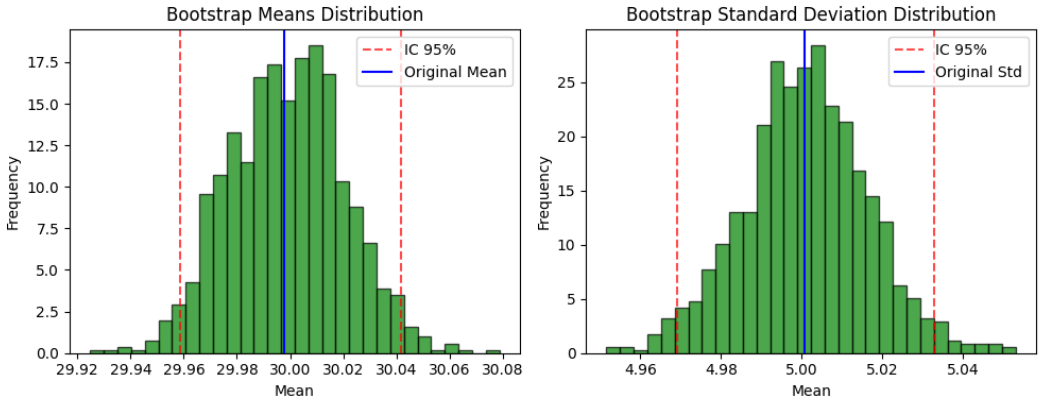


Figure 7: Bootstrap distributions of the sample mean and standard deviation across 1,000 resamples, showing the 95% confidence intervals.

Following with the same population and to ilustrate the functioning of the bootstrapping process, 1,000 samples of the same size as the original population were generated with replacement. For each of these samples, the mean and standard deviation were computed. As shown in Figure 7, the distributions of both statistics approximately follow a normal shape, which aligns with the theoretical expectations derived from the Central Limit Theorem.

When calculating the 95% confidence intervals, the results show that for the mean, the interval ranges from 29.96 to 30.04, clearly including the true population mean.

A similar pattern can be observed for the standard deviation, where the confidence interval extends from 4.97 to 5.03, once again covering the population's true standard deviation. These results confirm that the bootstrap successfully estimates the underlying population parameters, even when relying on repeated resampling from a single dataset.

2.2 Random Forest and Extra Trees

In this section, two tree-based ensemble methods are explored and compared; they are commonly referred to as randomization-based approaches. These methods aim to improve predictive performance by combining multiple weak learners into a single, stronger model. By introducing controlled randomness and iterative learning strategies, they seek to reduce both bias and variance while enhancing the model's ability to generalize to unseen data. These two models are Random Forest and Extremly Randomized Trees, also known as Extra Trees. An experiment is conducted to evaluate and compare their performance on a synthetic dataset, with the objective of analyzing how the level of randomness introduced during training affects model accuracy and stability.

- **Random Forest** is an ensemble learning method that constructs multiple decision trees using the bagging approach. Each tree is trained on a random bootstrap sample of the data, and at each split, a random subset of features is considered. This controlled randomness reduces correlation among trees and prevents overfitting, leading to improved generalization. The final prediction is obtained through averaging (for regression) or majority voting (for classification).
- **Extra Trees (Extremely Randomized Trees)** extend the idea of Random Forests by introducing an additional source of randomness. Instead of searching for the best possible split at each node, Extra Trees select split thresholds randomly for each feature and then choose the best among these random splits. This results in higher variance reduction and faster training times, often yielding slightly better generalization performance on noisy data. However, Extra Trees can sometimes be less stable in small or highly imbalanced datasets.

Following the same experimental setup as in the previous unit, both models were trained on a newly generated synthetic dataset, with $1,000$ samples, and evaluated using accuracy as the main performance metric. The comparison allows us to assess how the degree of randomness in tree construction influences the ensemble's ability to generalize and capture complex patterns in the data.

The results obtained from both models are highly comparable in terms of predictive performance, with accuracy values exceeding 0.8. Specifically, the Random Forest achieved an accuracy of 0.8233, while the Extra Trees model obtained 0.8167. The most remarkable difference between the two methods lies in their training efficiency. As theoretically expected, Extra Trees exhibited a considerably faster training process, finishing in 0.191 seconds compared to 0.414 seconds for Random Forest, an improvement of

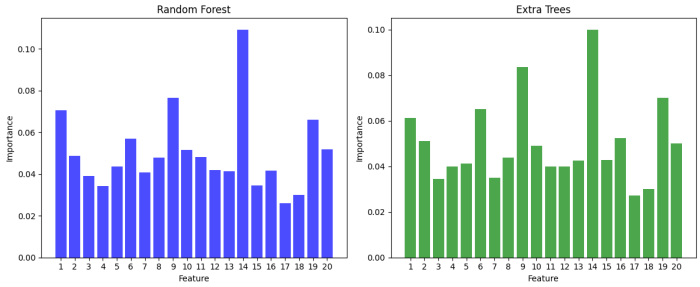


Figure 8: Feature importance comparison between Random Forest and Extra Trees models, showing similar relevance patterns.

approximately 116.8%. This highlights the computational advantage introduced by the additional randomization in the Extra Trees algorithm.
To complement the performance analysis, we further examined the feature importance assigned by both models. As illustrated in Figure 8, the two algorithms display a very similar distribution of feature relevance, even at the quantitative level. In both cases, feature 14 stands out with a relative importance of around 0.1. Additionally, features 1, 6, 19 and 20 consistently emerge as the most influential variables in both models, suggesting that both ensembles capture comparable structural patterns within the data set.

2.3 Hiperparameter Optimization with Cross-Validadtion (RF)

In this section, we focus on the optimization of hyperparameters, a critical step in building machine learning models. Hyperparameters are configuration settings that govern the learning process itself, such as the depth of a decision tree, the regularization strength in a logistic regression, or the number of neighbors in a k-Nearest Neighbors classifier. Unlike model parameters, which are learned directly from the training data, hyperparameters must be set prior to training. Optimizing these values is essential because they can significantly affect model performance, generalization, and the risk of overfitting or underfitting.
A common approach to evaluating hyperparameter choices is cross-validation, which involves partitioning the training data into multiple folds, training the model on a subset of folds, and validating it on the remaining fold(s). This process is repeated so that each fold serves as a validation set, providing a robust estimate of model performance. Additionally, early stopping is often employed to prevent overfitting during training, by monitoring validation performance and halting the learning process once improvements become negligible.
Two widely used strategies for hyperparameter search are described below:

- **Grid Search:** This method exhaustively evaluates all possible combinations of a predefined set of hyperparameter values. Although it guarantees that the global optimum within the grid is tested, it can be computationally expensive, especially for large search spaces.

- **Random Search:** Instead of testing all combinations, random search samples hyperparameter configurations randomly from specified distributions. This method often finds near-optimal solutions faster than grid search, particularly when only a few hyperparameters significantly influence model performance.

To illustrate these concepts, an experiment was conducted using a synthetic classification dataset composed of two classes. Both grid search and random search were applied to optimize hyperparameters of a Random Forest model, employing cross-validation to evaluate performance and early stopping to prevent overfitting during training. This setup allows a direct comparison of the two optimization methods in terms of efficiency and predictive performance.
After performing hyperparameter optimization with both techniques, several clear observations can be made. First, the number of combinations explored by each method differs substantially: grid search evaluated a total of 243 combinations, whereas random search tested only 50 configurations. Consistently with theoretical expectations, the search time required to find the best combination is also considerably lower for random search, taking 33.24 seconds compared to 78.99 seconds for grid search. These results might suggest that predictive performance (accuracy) would be substantially worse for random search; however, this is not the case. Grid search achieved an average accuracy of 0.7436 across the three folds for its best hyperparameter combination, while random search, despite requiring less than half the search time, reached an accuracy of 0.7371, differing only by a few hundredths.
When comparing both methods to a baseline model, as illustrated in Figure 9, the relative improvements over the baseline are modest, and there is no significant difference between the two optimization approaches. These findings highlight the practical advantage of random search for hyperparameter optimization, as it achieves near-equivalent predictive performance with substantially lower computational and temporal cost.

2.4 Project - Customer Classification

The objective of this project is to predict if a customer is going to generate profit over their lifetime, in order to determine whether the company should accept them as client or not. To achieve this, one Random Forest model will be developed, classification model to predict one of two categories, accept or deny.

2.4.1 Data Processing

The first step in the data preprocessing phase involved removing all columns containing information about the insurance policy itself, since these variables explicitly indicate whether a customer will generate profit or not, potentially introducing data leakage into the model. After this cleaning step, all rows with a value of 2 in the 'SINCO' column were removed. This filtering process resulted in a dataset composed of $51,618$ rows and 196 columns, ready for modeling.
Next, the target variable was defined to identify customers who generate profit for the company. This variable was derived from the 'BMA_corregido' field, where customers with values greater than zero were labeled as profitable.

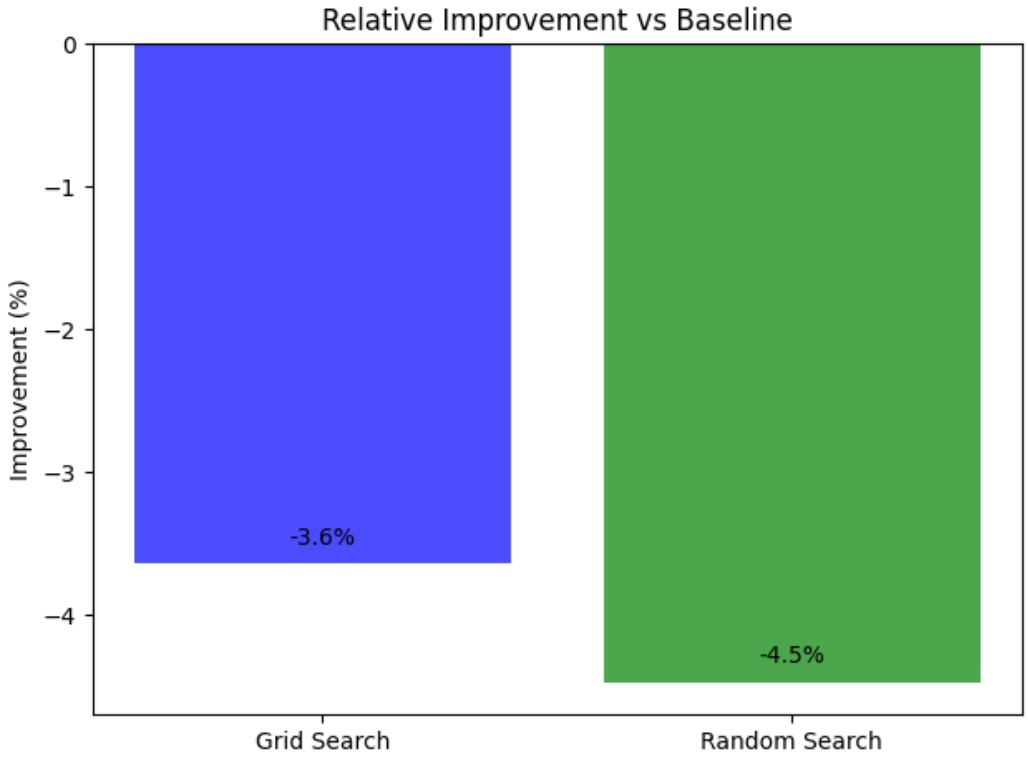


Figure 9: Relative improvement in accuracy over the baseline Random Forest model for both grid search and random search hyperparameter optimization methods.

As illustrated in Figure 10, the range of profits is considerably narrower than that of losses, and the number of profitable customers is also substantially smaller. Specifically, these customers represent only 19.3% of the total population, yet they generate an average profit of 17.63 euros per customer, resulting in a total profit of approximately 909, 881 euros, equivalent to about 7% of the total potential profit.

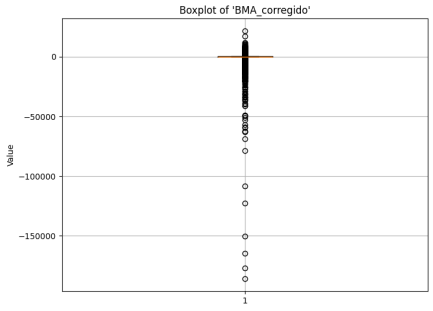


Figure 10: Distribution of customer profitability based on the 'BMA_corregido' variable. Profitable customers (values > 0) represent 19.3% of the total.

2.4.2 Model Training

For the training, validation, and testing process, the dataset was divided into three stratified subsets to preserve the original class distribution across all partitions. The resulting sample sizes were as follows: Training set with $32,518$ samples, Validation set with $15,486$ samples, and Test set with $3,614$ samples.

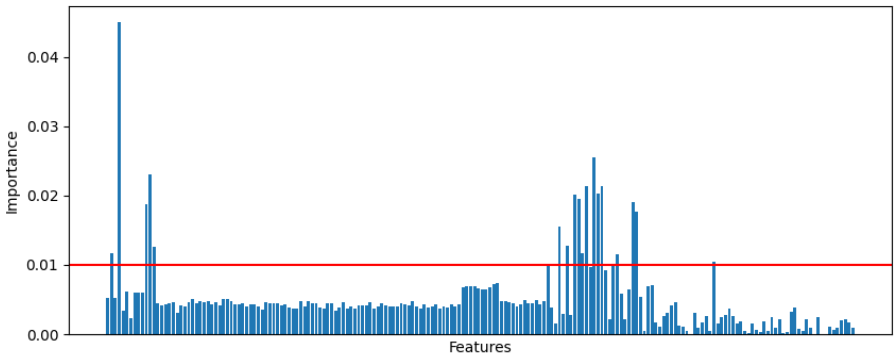


Figure 11: Feature importance distribution for the baseline model trained with all variables.

As a first step, a baseline model was trained using all available variables. This initial model served both as a benchmark for comparison and as a tool to evaluate feature importance. As shown in Figure 11, the relative importance of the features varies considerably, some variables exhibit minimal or even negligible contribution to the model's predictive performance. Moreover, the distribution of importance values is highly uneven: while most features contribute only marginally, a small subset accounts for a substantial proportion of the total importance.
Given this observation, the decision was made to isolate and retain only the most influential features—specifically, those within the top 5% of importance scores, resulting in a refined subset of nine key variables. A new model was then trained using only these selected features to assess whether predictive performance could be maintained or even improved with a more parsimonious representation of the data.

2.4.3 Results and Discussion

The model trained using all available variables required 31.02 seconds of training time and achieved a ROC AUC of 0.684 on the test set. In contrast, the model trained with only the nine most important features required just 11 seconds to train, while obtaining a slightly lower but comparable ROC AUC of 0.667. Given this small trade-off in predictive performance and the substantial reduction in computational cost, the nine-variable model was selected as the final version.
When examining the Figure 12 that contains the confusion matrix reveals that the model successfully classifies most observations belonging to Class 0, although it shows a tendency to over-classify in this dominant category. As a result, some instances that truly belong to Class 1 are incorrectly assigned to Class 0.

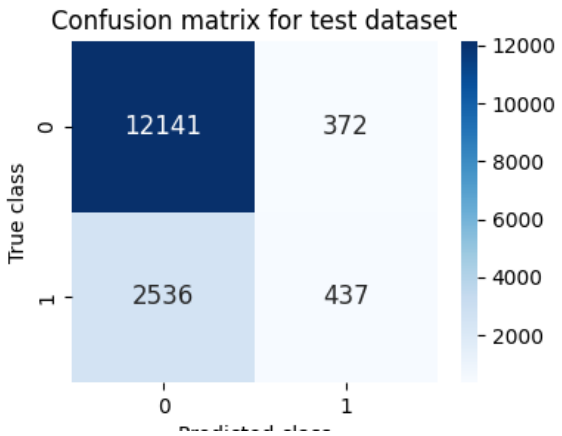


Figure 12: Confusion matrix of the final nine-variable model for customer classification.

From an economic standpoint, the model demonstrates strong practical performance: the average profit per client rises to 50.84 euros, and the overall profit margin increases to 21%, representing a 14-percentage-point improvement compared to the baseline scenario. The only noteworthy misclassification corresponds to a client erroneously accepted by the model, resulting in a loss of nearly $200,000$ euros.
In summary, despite minor classification imbalances, the final model effectively identifies profitable clients, enhancing both the average profit per client and the overall profitability margin.

3 Unit 3. Boosting Methods - Ada Gradient XGB

3.1 Boosting Algorithms Comparison

In this section, we focus on boosting, a powerful ensemble learning technique designed to improve predictive performance by sequentially combining multiple weak learners. Unlike bagging, which trains models independently on bootstrap samples, boosting trains each model in sequence, with each subsequent learner focusing on the errors made by the previous ones. By iteratively correcting the mistakes of prior models, boosting reduces bias and produces a strong overall predictor, often achieving higher accuracy than individual learners or simple ensembles.

- **AdaBoost (Adaptive Boosting):** Builds an ensemble of weak learners, typically decision stumps, by iteratively reweighting the training samples. Misclassified samples receive higher weights in subsequent iterations, forcing the next learner to focus on harder examples.
- **Gradient Boosting:** Sequentially fits new models to the residual errors of the ensemble constructed so far, effectively performing gradient descent in function space. This approach allows for the optimization of arbitrary differentiable loss functions.
- **XGBoost (Extreme Gradient Boosting):** An efficient and scalable implementation of gradient boosting that includes regularization, parallel processing, and handling of missing values, providing state-of-the-art performance in many machine learning tasks.
- **LightGBM (Light Gradient Boosting Machine):** A gradient boosting framework optimized for efficiency and scalability. It uses a leaf-wise growth strategy, histogram-based splitting, and other techniques to reduce training time while maintaining high predictive performance.

To illustrate the capabilities of these methods, an experiment was conducted using a synthetic dataset of 2,000 individuals divided into two clusters. This setup allows for a detailed analysis of both training time and predictive performance (accuracy) across the different boosting algorithms, providing insights into their relative strengths and computational efficiency.
As shown in Figure 13, the predictive performance of the four boosting models is generally good. Except for AdaBoost, the differences in accuracy among Gradient Boosting, XGBoost, and LightGBM are minimal, ranging from approximately 0.78 to 0.79. In contrast, AdaBoost achieves a notably lower accuracy of 0.717, representing a difference of more than 10% compared to the other methods.
Regarding training times, a higher degree of variability is observed, with some surprising results. AdaBoost completes training in 0.522 seconds, slightly below the average of the four models (1.591 seconds). Gradient Boosting requires 1.174 seconds, which is relatively close to the mean. The most striking values are found for XGBoost, which

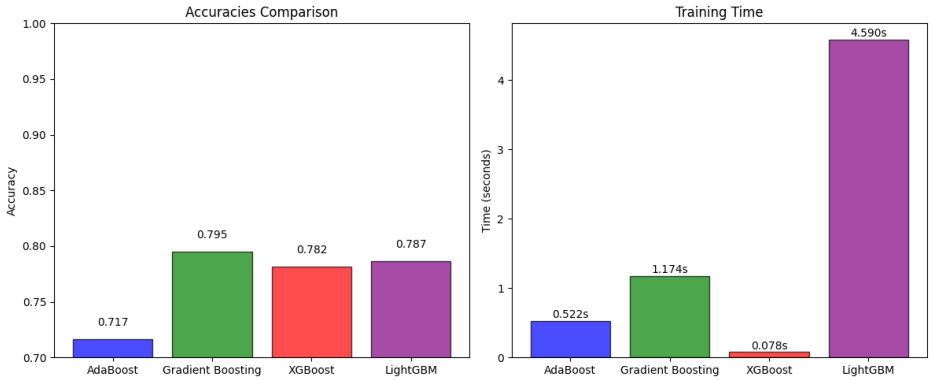


Figure 13: Accuracy and training time comparison of AdaBoost, Gradient Boosting, XGBoost, and LightGBM on a synthetic two-cluster dataset.

completes training in only 0.078 seconds — an unexpectedly short time given the complexity of the method, although this can be partially attributed to its highly optimized implementation. On the other hand, LightGBM requires more than 4 seconds to train, which is noteworthy considering that it is generally recognized for its fast training times. These observations highlight that while accuracy differences among most boosting methods are minor, training times can vary significantly depending on implementation details and algorithmic optimizations, emphasizing the importance of considering both performance and efficiency when selecting a boosting model. Taken together, these results demonstrate the remarkable superiority of XGBoost, which achieves comparable predictive performance to other boosting techniques while drastically reducing training time.

3.2 XGBoost

In this section, we focus exclusively on XGBoost, specifically we will evaluate its performance when combined with hyperparameter optimization using random search. To evaluate the effectiveness of hyperparameter tuning, random search is employed to explore different configurations of key XGBoost parameters, including the number of estimators, maximum tree depth, learning rate and subsampling ratios. Random search is particularly suitable in this context because it can efficiently explore high-dimensional hyperparameter spaces and often identifies near-optimal settings with fewer iterations compared to exhaustive grid search.

The experimental setup uses a synthetic data set specifically generated to test XGBoost under controlled conditions. This data set consists of two distinct clusters, allowing us to analyse both predictive performance (accuracy) and computational efficiency (training time).

The hyperparameter optimization process was carried out by evaluating a total of 30 parameter combinations using 3-fold cross-validation, resulting in the training of 90 models in total. Despite this extensive search, the total optimization time was only 22.42 seconds, which is remarkably low given the wide hyperparameter space explored. The best-performing configuration achieved an accuracy of 0.7576, demonstrating the

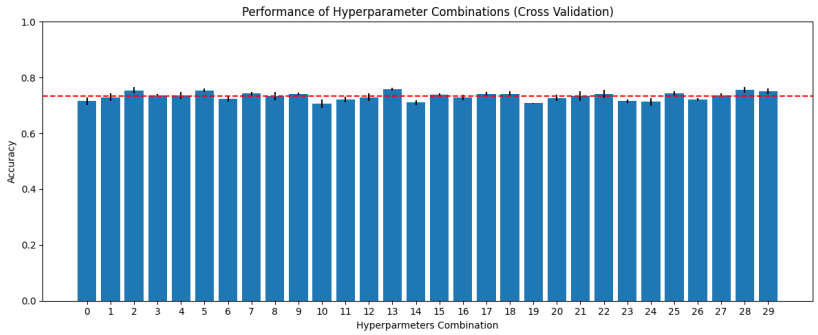


Figure 14: Distribution of accuracy scores across 30 hyperparameter combinations for XGBoost.

strong balance between efficiency and predictive performance. In addition, all tested configurations were analyzed to further assess the stability of the model. As shown in Figure 14, the performance scores are tightly clustered around the mean, indicating very limited variability across combinations. Moreover, the standard deviation across folds is minimal, confirming the high stability and robustness of XGBoost, which consistently maintains strong generalization capabilities regardless of the training subset used.

3.3 Income Classifier - XGBoost

In this section, we are going to use XGBoost in a more realistic context, we will employ the well-known "Adult" dataset from scikit-learn, also referred to as the Census Income dataset. This dataset contains information collected from the 1994 U.S. Census, the objective classification is to predict whether an individual's income exceeds \$50,000 per year based on demographic and employment attributes.
The dataset includes $48,842$ observations and 14 input features (both numerical and categorical), along with a binary target variable indicating the income class.

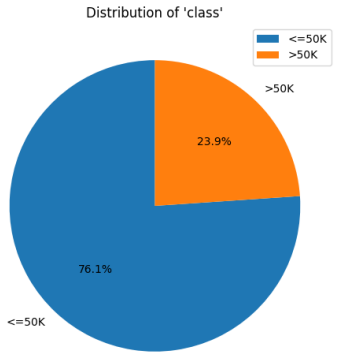


Figure 15: Class distribution of the target variable in the Adult dataset.

The first step in the analysis involves examining the class distribution of the target variable. As shown in Figure 15, the population earning less than $\$50,000$ represents

Table 1: Description of the variables included in the Adult dataset.

Variable	Type	Description
`age`	Categorical	Age of the individual.
`workclass`	Categorical	Type of employer.
`fnlwgt`	Numerical	Final sampling weight assigned to each observation.
`education`	Categorical	Highest level of education attained.
`education-num`	Numerical	Encoded numeric representation of education level.
`marital-status`	Categorical	Marital status.
`occupation`	Categorical	Type of occupation or job category.
`relationship`	Categorical	Relationship within the household.
`race`	Categorical	Race of the individual.
`sex`	Categorical	Gender of the individual.
`capital-gain`	Numerical	Monetary gain from capital investments.
`capital-loss`	Numerical	Monetary loss from capital investments.
`hours-per-week`	Categorical	Average number of working hours per week.
`native-country`	Categorical	Country of origin.
`income`	**Binary**	Indicates whether income $>$ \$50K or $\leq$ \$50K.

76.1% of the total, while only 23.9% earn more than $\$50,000$. Although this distribution indicates a certain degree of imbalance, it is not severe enough to warrant specific corrective measures. In practice, XGBoost is known to handle moderate class imbalances effectively through its internal weighting mechanisms.

In the second step, and as presented in Table 1, the dataset contains a considerable number of categorical variables. Since most machine learning algorithms—including XGBoost—require numerical input, these variables were transformed using the dummy encoding technique. This approach converts categorical variables into a series of binary columns, allowing the model to interpret them correctly without imposing ordinal relationships.

Additionally, a significant number of missing values were identified in the variables 'workclass', 'occupation', and 'native-country'. Given that all three are categorical, the mode imputation method was applied to replace missing values with the most frequent category in each feature. Finally, the variables 'capital-gain' and 'capital-loss', representing income from capital investments, were removed from the dataset due to their strong correlation with the target variable, which could otherwise lead to data leakage and an overestimation of model performance.

Model Training

For the training, validation, and testing process, the dataset was divided into three stratified subsets to preserve the original class distribution across all partitions. The resulting sample sizes were as follows: Training set with $27,351$ samples, Validation set with $14,653$ samples, and Test set with $6,838$ samples.

During training, early stopping was applied with a patience of 30 iterations to prevent overfitting by halting the optimization process once the model's performance on the

validation set stopped improving. Given that this is a binary classification problem, the logarithmic loss (log loss) was used as the evaluation metric, as it provides a continuous measure of the model's confidence in its predictions.

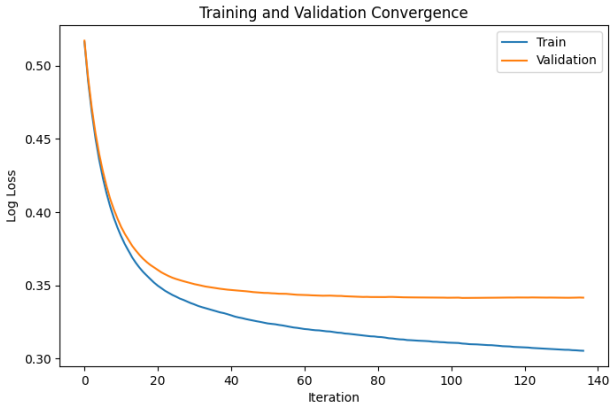


Figure 16: Convergence of the XGBoost model during training, showing the evolution of log loss across iterations for the training and validation sets.

As illustrated in Figure 16, the training process converged after 136 iterations, completing in a total of 2.8 seconds. The final results indicate a log loss of 0.30539 for the training set and 0.34166 for the validation set, reflecting a good balance between model fit and generalization capability.

Results and Discussion

The final results of the model demonstrate a high generalization capacity and predictive accuracy, achieving an accuracy of 0.8557 on the training set and 0.8443 on the test set. The minimal difference between both values indicates that the model does not suffer from overfitting, confirming a well-balanced learning process.

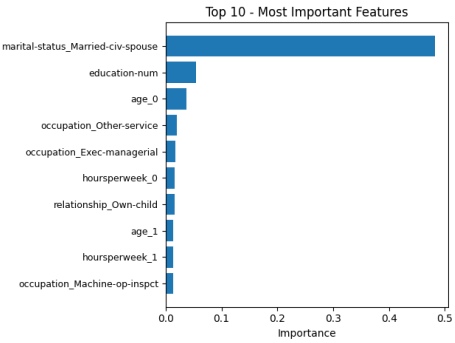


Figure 17: Feature importance derived from the trained XGBoost model.

When analyzing feature importance, as illustrated in Figure 17, it becomes evident that one variable stands out significantly from the rest: marital-status, specifically the

category married-civ-spouse. This label refers to individuals who are married to a civilian spouse, that is, not married to a member of the armed forces. Interestingly, this group exhibits a strong positive association with higher income levels, likely reflecting the socioeconomic stability associated with civilian households. The second most relevant feature is the education level, which also shows a clear positive relationship with income, confirming the model's ability to capture meaningful socioeconomic patterns in the data.

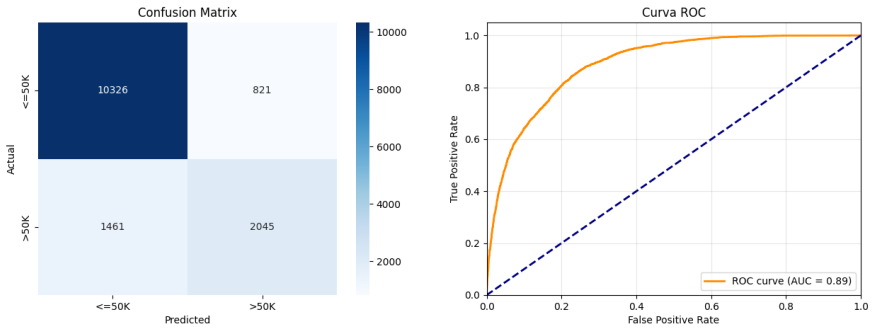


Figure 18: Model performance visualization showing the confusion matrix (left) and ROC curve (right).

From a performance perspective, the model achieves an AUC of 0.89, confirming its strong discriminative power. As shown in Figure 18, the ROC curve reveals a solid trade-off between sensitivity and specificity, while the confusion matrix indicates that most observations are correctly classified. Nevertheless, a slight tendency remains for the model to underpredict high-income individuals, classifying some of them as belonging to the lower-income group due to the moderate imbalance present in the dataset.

3.4 Project - Losses Prediction

The objective of this project is to predict the damage a hurricane might cause to a hotel in Kingston, Jamaica, in order to determine whether insurance should cover the damage, and if so, whether the payment should be partial or full. To achieve this, two XGBoost models will be developed: the first is a classification model to predict one of three categories (non-payable, partially payable, fully payable), and the second is a regression model to estimate the monetary value of the damage in dollars.

3.4.1 Data Processing

The available dataset includes various characteristics of past hurricanes affecting the region and the damaged caused to the hotel. To better reflect real-world impacts, enhanced features were generated using domain-specific knowledge. The following metrics were calculated:
These enhanced features were then crossed with the hotel damage records to build the dataset used for modeling.
Before developing the models, it is crucial to analyze both the distribution of classes and the distribution of numeric losses. The three classes were defined based on two

Table 2: Description of the custom variables created for hurricane characterization.

Variable	Description
`inverse_dist`	Inverse distance to the hotel (closer hurricanes have higher values)
`mws_weighted`	Maximum wind speed weighted by proximity
`mslp_weighted`	Minimum sea-level pressure weighted by proximity
`pressure_deficit`	Deviation of central pressure from mean sea-level pressure (1013.25 hPa)
`severity_index`	Ratio of wind intensity to distance, reflecting potential for damage
`risk_exposure`	Combined measure of wind and pressure deficits relative to distance
`wind_pressure`	Dynamic wind pressure based on wind speed
`rmw_ratio`	Relative size of the hurricane eye
`compound_impact`	Combination of severity index and wind pressure
`proximity_intensity`	Combined effect of proximity and wind intensity

thresholds: $\$15,000$ for partially payable and $\$115,000$ for fully payable damages. This categorization allows the models to distinguish between events that do not require insurance coverage, those that require partial compensation, and those that are fully payable.

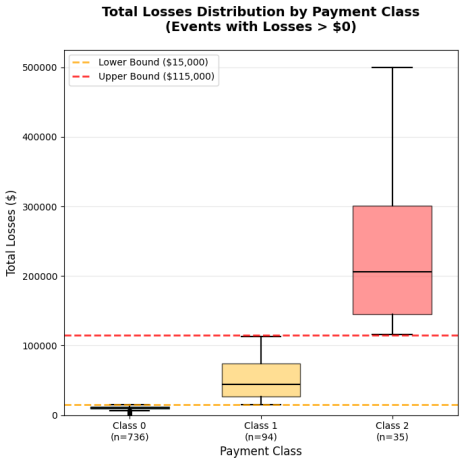


Figure 19: Distribution of hurricane damage classes for the hotel. Class 0 (Non-payable), Class 1 (Partially Payable) and Class 2 (Fully Payable).

As observed in Figure 19, the distribution of classes is highly imbalanced. The majority class, Non-payable (Class 0), contains 736 events with positive losses, representing almost all cases where no payment is needed. The losses in this class are relatively low, ranging from as little as $\$7.20$ up to $\$14,930.80$, with an average of approximately $\$10,330$ and a median around $\$10,496$. The narrow range of losses in this class indicates a well-defined set of minor events that typically do not trigger insurance claims.
The Partially Payable class (Class 1) includes 94 events with losses between $\$15,327$

and $112,655, showing significantly higher variability compared to Class 0. The average loss in this category is around $51,383, while the median is approximately $44,303. This discrepancy between the average and median reflects the presence of a few events with extremely high damages, which increases the overall variability. The distribution in this class suggests that the insurance payout may vary substantially depending on the intensity and proximity of the hurricane, making this a challenging category for classification.
Finally, the Fully Payable class (Class 2) represents the rarest events, with only 35 instances. Losses in this category are extreme, ranging from roughly $115,835 up to $500,087, with an average of $237,933 and a median of $205,758. The very high variance in this class highlights the unpredictability of catastrophic hurricanes and the potential for severe damage to the hotel. Despite the small number of events, accurately predicting this category is crucial for risk management and insurance planning.
Overall, this analysis demonstrates the significant imbalance and variability present in the dataset. Class 0 dominates the observations, while Classes 1 and 2 contain progressively fewer but more variable losses. These characteristics must be carefully considered when training both the classification and regression models, as they directly impact model performance, metric selection, and evaluation strategies.

3.4.2 Model Training

For the training, validation, and testing process, the data set was divided into three stratified subsets to preserve the original class distribution across all partitions. The resulting sample sizes were as follows: Training set with 80,354 samples, Validation set with 43,047 samples, and Test set with 20,089 samples.
During training, early stopping was applied with a patience of 50 iterations to prevent overfitting by halting the optimization process once the model's performance on the validation set stopped improving. Given that this is a multiclass classification problem, multi logarithmic loss was used as the evaluation metric, as it provides a continuous measure of the model's confidence in its predictions. In the case of the regression model the mean absolute error was used as the evaluation metric.

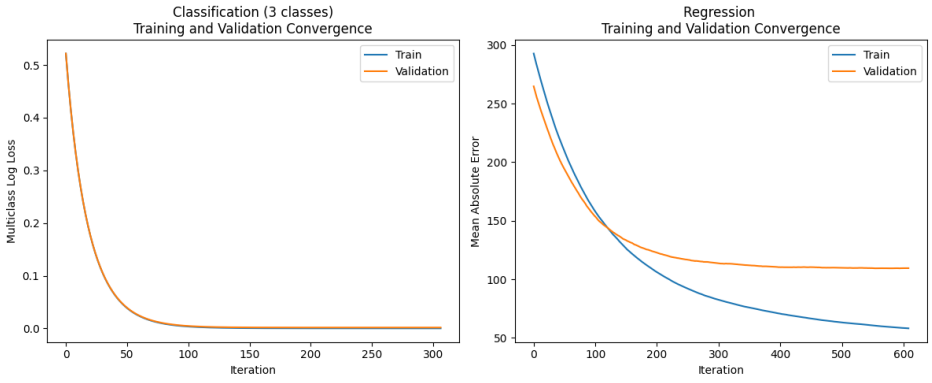


Figure 20: Convergence of the XGBoost classification and regression models.

As observed in the Figure 20 the convergence of the models differs substantially. The

classification model reaches convergence much earlier than the regression model, specifically at iteration 305 compared to iteration 607 for regression. Examining the convergence in detail, we see that for the classification model, the training and validation curves remain very close throughout the process. This indicates that the model generalizes effectively to the validation data, as confirmed by the final multi-class log-loss values: 0.00026 for training and 0.00181 for validation.
In contrast, the regression model shows a different pattern. The training and validation error curves are not as parallel. Initially, the model predicts the validation data better than the training data, and this behavior persists until approximately iteration 150. After this point, the reduction in validation error slows down considerably compared to the training error, resulting in an increasing gap between the two curves with each iteration. The final errors for the regression model are 58.1779 for training and 109.5144 for validation, reflecting the greater difficulty in accurately predicting continuous loss values and the presence of more variability in the regression task compared to the classification task.

3.4.3 Results and Discussion

The final performance of the classification model demonstrates that it is capable of making correct predictions on both the training and test sets. The model achieved a macro F1 score of 0.9883 on the training set and 0.5168 on the test set. While there is a notable difference between these values, this discrepancy can largely be attributed to the highly imbalanced number of samples across classes. Examining the F1 score per class confirms this: for Class 0, the F1 score is nearly 1, whereas for Class 1 and Class 2 the results are much lower, at 0.21 and 0.33 respectively, reflecting the challenge of accurately predicting the rare, high-damage events.
In the regression model, the differences between training and test performance are evident but less pronounced. The mean absolute error (MAE) is $60.15 on the training set and $106.94 on the test set, indicating that while the model struggles more on unseen data, it maintains reasonable generalization for continuous damage predictions.

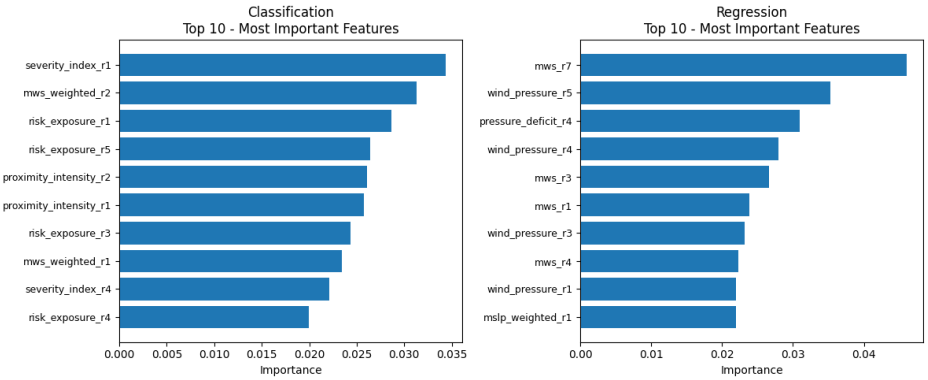


Figure 21: Feature importance derived from the trained XGBoost model.

Focusing in the Figure 21, the feature importance differ considerably between the two models. First, the varibles that each model consider most influential are almost com-

pletly different. For classification, the most important feature is the severity index at point R1, whereas for regression it is the maximum wind speed (MWS) at point R7. Despite these differences in ranking, the overall scale of importance is similar between the two models, and no single feature dominates excessively, suggesting that both models rely on a combination of multiple features to make predictions.

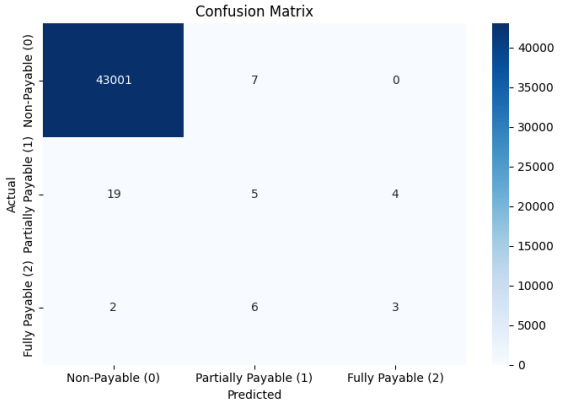


Figure 22: Confusion matrix for the classification model for losses prediction.

A closer analysis of the Figure 22, classification model predictions reveals behavior that aligns with expectations given the highly imbalanced nature of the dataset. The model tends to overwhelmingly predict the Non-payable class, which is the majority category, even in cases where this classification is not correct. This tendency is a direct consequence of the large disparity in sample sizes across the classes. While the model achieves high overall accuracy, its ability to correctly identify the rare events—Partially Payable and Fully Payable damages—is limited. In fact, only a very small number of predictions outside the majority class are made correctly, highlighting the challenges inherent in imbalanced classification tasks and emphasizing the need for careful metric selection and potentially for strategies such as class weighting or oversampling when training future models.

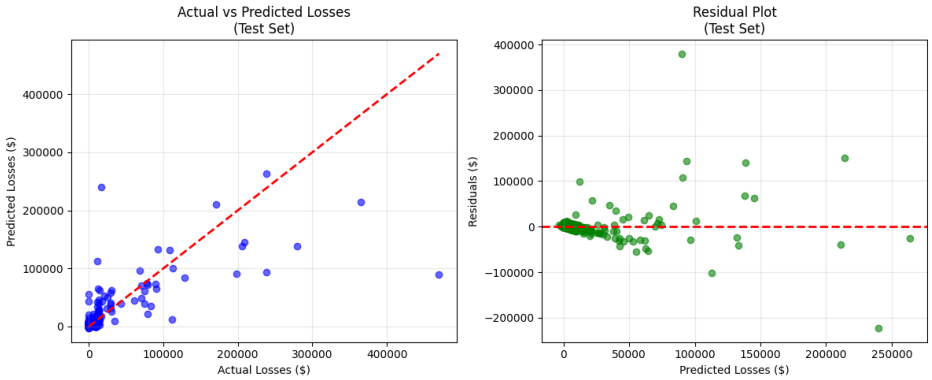


Figure 23: Predicted vs. actual losses and residuals for the regression model for losses prediction.

In contrast, the regression model exhibits a different pattern of errors, reflecting the continuous nature of the target variable. The lowest prediction errors occur for cases

with minimal losses, which are by far the most common in the dataset. As the magnitude of the losses increases, the model tends to produce larger deviations from the true values. An examination of the residuals shows that the errors are spread both above and below the actual values, with no systematic bias in one particular direction. However, extreme cases illustrate the limitations of the model: some predictions overestimate the actual losses by as much as $\$400,000$, while others underestimate them by more than $\$200,000$. These results underscore the difficulty of modeling rare and extreme loss events, which are inherently harder to predict due to their high variability and low frequency in the data.

4 Unit 4. Clustering

Unsupervised learning represents a fundamental branch of machine learning where the algorithm operates on datasets without pre-existing labels or target variables. Unlike supervised learning, which is task-driven and relies on categorized input-output pairs to train a model, unsupervised learning is strictly data-driven. The primary objective in this domain is to infer the natural structure, underlying patterns, or intrinsic distribution within the input data X without human intervention or "correct" answers provided during the training phase.
Within this framework, clustering serves as a primary technique for structure discovery. Clustering is defined as the process of organizing data points into distinct groups, or "clusters", based on similarity metrics. The mathematical goal is to partition the data such that items within the same group share a high degree of similarity (minimizing intra-cluster variance), while items in different groups are distinct from one another (maximizing inter-cluster variance). This technique is widely utilized for reducing data complexity and uncovering hidden relationships in various applications, such as customer segmentation in marketing, anomaly detection in security, and pattern recognition in healthcare.

4.1 K-means Clustering

K-means is a centroid-based clustering algorithm. It operates on the principle of partitioning an N-dimensional dataset into a pre-defined number of clusters, denoted as k. The algorithm characterizes each cluster by a central point known as a "centroid", which is the mean for all the points that belong to that cluster, and assigns every data point in the dataset to the nearest centroid. The ultimate mathematical objective of K-means is to minimize the total variance within the clusters, often referred to as the Within-Cluster Sum of Squares (WCSS) or inertia.
Building on the introductory discussion of K-means clustering, the following key insights summarize its practical behavior and limitations. These highlights have been extracted from the recommended instructional video and are intended to emphasize the most relevant aspects to consider when applying the algorithm in practice.

- **The Elbow Method:** the parameter k must be specified before execution, selecting an inappropriate k can lead to poor results. The "Elbow Method" is the standard heuristic for optimization. It involves plotting the total within-cluster variation (or average distance to the centroid) against a range of k values. As k increases, variation decreases, however, the goal is to identify the "elbow" of the curve, which is the point where adding another cluster yields diminishing returns in reducing variation. This point represents the optimal trade-off between compression and accuracy.

- **Sensitivity to Initialization:** The algorithm is highly sensitive to the initial random placement of centroids. A poor random initialization can result in the algorithm converging to a sub-optimal local minimum rather than the global minimum. To mitigate this, it is standard practice to run the algorithm multiple

times with different random initializations and select the result that yields the lowest total variance.

- **Cluster Shape and Density:** K-means relies on Euclidean distance and mean calculations, which implicitly assumes that clusters are spherical and linearly separable. Consequently, it struggles to correctly cluster data that forms complex, arbitrary shapes or data with varying densities.

Algorithm

Algorithm 1 K-means Clustering Algorithm

Require: Dataset $X = \{x_1, x_2, \ldots, x_n\}$, Number of clusters k
Ensure: Set of k clusters and their centroids

Step 1: Initialization
Randomly select k distinct data points from X as initial centroids $C = \{\mu_1, \mu_2, \ldots, \mu_k\}$

Step 2: Iterative Optimization
repeat
// Assignment Step
for each data point $x_i \in X$ **do**
Calculate Euclidean distance $d(x_i, \mu_j) = \|x_i - \mu_j\|^2$ for all $j \in \{1 \ldots k\}$
Assign x_i to the cluster S_j with the nearest centroid:
$S_j = \{x_p : \|x_p - \mu_j\|^2 \leq \|x_p - \mu_l\|^2 \forall l, 1 \leq l \leq k\}$
end for

// Update Step
for each cluster $j \in \{1 \ldots k\}$ **do**
Recalculate centroid μ_j as the mean of all points assigned to S_j:
$\mu_j = \frac{1}{|S_j|} \sum_{x \in S_j} x$
end for

until centroids μ do not change significantly (convergence)

The execution of the K-means algorithm, as shown in Algorithm 1, begins with the Initialization phase, where the algorithm arbitrarily selects k points from the data domain to serve as the initial "means" or centroids. In the video example, this is visualized by picking three random of the plane.
Following initialization, the algorithm enters an iterative loop consisting of two primary phases:

1. **Assignment Step.** The algorithm computes the distance (typically Euclidean) between every data point and each of the k centroids. Each point is then associ-

ated with the specific cluster centroid to which it is closest. Once all points have been categorized, the algorithm proceeds to the next step.

2. **Update Step.** The position of each centroid is recalculated by computing the arithmetic mean of all data points currently assigned to that specific cluster.

This cycle of assigning points and updating means repeats until a convergence criterion is met. Convergence occurs when the assignments no longer change between iterations, meaning the centroids have stabilized and the total variation within the clusters is minimized locally.

Implementation

In order to study and understand the K-means algorithm in greater depth, an explicit implementation has been developed from scratch.
The process begins with the random initialization of centroids. Specifically, k distinct observations are randomly selected from the dataset X without replacement, and these points are used as the initial centroids. This initialization defines the starting configuration of the algorithm. Since the selection is random, different executions may lead to different convergence paths and final solutions, highlighting the well-known sensitivity of K-means to its initial conditions.
Once the initial centroids are defined, the algorithm proceeds to the cluster assignment phase. In this step, the Euclidean distance between each data point and each centroid is computed. These distances are calculated in a vectorized manner, resulting in a distance matrix that captures the proximity of every observation to every centroid. Each data point is then assigned to the cluster associated with the nearest centroid, producing a vector of labels that specifies cluster membership for all observations.
After assigning points to clusters, the algorithm moves to the centroid update phase. For each cluster, a new centroid is computed as the arithmetic mean of all data points currently assigned to that cluster. If a cluster happens to receive no points in a given iteration, its centroid remains unchanged, preventing numerical instability. This update step directly reflects the definition of a centroid as the point that minimizes the sum of squared distances within the cluster.
The algorithm then iteratively alternates between the assignment and update phases. At each iteration, intermediate states are stored, including both the cluster assignments and the updated centroid positions. This allows for a detailed visualization and analysis of the clustering process, making it possible to observe how centroids gradually move toward regions of higher data density and how cluster boundaries stabilize over time.
The stopping criterion is based on centroid convergence. When the newly computed centroids are numerically close to those from the previous iteration, within a predefined tolerance, the algorithm is considered to have converged and the process terminates. At this point, neither the cluster assignments nor the centroid positions change, indicating that a local minimum of the objective function has been reached.
To evaluate the efficacy of the implemented K-means algorithm, a synthetic dataset was generated comprising 300 samples distributed across 3 distinct clusters, with a standard deviation of 1.5 per cluster.

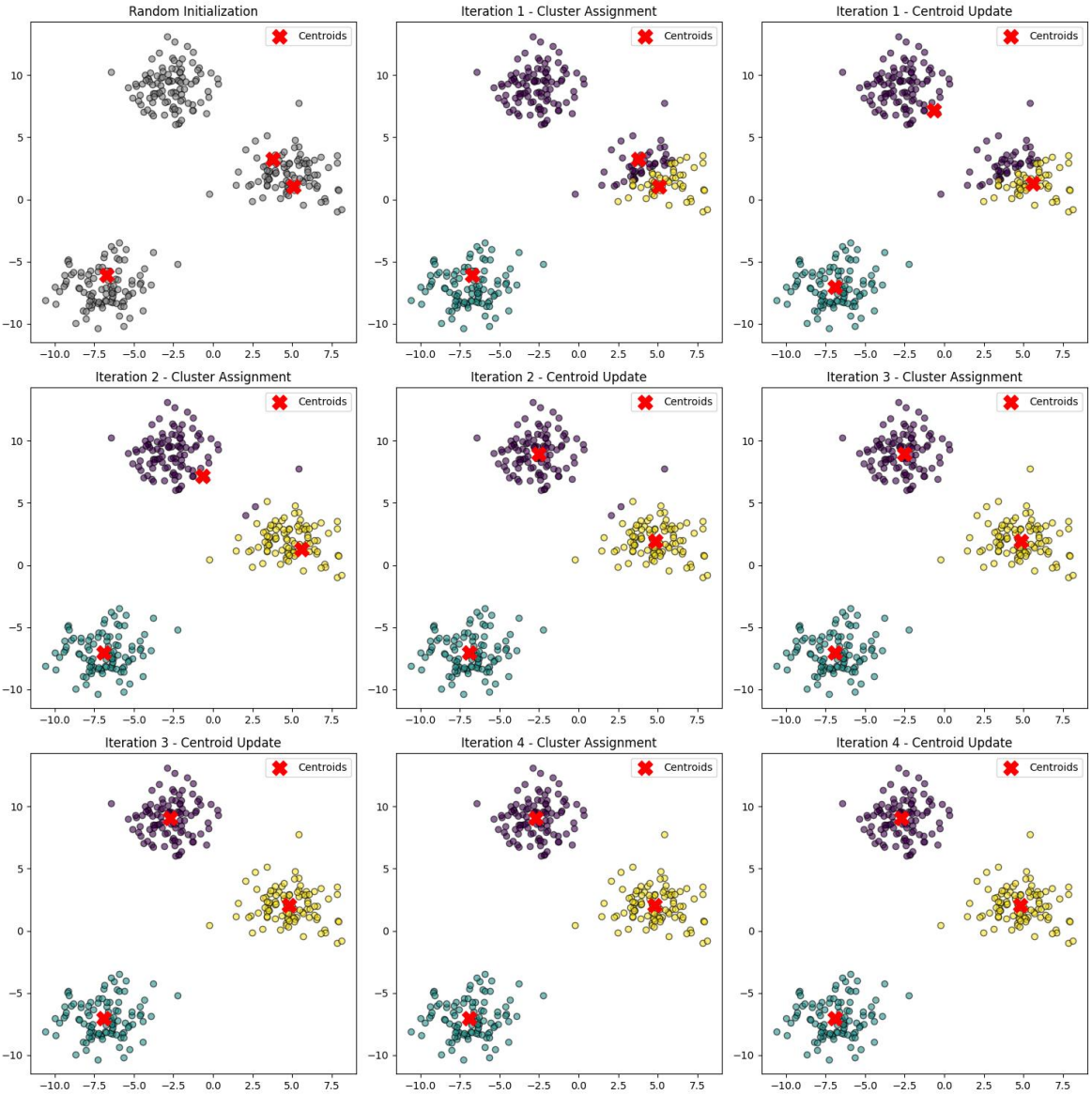


Figure 24: Iterative convergence of K-means from random initialization to final clusters.

The resulting visualization (Figure 24) offers a complete view of the algorithm's iterative performance and its capability to recover structure from unlabeled data.

The first panel, Random Initialization, establishes the baseline challenge. While the three natural clusters are visually distinct to the human observer, the algorithm's stochastic initialization selected starting points effectively at random. In this specific instance, the initialization was sub-optimal: two centroids were instantiated within the bounds of a single cluster, while the third centroid was placed in a separate group, leaving the final cluster (the upper grouping) initially unrepresented.

Despite this skewed starting configuration, the algorithm demonstrates high efficiency, achieving convergence in just 4 iterations. This rapid stabilization highlights the robustness of the coordinate descent approach used in K-means; even with a "bad" random

seed, the algorithm quickly minimized the within-cluster variance.
A closer examination of the intermediate iterations reveals the mechanics of this correction. As the algorithm alternates between the assignment and update steps, we observe a significant displacement of the centroids toward the true geometric centers of the data. This "correction trajectory" is most evident in the purple cluster (top grouping). Although its assigned centroid essentially began in a neighboring cluster, the iterative recalculation of the means successfully migrated the centroid across the feature space to its correct position.
Ultimately, the final iteration demonstrates that the algorithm successfully overcame the initialization bias. It correctly identified the underlying structure of the data, resulting in a segmentation that perfectly aligns with the three ground-truth clusters defined in the dataset generation phase.

4.2 Density-Based Clustering

The centroid-based methods like K-means partition the data space into distinct cells based on distance to a central mean, Density-Based Clustering represents a fundamental shift in perspective. Instead of assuming that clusters are spherical, this approach defines clusters as continuous regions of high data density separated by regions of low density.
In this paradigm, a cluster is essentially a dense "connected" component. This definition provides two significant advantages over partitioning methods. First, it allows for the discovery of clusters with arbitrary shapes, which K-means often fragments incorrectly. Second, it inherently incorporates the concept of noise or outliers. Data points located in low-density regions are not forced into a cluster but are instead classified as anomalies.
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is the most prominent algorithm within this family. It operationalizes the concept of density using two user-defined parameters: ϵ, which defines a radius around a data point and `MinPts`, which sets the minimum number of data points required within that radius to constitute a "dense" region.

Algorithm

The operation of the DBSCAN algorithm is articulated as a continuous process of exploration and density propagation, as can be seen in Algorithm 2.
The execution begins with a core point identification phase, where the algorithm iteratively traverses the dataset X, evaluating every point that has not yet been visited. For each point x, its local neighborhood is calculated based on the radius ϵ; if the number of neighbors within this area exceeds the threshold defined by MinPts, the point is classified as a "Core Point". This identification is critical, as it signals the discovery of a high-density zone and triggers the creation of a new cluster. Conversely, if the point does not meet this density criterion, it is provisionally labeled as noise, remaining available to potentially be claimed by another cluster in future iterations.
Once a new cluster is founded from a core point, the density expansion phase is immediately activated via the Expand procedure. This stage is responsible for defining the

Algorithm 2 DBSCAN Clustering Algorithm

Require: Dataset X, radius ϵ, MinPts
  Mark all $x \in X$ unvisited, $C \leftarrow 0$
  for $x \in X$ **do**
    if x unvisited **then**
      Mark x visited, $N \leftarrow$ neighbors(x, ϵ)
      if $|N| \geq$ MinPts **then**
        $C \leftarrow C + 1$, EXPAND(x, N, C)
      else
        Mark x noise
      end if
    end if
  end for
  procedure EXPAND(x, N, C)
    Assign x to C
    for $y \in N$ **do**
      if y unvisited **then** Mark y visited, $N \leftarrow N \cup$ neighbors(y, ϵ)
      end if
      if y not in any cluster **then** Assign y to C
      end if
    end for
  end procedure

shape of the cluster and functions through a chain-reaction mechanism: the algorithm takes all neighbors of the initial point and adds them to the cluster. As it examines these neighbors, if it finds that any of them is also a core point (i.e., it possesses its own dense neighborhood), their respective neighbors are dynamically added to the exploration list. This process of recursive aggregation allows the cluster to "grow" and propagate through density-connected data, absorbing all reachable points until the local density drops below the threshold, at which moment the expansion halts, and the algorithm returns to the main loop to search for new structures.

Implementation

To understand the mechanics of density-based clustering in greater depth, an explicit implementation of the DBSCAN algorithm has been developed, focusing on its ability to handle noise and varying densities.
The process begins with the generation of a synthetic dataset designed to challenge the algorithm's specific capabilities.
Once the data is prepared, the algorithm proceeds to the neighborhood calculation phase. Two critical hyperparameters are defined: the radius (ϵ), set here to 0.6, and the minimum number of points required to define a dense region ($MinPts$), set to 5. Unlike centroid-based approaches that calculate global distances, this implementation utilizes a nearest-neighbors search to efficiently map the local neighborhood of every

data point. This step identifies exactly which points fall within the ϵ, radius of one another.

With the neighborhood relationships established, the algorithm identifies the "core points" of the dataset. A boolean mask is computed where any point possessing at least $MinPts$ neighbors is flagged as a core point. This distinction is fundamental, as it effectively separates the dataset into points that have the structural density to initiate a cluster and those that are merely border points or outliers.

Initially, all points are labeled as noise (-1), establishing a baseline where no structure is assumed. The algorithm then enters the cluster expansion phase. It iterates through the dataset, bypassing points that have already been labeled or do not qualify as core points. When an unvisited core point is encountered, a new cluster ID is generated, and a queue-based propagation mechanism is triggered. This queue is initialized with the neighbors of the current core point, serving as the frontier for cluster growth.

Inside the expansion loop, the algorithm processes the queue to trace the density-connected components. For each point retrieved from the queue, if it was previously labeled as noise or unvisited, it is assigned to the active cluster. Crucially, if this point is also identified as a core point, its own neighbors are added to the queue. This recursive inclusion allows the cluster to "flow" through the dense regions of the data, adapting to arbitrary shapes until the density drops below the threshold.

The stopping criterion is implicit in the visitation of all points. Once the main loop completes, any data point that remains with the initial label of -1 is permanently classified as an outlier.

To evaluate the implementation rigorously, a synthetic dataset was generated using a fixed random seed to ensure reproducibility. The data construction involves three distinct components designed to challenge density-based methods: a dense cluster (150 samples, $\sigma = 0.3$) representing ideal structure, a sparse cluster (150 samples, $\sigma = 1.0$) to test sensitivity to varying densities, and 50 uniform outliers distributed across the feature space to simulate background noise. These elements are vertically stacked into a single heterogeneous dataset of 350 observations, specifically created to validate the algorithm's ability to distinguish meaningful structural patterns from random anomalies.

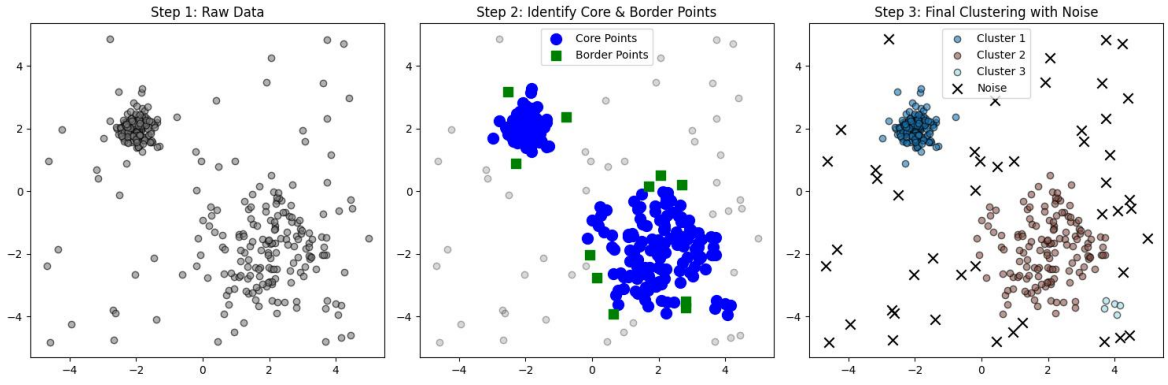


Figure 25: DBSCAN clustering on synthetic data generated with varying densities and noise.

Based on the visual evidence presented in Figure 25, the results strongly validate

the implementation. The first panel confirms the successful generation of the synthetic environment: we observe two distinct structures, a compact, high-density cluster in the upper-left quadrant and a dispersed, lower-density cluster in the lower-right—surrounded by effectively distributed noise.
The second panel highlights the algorithm's internal logic, demonstrating an accurate discrimination between core points, border points, and noise. Proceeding to the final classification, the algorithm successfully segments the two primary clusters as expected. However, a third, unintended cluster emerges from a small isolated subset within the sparse region. This fragmentation occurs because the local distance between this subgroup and the main body of the sparse cluster exceeded the defined ϵ threshold (0.6). Consequently, the chain of density-reachability was broken. However, since this isolated pocket still satisfied the $MinPts$ condition (containing at least 5 points), DBSCAN mathematically identified it as an independent cluster rather than merging it or classifying it as outliers.

4.3 Hierarchical Clustering

The Hierarchical Clustering is a method that seeks to build a hierarchy of clusters rather than a single partition. Unlike K-means, this technique does not require the user to specify the number of clusters (k) in advance. Instead, it creates a tree-like structure that represents the data at various levels of granularity.
The presentation highlights two main approaches to achieving this hierarchy:

- **Agglomerative (Bottom-Up)**: This is the most common approach. It starts by treating each data point as a single cluster. Then, in successive steps, it merges the pairs of clusters that are closest to each other until all clusters have been merged into a single cluster containing all data points.
- **Divisive (Top-Down)**: This approach works in the opposite direction. It starts with all data points in one giant cluster and recursively splits the clusters into smaller ones until each data point is in its own cluster.

The results of hierarchical clustering are visualized using a dendrogram, a tree diagram that records the sequences of merges (or splits). The vertical axis (height) represents the distance or dissimilarity between clusters. The higher the connection between two branches, the more dissimilar the clusters are. Then, to determine a specific number of clusters, a "horizontal cut" is made across the dendrogram at a specific height (distance threshold). The number of vertical lines the cut intersects is equal to the number of clusters (k) at that level. This allows the user to select the optimal number of clusters visually by observing the structure of the tree.

Implementation

To evaluate the hierarchical classifier, we implemented a classifier using standard scientific libraries to manage the complexity of the clustering process. The core of the procedure begins with the calculation of the linkage matrix. This matrix is fundamental as it computes and stores the distances between all clusters and records the

history of every merge. Subsequently, based on this linkage information, the dendrogram is computed and plotted, serving as the primary tool to visualize the hierarchical relationships within the data.

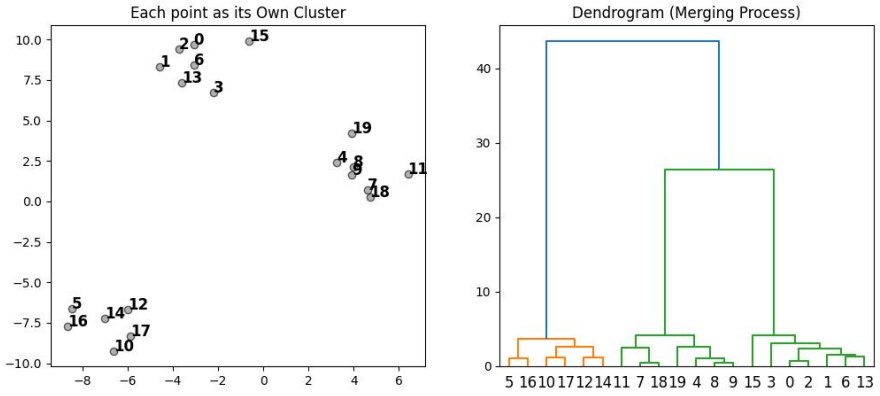


Figure 26: Raw data and dendrogram showing the hierarchical merging process and cluster distances.

The results of this process are presented in Figure 26. In the first image (left panel), we can see the initial state of the algorithm where each data point forms its own unitary cluster. It is from this granular state that the hierarchical building process begins.
Moving to the image on the right, we observe the calculated dendrogram. The structure is clearly defined: the initial unions between individual points are formed in the lowest part of the tree, specifically below a distance of 5 on the Y-axis. This behaviour indicates that the points that belong to a cluster are very similar. Furthermore, we can observe how, in general, the algorithm suggests the formation of 3 clusters; this is evident because there is a long, stable vertical gap ranging from approximately 5 up to 25 (Y-axis) where exactly three vertical branches exist.

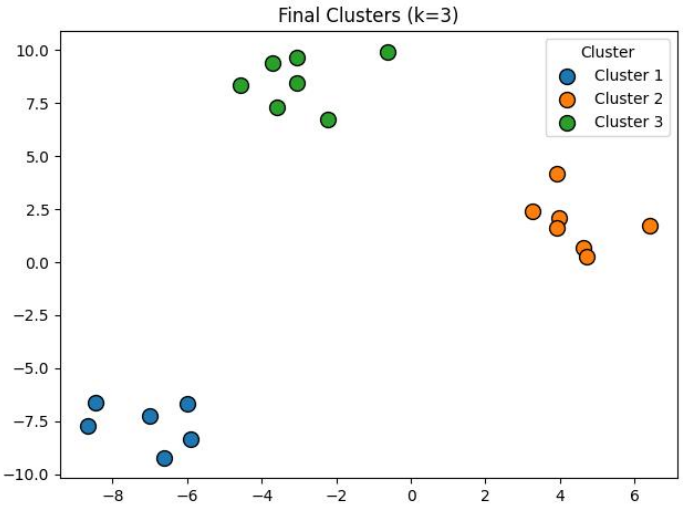


Figure 27: Final cluster assignment from hierarchical agglomerative clustering.

Based on this visual analysis, a horizontal cut is performed to define exactly 3 clusters. As observed in Figure 27, we can see how the algorithm successfully identifies each of the clusters. The resulting classification aligns perfectly with the spatial distribution of the data, accurately separating the three groups without any overlap or misclassification, demonstrating the effectiveness of Hierarchical Clustering in recovering the natural structure of the dataset.

4.4 Distribution-Based Clustering (Gaussian Mixture Models)

While K-means assumes clusters are hard partitions defined by spherical distances to a center, Distribution-Based Clustering takes a probabilistic approach. It operates on the assumption that the data is generated from a mixture of several underlying probability distributions—typically Gaussian (Normal) distributions.
In this framework, a cluster is not just a geometric region but a mathematical model represented by a Gaussian distribution. Each distribution is defined by parameters that describe the cluster's characteristics:

- **Mean (μ)**: Defines the center of the cluster.
- **Covariance (Σ)**: Defines the shape, spread, and orientation of the cluster (ellipsoidal vs. spherical).
- **Mixing Coefficient (π)**: Represents the weight or probability of a data point belonging to that specific cluster.

The algorithm used to fit these models is called Expectation-Maximization (EM). Unlike K-means, which assigns a point strictly to one cluster (hard clustering), GMM calculates the probability that a point belongs to each cluster (soft clustering).

Implementation

To fully understand the mechanics of the Expectation-Maximization (EM) algorithm, a custom `GaussianMixtureModel` class was developed to encapsulate the iterative probabilistic logic. The initialization phase within the fit method establishes the starting state of the system: the mixture weights are distributed uniformly, the initial means are selected stochastically from the dataset to ensure they fall within the valid data domain, and the covariance matrices are initialized as identity matrices, assuming spherical distributions at the outset.
The core of the implementation is the EM loop, which alternates between two distinct phases until convergence. In the Expectation step (E-step), the algorithm computes the "responsibilities", which quantify the probability of each data point belonging to each specific cluster based on the current parameters. This is achieved by evaluating the Gaussian Probability Density Function (PDF) for every point and normalizing the results so that the probabilities across all components sum to one. Subsequently, in the Maximization step (M-step), these responsibilities serve as weights to update the model

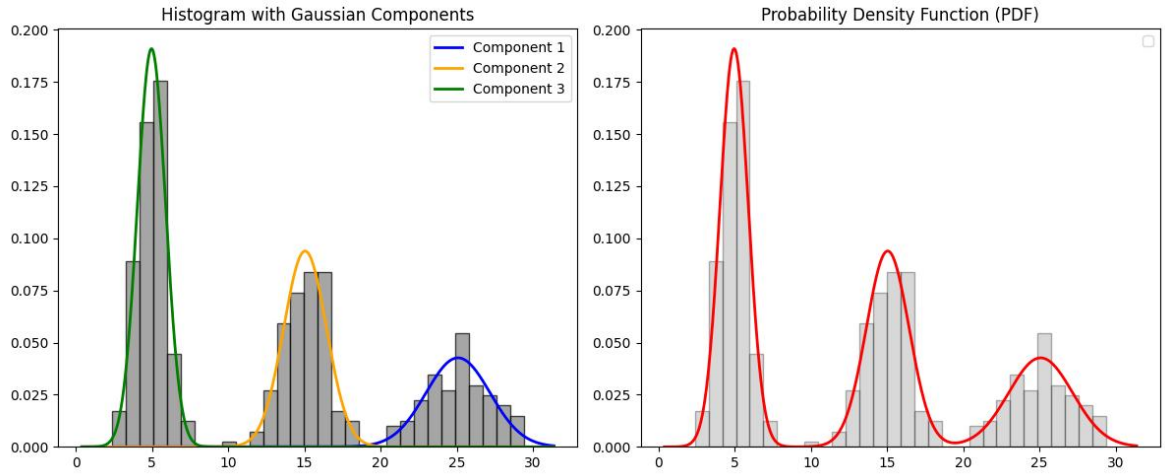


Figure 28: Gaussian Mixture Model fitting a tri-modal synthetic distribution.

parameters. The new means are calculated as the weighted average of the data points, while the new covariance matrices are derived from the weighted squared deviations from the means. A crucial detail in this implementation is the addition of a small regularization term (epsilon 10^{-6}) to the diagonal of the covariance matrices, ensuring numerical stability and preventing singularity issues during matrix inversion.
The loop concludes with a convergence check, where the log-likelihood of the data is computed to measure how well the model fits the distribution. If the improvement in log-likelihood between iterations falls below a defined tolerance threshold, the algorithm terminates early, signalling that a local maximum has been reached. Finally, the predict method utilizes the learned parameters to assign labels to data points by selecting the component with the highest responsibility.
To evaluate the classifier, a 1D synthetic dataset was constructed to simulate a multi-modal distribution. The data generation process (np.concatenate) combines three distinct Gaussian distributions to create clear "peaks" in the data:

- Cluster 1: A dense peak centered at 5 ($\mu = 5$) with low variance ($\sigma = 1.0$).
- Cluster 2: A central peak at 15 ($\mu = 15$) with moderate spread ($\sigma = 1.5$).
- Cluster 3: A broader, flatter peak at 25 ($\mu = 25$) with high variance ($\sigma = 2.0$).

This composite dataset of 450 points is then reshaped into a column vector to mimic a single feature input.
The results presented in Figure 28 demonstrate that the algorithm successfully identifies each of the underlying distributions, as the inferred Gaussian curves simulate the behavior of the synthetic data almost to perfection. However, the validation of the model extends beyond visual inspection; a quantitative analysis of the learned parameters—specifically the means and weights—reveals near-perfect numerical alignment with the ground truth.
Regarding the means, the model demonstrates high precision in locating the centers of the sub-populations. For the cluster generated with a theoretical mean of 25, the

model estimates a centroid of 25.078, a deviation of less than 0.1%. A similar degree of accuracy is observed for the other two distributions, with estimated means of 15.040 (target 15) and 4.959 (target 5), respectively.
Furthermore, the analysis of the component weights (π) confirms the algorithm's ability to correctly infer the relative proportions of each cluster within the mixture. Considering the total dataset size of 450 observations, the first distribution (mean 5) contains 200 samples, which theoretically represents 44.44% of the total mass. The model assigns a weight of 0.44444465 to the corresponding component, matching the actual proportion almost exactly. This robustness holds for the remaining clusters as well: the component corresponding to mean 15 was assigned a weight of 0.3307 (reflecting the 150 samples or 33.3%), and the component for mean 25 received a weight of 0.2248 (reflecting the 100 samples or 22.2%). These metrics conclusively demonstrate the correct functioning of the implementation and its capacity to deconstruct complex multi-modal distributions.

4.5 Customer Segmentation

Having explored the theoretical foundations and synthetic implementations of K-means, Hierarchical Clustering, and DBSCAN, We wanted to explore how they work in a real-world data. That is because synthetic datasets are useful for isolating specific algorithmic behaviors, but they lack the noise, outliers, and complex feature correlations inherent in production environments.
To address this, we utilized the Online Retail II dataset, sourced from the UCI Machine Learning Repository. This dataset contains actual transaction records from a UK-based, non-store online retail business registered between 2009 and 2011. The primary objective of this phase is to perform Customer Segmentation. By grouping customers based on their purchasing behavior, we aim to observe how each algorithm handles high-dimensional, non-uniform data and to compare their effectiveness in identifying distinct consumer profiles.

4.5.1 Implementation

To effectively apply clustering algorithms and extract meaningful patterns, raw transactional data must first undergo a rigorous preprocessing pipeline, as algorithms like K-means and DBSCAN are highly sensitive to noise and data structure. The process commences with the loading of the dataset, followed immediately by a filtering phase designed to isolate genuine purchasing behaviour. this involves excluding records with non-positive quantities or prices—which typically indicate returns, cancellations, or system errors—and dropping rows missing a 'Customer ID' to ensure every data point corresponds to an identifiable client. Following this cleaning phase, the granular transaction-level data is transformed into a customer-centric feature matrix through an aggregation process. By grouping the data by unique customer identifiers, we extract key behavioural metrics: the unique count of invoices is calculated to measure purchase frequency, while the total sum of quantities provides an indicator of consumption volume. Additionally, the average price is computed to distinguish between high-value and budget customers, and a 'recency' metric is derived by calculating the number of

days between a customer's first and last activity, representing their lifespan within the dataset. Finally, to ensure mathematical continuity, any missing values resulting from this aggregation are imputed using the column-wise mean, producing a complete and dense dataset.
With the dataset transformed into a feature matrix, the experimental framework was established to rigorously compare the three algorithms under identical conditions. First, a StandardScaler is applied to normalize the features to a mean of 0 and a standard deviation of 1; this step is critical because clustering algorithms calculate distances in Euclidean space, and without scaling, features with larger magnitudes (like total quantity) would disproportionately dominate the objective function over smaller features (like average price).
Three distinct models were defined to test different clustering paradigms: K-Means and Hierarchical Clustering (Agglomerative) were both configured with $k = 4$, assuming the existence of four primary customer segments for direct comparison, while DBSCAN was configured with $\epsilon = 1.5$ and $MinPts = 5$ to test for natural density discovery without a pre-imposed cluster count. To evaluate and visualize the results of these 4-dimensional models, Principal Component Analysis (PCA) is employed to project the standardized data into a lower-dimensional 2D space (PC1 and PC2), allowing for the visual inspection of cluster separability. Quantitatively, the performance is assessed using the Silhouette Score to measure cluster cohesion and separation, alongside execution time to gauge computational efficiency.

4.5.2 Results

The comparative analysis of the three clustering algorithms yields significant insights regarding both computational efficiency and segmentation quality.

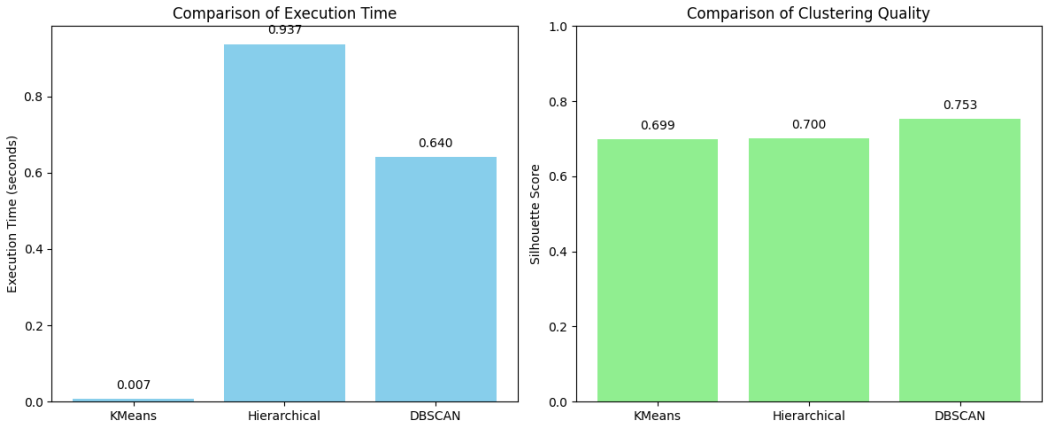


Figure 29: Performance comparison of clustering algorithms on customer data.

The first critical observation arises from the trade-off between execution time and the Silhouette Score (Figure 29). K-means demonstrates superior performance, combining the lowest computational cost with a high Silhouette Score. While the difference in Silhouette scores between K-means and Hierarchical Clustering is negligible. Hierarchical

Clustering, due to its time complexity depending on the linkage method, requires significantly more time to compute the distance matrix for the entire dataset. In contrast, K-means, with its linear complexity converges rapidly. Given that K-means achieves the same quality of segmentation as Hierarchical Clustering but at a fraction of the time, it emerges as the most robust and scalable choice for this specific dataset.

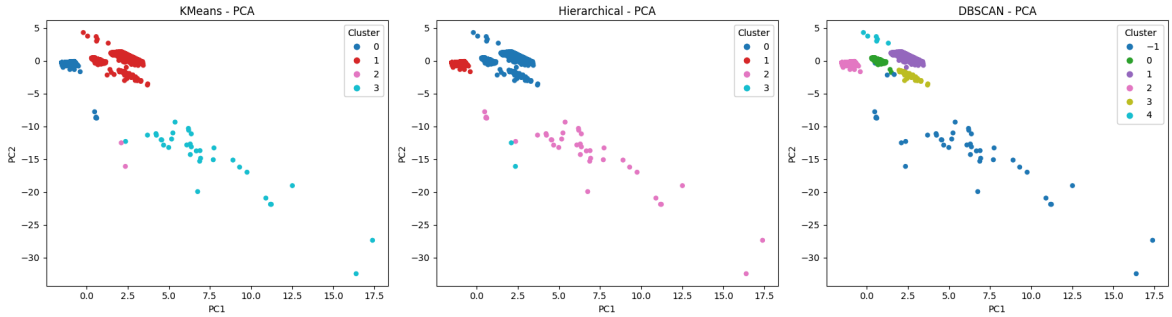


Figure 30: PCA visualization of customer clusters identified by different algorithms.

Visually, the projection of clusters onto the Principal Components (PCA) reinforces the quantitative findings. K-means partitions the latent space into clear, distinct convex regions, effectively capturing the variance in customer behavior. In contrast, DBSCAN struggles with the varying densities inherent in retail data; the presence of outliers makes it difficult to find a single global ϵ that works for all clusters, leading to either excessive noise classification or merged clusters. Hierarchical clustering produces a structure very similar to K-means but fails to justify its computational overhead. Consequently, K-means is selected as the optimal method for profiling the customer base.

Table 3: Behavioral profiles of customer segments derived from clustering.

Cluster	Num Trans	Total Quantity	Avg Price	Recency (Days)
0	2.23	44.93	7.09	71.93
1	10.37	2,503.10	3.41	541.47
2	1.50	1.50	8,955.84	0.00
3	110.41	88,433.74	6.61	620.21

The quantitative analysis of the cluster centroids (Table 3) reveals four distinct behavioral patterns within the customer base. Cluster 0 (Casual Shoppers) constitutes the largest segment of the population but contributes the least in terms of engagement. These customers exhibit a low purchase frequency (approximately 2 transactions) and a relatively short lifespan in the dataset (72 days), suggesting they are likely one-time buyers or new customers who have not yet established brand loyalty.
In contrast, Cluster 1 (Loyal Regulars) represents the steady, reliable core of the business. With an average tenure of over 541 days and a significantly higher purchase volume (2,500 units) compared to casual shoppers, these customers transact consistently over long periods. While their average spend per item is lower (£3.41), their sustained engagement makes them crucial for recurring revenue.

The most valuable segment is undoubtedly Cluster 3 (Whales). Although smaller in number, these customers drive massive volume, purchasing an average of over 88,000 units across 110 separate transactions. Their extensive history with the platform (620 days) and high frequency indicate they are likely B2B clients or wholesalers. Finally, Cluster 2 (High-Ticket Anomalies) captures a niche or potential data artifact characterized by an astronomical average price (£8,955) but minimal activity (1.5 transactions). This group likely represents specific bulk orders of high-value items or manual system adjustments (such as postage fees) rather than standard retail behavior.

5 Unit 5. K-means and Hierarchical Clustering

5.1 K-means

One of the fundamental constraints of the K-means algorithm is that it is a parametric method requiring the number of clusters, k, to be specified before execution. Since the optimal number of partitions is often unknown in real-world unsupervised scenarios, several statistical heuristics are employed to evaluate the quality of clustering across different values of k. The presentation highlights three primary metrics for this purpose: the Elbow Method, the Silhouette Analysis, and the Gap Statistic.

The Elbow Method (Within-Cluster Sum of Squares)

The Elbow Method is perhaps the most intuitive heuristic. It evaluates the compactness of clusters by calculating the Within-Cluster Sum of Squares (WCSS), also known as inertia. This metric quantifies the variance within the clusters; mathematically, it is the sum of the squared Euclidean distances between each data point and its assigned centroid.
The interpretation relies on the principle of diminishing returns. As the value of k increases, the WCSS naturally decreases because the clusters become smaller and points are closer to their centroids (in the extreme case where $k = N$, WCSS is zero). To find the optimal k, one plots the WCSS against the number of clusters. The goal is to identify the "elbow" of the curve—the point of inflection where adding another cluster no longer results in a significant reduction in variance. This point represents the optimal trade-off between minimizing error and maintaining a parsimonious model.

The Silhouette Method

While the Elbow Method focuses solely on compactness (cohesion), the Silhouette Method provides a more comprehensive assessment by validating both cohesion and separation. For each data point i, the silhouette coefficient $s(i)$ is calculated using two measures: $a(i)$, the average distance between i and all other points in the same cluster (intra-cluster distance), and $b(i)$, the minimum average distance between i and all points in the nearest neighboring cluster (inter-cluster distance).
Interpretation: The coefficient ranges from -1 to +1. A value near +1 indicates that the sample is far away from the neighboring clusters and very close to its assigned cluster, implying a strong and correct definition. A value near 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters (overlapping). A value near -1 indicates that those samples might have been assigned to the wrong cluster.
To select the optimal k, one typically looks for the configuration that maximizes the average silhouette score across the entire dataset, ensuring that clusters are both dense and well-separated.

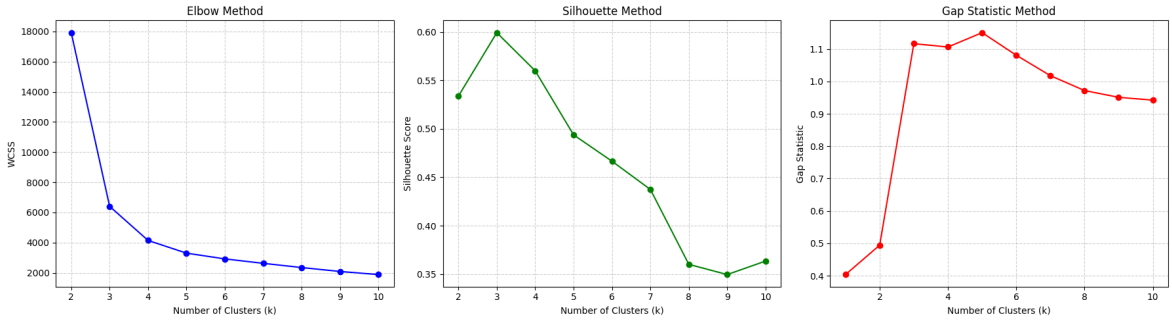


Figure 31: Determination of optimal k using elbow, silhouette, and gap statistic methods.

The Gap Statistic

The Gap Statistic offers a more formalized statistical approach compared to the previous heuristics. It compares the total intra-cluster variation for different values of k with their expected values under a null reference distribution of the data (typically a uniform distribution over a box aligned with the principal components of the data).
The algorithm calculates the "gap" between the log of the expected WCSS (from the random null reference) and the log of the observed WCSS. The interpretation is straightforward: the optimal k is the value that maximizes the Gap Statistic. This maximizes the difference between the clustering structure found in the actual data and the random structure of noise, effectively identifying the k where the clusters are most distinct from random background distribution.

Implementation

To empirically validate the effectiveness of these metrics, a synthetic experiment was conducted using generated data.
A dataset of 500 samples was created, specifically configured to have 5 distinct centers and a standard deviation of 2.0. This setup provides a known "ground truth" ($k = 5$) against which the heuristics can be tested. The algorithm then iterates through a range of candidate k values from 2 to 10. For each k, the K-means algorithm fits the data, and then each metric is calculated.
The results obtained from the three diagnostic heuristics (Figure 31) present a divergent set of recommendations ($k = 4$, $k = 3$, and $k = 5$, respectively). This disagreement is common in unsupervised learning and necessitates a nuanced interpretation of what each metric prioritizes in the data structure.
The Elbow Method identified the optimal parameter at $k = 4$. This suggests that the "elbow" of the Within-Cluster Sum of Squares (WCSS) curve, the point where the marginal gain in variance reduction significantly diminishes, occurred at four clusters. Technically, this implies that while splitting the data into a fifth cluster does reduce the inertia, the algorithm deemed the reduction insufficient relative to the cost of added complexity. This heuristic tends to prioritize compactness and can sometimes be conservative if two distinct clusters are spatially close, potentially merging them into a single entity.

The Silhouette analysis proposed an even more conservative structure, peaking at $k = 3$. Since the Silhouette coefficient measures the trade-off between intra-cluster density and inter-cluster separation, this result indicates that when the data is forced into 3 groups, the resulting clusters are the most "distinct" and well-separated from one another. The Silhouette metric penalizes proximity, leading it to recommend merging adjacent groups to maximize the average separation distance, resulting in an under-segmentation of the dataset.
In contrast to the geometric heuristics, the Gap Statistic suggested $k = 5$. This metric compares the observed clustering structure against a null reference distribution (random noise). By identifying 5 as the optimal point, the Gap Statistic signals that the structural evidence for 5 distinct groups is statistically stronger than for any other configuration when compared to randomness.

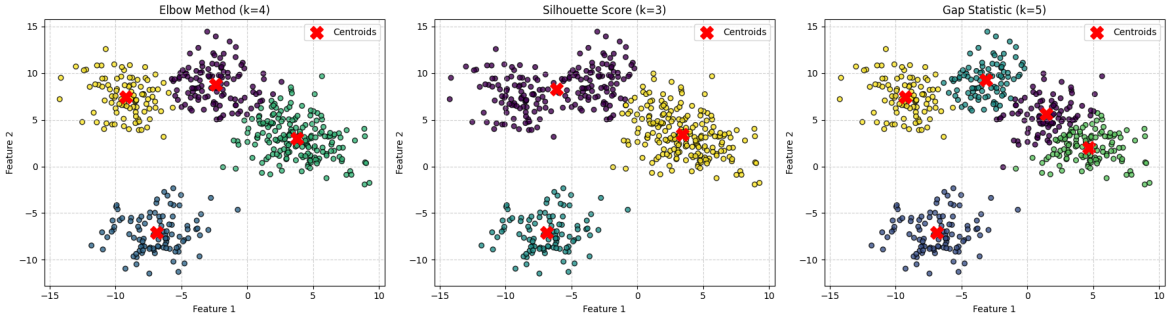


Figure 32: Visual comparison of K-means clustering for different values of k.

When comparing predictions with the reality of the data generated, the Statistical Gap is indisputably confirmed as the most robust and accurate method in this experiment. While the Silhouette Coefficient failed because it was overly conservative in the face of data overlap, and the Elbow Method offered a suboptimal approximation due to the subtlety of variance reduction, the Gap Statistic managed to recover the exact topological structure of the data.

5.2 Hierarchical Clustering

In Agglomerative Hierarchical Clustering, the algorithm iteratively merges pairs of clusters. The decision of which pair to merge depends on how we define the distance (or similarity) between two groups of points. This definition is called the Linkage Criterion. While the distance between individual points is usually calculated using Euclidean or Manhattan distance, the linkage criterion determines how we measure the distance between two clusters, A and B, which may contain multiple points.

- **Single Linkage (Nearest Neighbor):** This method defines the distance between two clusters as the minimum distance between any single point in cluster A and any single point in cluster B. It tends to produce long, chain-like clusters (a phenomenon known as "chaining") because a single pair of close points can merge two large groups, even if the rest of the points are far apart.

- **Complete Linkage (Farthest Neighbor):** This method defines the distance between two clusters as the maximum distance between any point in cluster A and any point in cluster B. It forces clusters to be compact and spherical. A merge only happens if all points in both clusters are relatively close to each other, avoiding the chaining effect of single linkage.

- **Average Linkage:** This method calculates the distance between two clusters as the average of all pairwise distances between every point in cluster A and every point in cluster B. It represents a middle ground between Single and Complete linkage, considering the overall structure of the clusters rather than just the extreme points.

- **Centroid Linkage:** This method computes the centroid (geometric center) of each cluster and defines the distance between clusters as the distance between their respective centroids. It is computationally efficient but can sometimes result in "inversions" in the dendrogram (where a parent node has a lower height than its children), which makes interpretation difficult.

Ward's Method

Ward's Method (Ward's Minimum Variance Method) is distinct from the simple linkage criteria described above. Instead of measuring direct distances between points, it is an objective function approach that seeks to minimize the total within-cluster variance. At each step of the algorithm, Ward's method analyzes every possible pair of clusters that could be merged. For each potential merge, it calculates how much the Error Sum of Squares (ESS) (or total variance) of the system would increase.

1. The ESS of a cluster is the sum of the squared distances between every point in the cluster and the cluster's centroid.

2. Initially, when every point is its own cluster, the variance is zero.

3. The algorithm selects the pair of clusters to merge that results in the minimum increase in total within-cluster variance.

Ward's method is biased towards creating compact, spherical clusters of roughly equal size. It is widely used because it tends to produce interpretable and well-structured dendrograms, avoiding the extremes of long chains (Single Linkage) or strictly compact but potentially fragmented groups (Complete Linkage). It effectively acts like a "hierarchical K-means", minimizing the same objective function (variance) but in a greedy, bottom-up fashion.

Implementation

To empirically evaluate the structural impact of different linkage criteria on the formation of hierarchical clusters, a comparative experiment was conducted using a synthetic dataset. The data generation process involved creating 250 samples distributed across

5 distinct centers with a standard deviation of 1.0, establishing a clear ground truth to validate the clustering performance.
The core of the experiment consisted of iterating through four distinct linkage methods: Single, Complete, Average, and Ward; and apply them to the exact same dataset. For each method, the linkage function computed the hierarchical clustering matrix (Z), encoding the history of merges and the distances at which they occurred. These matrices were then visualized as dendrograms.

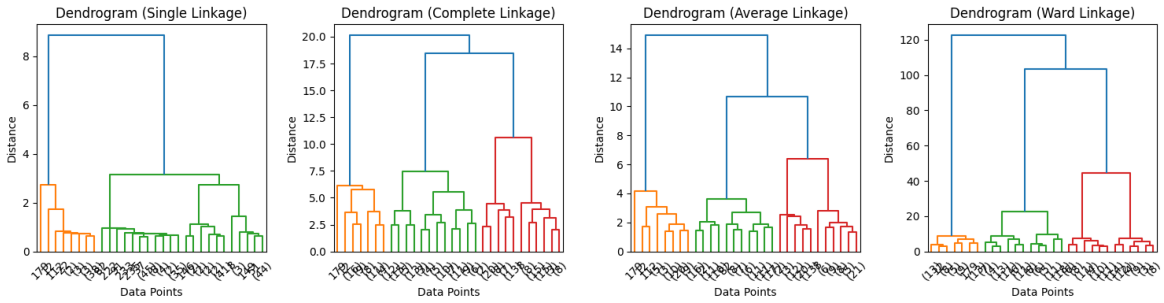


Figure 33: Dendrograms generated using different linkage criteria.

The Figure 33 highlights the profound influence the choice of linkage criterion has on the shape, height, and interpretability of the cluster hierarchy.
The dendrogram for Single Linkage exhibits the classic "chaining effect" The tree structure is highly compressed on the Y-axis (Distance), with most merges occurring at very low values. Instead of distinct, balanced branches where individual points or small groups are sequentially added to a larger cluster one by one. Consequently, identifying the 5 ground-truth clusters is difficult; a horizontal cut would likely isolate single outliers rather than meaningful groups. This confirms the method's sensitivity to noise and its tendency to create elongated, unstructured shapes.
Both Complete and Average linkage produce significantly more balanced structures than Single Linkage. In the complete linkage the Y-axis scale increases compared to Single Linkage because it measures the maximum distance between clusters. The structure is more compact, forcing distinct groupings. However, the branches representing the final 5 clusters are not as clearly separated vertically from the lower sub-branches. In the other hand, in the average linkage the dendrogram represents a middle ground. It balances the clusters better than Single Linkage but still retains some sensitivity to the internal spread of the data.
Finally, Ward's Method delivers the most definitive and interpretable structure. The most striking feature is the scale of the Y-axis, which is significantly larger than the others (reaching over 100). This occurs because Ward minimizes the sum of squared differences (variance) rather than simple geometric distance. Visually, Ward's dendrogram shows distinct, long vertical lines before the final merges. This "vertical gap" indicates high stability; there is a wide range of threshold values where a horizontal cut would unambiguously result in exactly 5 clusters. The hierarchy is balanced and clean, perfectly recovering the 5 centroids generated in the synthetic data.

5.3 Biased-Randomization and Ward's Method

Constructive heuristics, such as the standard Ward's method discussed previously, are widely used because they are efficient and easy to implement. However, they share a significant limitation: they are deterministic and greedy algorithms Grasas et al., 2017. In a deterministic greedy approach, the algorithm always makes the locally optimal choice at every step. For example, in Ward's method, the algorithm calculates the increase in error for all possible merges and strictly selects the one with the absolute minimum value. While this strategy seems logical, it is often described as "myopic" behavior. By always choosing the best immediate option, the algorithm fails to consider how that choice might constrain future decisions, often leading to a final solution that is a local optimum rather than the global best Grasas et al., 2017. Furthermore, because the process is deterministic, running the algorithm multiple times on the same dataset will always yield the exact same result, making it impossible to explore alternative clustering structures.
To address these limitations without losing the "common sense" logic of the heuristic, we can apply a Biased-Randomized Procedure (BRP). As described by Grasas et al., 2017, this methodology transforms a deterministic heuristic into a probabilistic algorithm.

The Logic of Biased Randomization

The core idea of Biased Randomization is to introduce a slight modification to the greedy constructive behavior. Instead of always selecting the absolute best candidate, the algorithm is allowed to select other promising candidates with a certain probability. In the context of Ward's method, the process changes as follows:

1. **Ranking Candidates:** At each step of the clustering process, we calculate the cost (ΔESS) for all potential merges. We then sort this list of candidates from the best (lowest variance increase) to the worst.

2. **Assignable Probabilities:** In a purely random approach, any merge in the list would have an equal chance of being selected. However, this would destroy the logic of the clustering. In a Biased-Randomized approach, we assign a probability of selection to each candidate that depends on its position in the sorted list.

3. **Skewed Distributions:** To maintain the heuristic's logic, we use a skewed theoretical probability distribution, such as the Geometric distribution or the Decreasing Triangular distribution. These distributions assign a very high probability to the top elements of the list (the "greedy" choices) and a rapidly decreasing probability to the subsequent elements.

This mechanism ensures that the algorithm still behaves intelligently—prioritizing good merges—but introduces enough variation to avoid getting trapped in the same local optimum every time Grasas et al., 2017.

The Role of the Geometric Distribution

A common distribution used in these procedures is the Geometric distribution, which is controlled by a single parameter β (where $0 < \beta < 1$). This parameter controls the degree of randomness in the selection process:

- If β is close to 1, the distribution becomes very steep. The probability of choosing the top candidate becomes nearly 100%, making the algorithm behave almost exactly like the deterministic Ward's method.
- If β is close to 0, the distribution flattens out, and the selection process becomes more like a uniform random selection.

One of the main advantages of using theoretical distributions like the Geometric, as opposed to empirical weights, is computational efficiency. The selection of the next candidate can be performed using analytical expressions (mathematical formulas) rather than complex loops. This allows the algorithm to remain extremely fast, which is crucial when dealing with large datasets Grasas et al., 2017.

Multi-Start and Parallelization

Because the Biased-Randomized Ward's method is probabilistic, it produces a different dendrogram each time it is executed. This feature allows us to implement a multi-start strategy.
We can execute the algorithm in parallel (for example, running 100 iterations simultaneously on different computer threads), with each execution using a different random seed. This generates a population of different solutions in roughly the same time it would take to run the standard heuristic once. Finally, we can analyze this population of solutions to find the one with the best global performance (e.g., the highest Silhouette Score) or to identify a "consensus" clustering structure that appears most frequently. This approach significantly improves the robustness and quality of the final clustering solution compared to the traditional single-pass method Grasas et al., 2017.

5.4 Project - Clustering by Sales Over Time

Clustering techniques play a fundamental role in the analysis of large-scale retail data, particularly when the objective is to identify groups of entities that exhibit similar behavior without relying on predefined labels. In the retail sector, understanding similarities between stores based on their sales dynamics is essential for a wide range of strategic decisions, including targeted marketing campaigns, inventory optimization, demand forecasting, and supply chain planning.
This project focuses on the application of unsupervised learning, specifically K-means clustering, to group retail stores according to their historical sales patterns over time. Rather than analysing individual transactions or products in isolation, each store is treated as a multivariate time series that captures its sales behaviour across different periods. This perspective allows for the identification of stores that respond similarly to

seasonality, promotions, or demand fluctuations, even if they differ in size or absolute sales volume.
The dataset used in this project contains historical sales records from multiple stores and products, including information such as store identifiers, product identifiers, dates, quantities sold, and additional metadata related to store characteristics (e.g., location or store type). By aggregating and restructuring this information at the store level, each store can be represented as a high-dimensional feature vector summarizing its temporal sales profile.
The main challenge addressed in this project is to transform raw transactional data into a meaningful representation suitable for clustering, and to determine an appropriate number of clusters that capture distinct sales behaviors across stores. By applying K-means clustering, the goal is to uncover latent groupings of stores that share similar sales trajectories, providing actionable insights that can support data-driven decision-making in retail environments.

5.4.1 Implementation

The original dataset consists of transactional sales records at the product and store level, with each observation associated with a specific date. Since the objective of this project is to cluster stores based on their temporal sales behavior, the first preprocessing step involves aggregating the data at the store–date level. Specifically, total daily sales are computed for each store by summing sales across all products sold on the same date.
Once the data are aggregated, the dataset is reshaped into a store-by-time matrix using a pivot operation. In this representation, each row corresponds to a store, each column corresponds to a date, and each cell contains the total sales of that store on that date. Missing values, which arise when a store has no recorded sales on a given date, are filled with zeros. This transformation allows each store to be treated as a high-dimensional feature vector capturing its complete sales trajectory over time.
Because K-means clustering is sensitive to the scale of the input features, the store–time matrix is standardized using z-score normalization. This step ensures that all time periods contribute equally to the distance calculations, preventing stores with systematically higher sales volumes from dominating the clustering process purely due to scale effects.
The clustering process is performed using the K-means algorithm, which partitions stores into k clusters by minimizing the within-cluster sum of squared distances. A critical aspect of applying K-means is the selection of an appropriate number of clusters. To guide this choice, two complementary validation techniques are employed: the within-cluster sum of squares (WCSS) and the silhouette score.
After selecting the number of clusters, the K-means algorithm is fitted to the standardized store sales data, and each store is assigned to a cluster. To facilitate interpretation and visualization, Principal Component Analysis (PCA) is applied to project the high-dimensional sales vectors into a two-dimensional space.
This low-dimensional representation does not affect the clustering itself but allows the clusters to be visually inspected. The resulting scatter plot provides an intuitive view

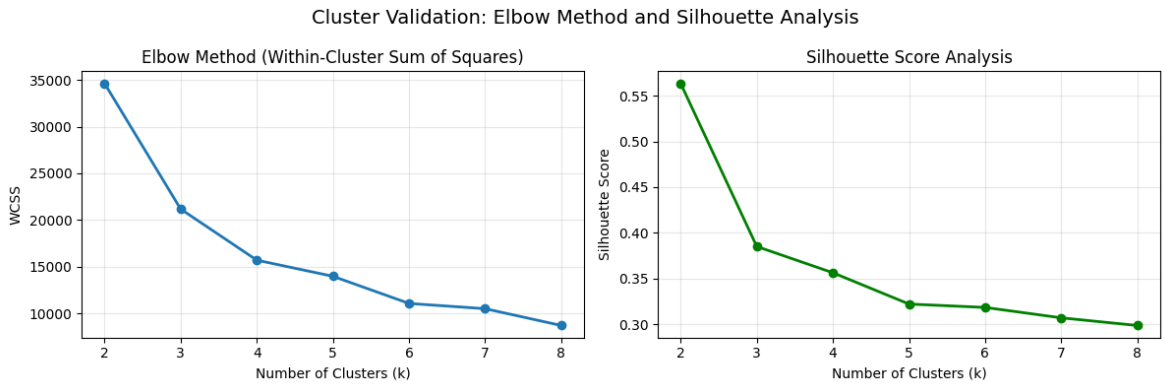


Figure 34: Metrics for selecting the number of clusters for store sales segmentation.

of how stores are grouped in terms of their dominant sales patterns and how clearly separated the clusters are in the reduced feature space.
To interpret the economic and operational meaning of each cluster, a set of store-level summary statistics is computed. These include the mean, standard deviation, minimum, and maximum daily sales for each store across the entire time horizon. These metrics capture both the typical sales level of a store and its variability over time. Cluster-level profiles are then obtained by aggregating these statistics within each cluster. This allows each cluster to be characterized in terms of: the number of stores it contains, average sales intensity, dispersion of sales across stores, and overall sales variability. In addition, the coefficient of variation (CV) is computed for each cluster as the ratio between average variability and average sales. This normalized measure provides insight into the relative stability or volatility of sales within each group, enabling comparisons across clusters with different sales scales.

5.4.2 Results

We first focus on Figure 34, which summarizes the selection of the number of clusters using the two validation metrics introduced previously: the Within-Cluster Sum of Squares (WCSS) and the silhouette coefficient. A particular and somewhat non-standard behavior is observed for the silhouette coefficient. As the number of clusters increases, the silhouette score monotonically decreases, contrary to what is typically expected. This pattern suggests that increasing the number of clusters leads to progressively less well-separated groupings, likely due to strong similarities across some stores' sales trajectories and a gradual fragmentation of otherwise coherent groups.
In contrast, the WCSS exhibits the expected behavior: as the number of clusters increases, the total within-cluster variance decreases. A notable reduction is observed when moving from two to three clusters, where WCSS drops from approximately 35,000 to 20,000. Beyond this point, additional clusters provide progressively smaller reductions in within-cluster variance.
Considering both metrics jointly, we select three clusters as a compromise solution. Although this configuration does not maximize the silhouette score, it provides a substantial improvement in WCSS while maintaining interpretability and avoiding exces-

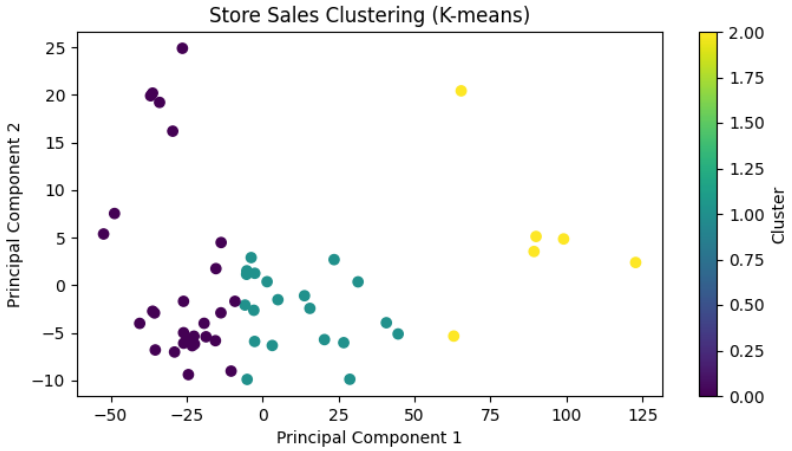


Figure 35: PCA projection of stores clustered by temporal sales patterns.

sive fragmentation of store groups.

After applying K-means with three clusters, Principal Component Analysis (PCA) is used to project the high-dimensional store sales time series into a two-dimensional space for visualization purposes (Figure 35).

The PCA projection reveals that the clustering algorithm is able to separate stores into three clearly distinguishable groups. While some overlap is expected due to the dimensionality reduction and the intrinsic similarity between certain stores, the overall structure shows well-defined regions associated with each cluster. This visual separation supports the validity of the selected clustering solution and confirms that the clusters capture meaningful differences in sales behavior rather than random fluctuations.

Table 4: Sales intensity and variability profiles for each store cluster.

Cluster	Stores	Average	Std Dev	Min	Max	Variability	CV
0	28	6432.33	1999.04	1601.05	9733.78	3572.91	0.56
1	20	13866.27	3362.09	10148.89	21337.96	5305.47	0.38
2	6	30020.98	4400.64	24878.90	36869.09	12538.40	0.42

To characterize the economic meaning of each cluster, Table 4 reports summary statistics computed at the store level and aggregated by cluster. These statistics allow us to interpret each cluster in terms of sales intensity and variability.

Cluster 0: Low-to-Medium Sales, High Relative Variability

Cluster 0 contains 28 stores, making it the largest group. These stores exhibit the lowest average sales level, with a mean daily sales value of approximately 6,432 units. However, this cluster also presents relatively high variability, as reflected by a coefficient of variation (CV) of 0.56, the highest among all clusters. This indicates that stores in this group experience substantial fluctuations in sales relative to their average volume, suggesting demand instability or sensitivity to external factors such as seasonality or local conditions.

Cluster 1: Medium-High Sales, Stable Performance

Cluster 1 includes 20 stores with a significantly higher average sales level of approximately 13,866 units. Compared to Cluster 0, these stores display lower relative variability, with a CV of 0.38, indicating more stable and predictable sales patterns. This cluster can be interpreted as a group of well-performing stores with consistent demand, making them suitable candidates for standardized inventory and marketing strategies.

Cluster 2: High Sales, Moderate Variability

Cluster 2 is the smallest group, consisting of only 6 stores, but it clearly represents the top-performing segment. These stores achieve the highest average sales, exceeding 30,000 units, and also exhibit the largest absolute variability. However, when normalized by sales volume, their coefficient of variation (0.42) remains moderate. This suggests that although sales fluctuate in absolute terms, these fluctuations are relatively small compared to the overall scale of demand. These stores likely correspond to flagship or high-traffic locations with consistently strong sales performance.

6 Unit 6. Density-Based Clustering

6.1 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) offers a fundamentally different approach to clustering compared to centroid-based methods like K-means. While K-means relies on the distance of points from a central average—forcing clusters to be spherical and requiring every point to belong to a group, DBSCAN defines clusters based on the density of data points in the feature space. The algorithm operates using two primary user-defined parameters: Epsilon (ϵ), which specifies a search radius around a point, and $MinPts$, which sets the minimum number of points required within that radius to consider a region "dense".
As the video highlights, DBSCAN presents significant distinct advantages over traditional partitioning methods, particularly in scenarios involving complex data structures:

- **Detection of Arbitrary Shapes:** One of the most distinct benefits of DBSCAN is its ability to identify non-convex clusters. Traditional algorithms like K-means assume clusters are spherical due to their reliance on Euclidean distance from a centroid. In contrast, by following density continuity, DBSCAN can successfully model complex, non-linear geometric shapes, such as concentric rings or elongated "snake-like" structures, which partitioning methods would fail to separate correctly.

- **Automatic Determination of Cluster Count:** Unlike K-means or Hierarchical Clustering, which often require the user to specify the number of clusters (k) in advance, DBSCAN does not rely on a pre-imposed count. Instead, the number of clusters is determined naturally by the structure of the data itself, governed by the density parameters ϵ (Epsilon) and $MinPts$. This allows the algorithm to adapt to the inherent complexity of the dataset.

- **Robustness to Outliers (Noise Handling):** In centroid-based models, every data point must be assigned to a cluster, which means significant outliers can distort the centroid position and reduce clustering accuracy. DBSCAN explicitly handles anomalies by classifying points located in low-density regions as noise (typically labeled as -1). By discarding these outliers rather than forcing them into a group, the resulting clusters remain purer and more representative of the core data distribution.

The performance and topological accuracy of the DBSCAN algorithm are intrinsically tied to the precise calibration of its two governing hyperparameters, Epsilon (ϵ) and Minimum Points ($MinPts$), as these determine the density thresholds required to distinguish meaningful structures from background noise.

Implementation

To systematically evaluate this sensitivity and identify the optimal configuration, a controlled experiment was implemented. First, a synthetic dataset comprising 300 samples

distributed across 4 distinct centers was generated to establish a ground truth. Crucially, the data was then normalized using StandardScaler, ensuring that the distance calculations are not skewed by varying feature magnitudes. The experimental design involves a comprehensive grid search, defining a spectrum of ϵ values ranging from tight (0.1) to loose (0.8) neighborhoods, and varying $MinPts$ thresholds between 3 and 10.

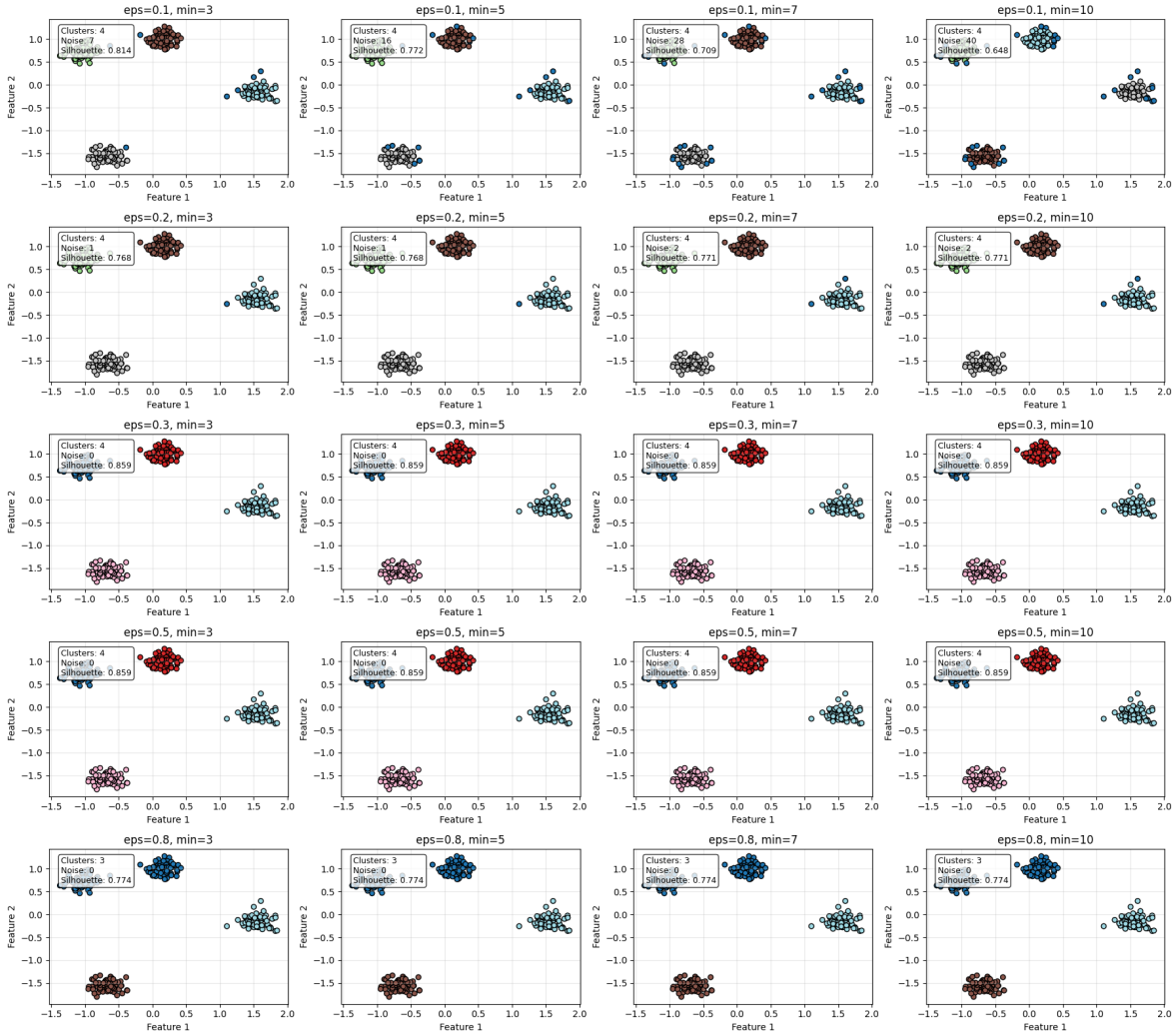


Figure 36: Grid search evaluating DBSCAN parameters ϵ and $MinPts$.

As observed in Figure 36, an extensive range of hyperparameter combinations has been explored to determine the optimal configuration for the DBSCAN algorithm. The results of this grid search highlight the critical sensitivity of the model to the Epsilon (ϵ) parameter, while showing relative stability with respect to MinPts.

The analysis reveals that ϵ values of 0.3 and 0.5 yielded the highest Silhouette Scores. This success can be attributed to the topological match between these radii and the natural scale of the dataset. An ϵ in this range is sufficiently large to bridge the gaps between neighboring points within the same cluster (ensuring continuity) but sufficiently small to avoid bridging the gap between distinct clusters (preventing merging). Conversely, $\epsilon = 0.1$ provided the poorest performance. At this restrictive threshold, the

radius is likely smaller than the average distance between points within a single blob. This causes "over-fragmentation", where a single natural cluster is shattered into many micro-clusters or, more drastically, the majority of valid data points are isolated and incorrectly classified as noise because they fail to find neighbors within such a tight vicinity.Impact of Minimum

On the other hand, the results regarding $MinPts$ are relatively consistent across the tested range. This behavior is characteristic of datasets with high, uniform density like the generated blobs. Once ϵ is correctly tuned to capture the local neighborhood, the core point condition is robustly satisfied whether the threshold is 3 or 10. While higher values of $MinPts$ generally produce more noise-resistant clusters (smoothing out anomalies), in this clean synthetic environment, the structural definition was primarily driven by the spatial reach (ϵ) rather than the density threshold.

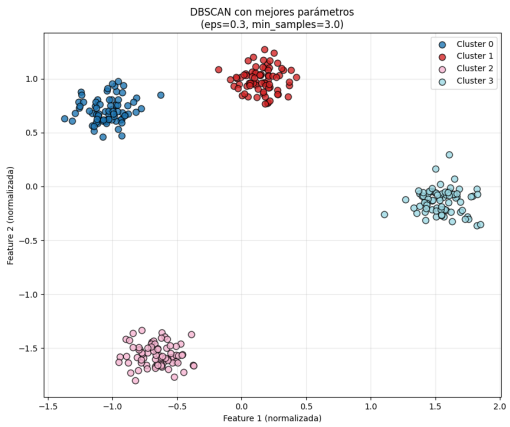


Figure 37: Optimal DBSCAN clustering achieving perfect separation of synthetic clusters.

The grid search identified the combination of $\epsilon = 0.3$ and $MinPts = 3$ as the global optimum. This configuration achieved a Silhouette Score of 0.859 and successfully identified a total of 4 clusters with 0 noise points. As confirmed visually in Figure 37, this parameter set allows the algorithm to perfectly recover the underlying structure of the data. The absence of noise points indicates that the chosen ϵ was inclusive enough to capture even the peripheral points of the blobs, while the perfect cluster count (4) demonstrates that it successfully avoided bridging the separate distributions.

7 Unit 7. Dimensionality Reduction Techniques

Dimensionality reduction represents a key methodology in machine learning and data analysis aimed at simplifying high-dimensional datasets while preserving their most informative characteristics. High-dimensional data often suffers from issues such as noise, redundancy, and the "curse of dimensionality", which can hinder learning algorithms and visualization efforts. Dimensionality reduction techniques transform the original feature space $X \in R^{N \times D}$ into a lower-dimensional representation $Z \in R^{N \times d}$, where $d < D$, in a way that retains the essential structure, variance, or information content of the data.
These methods are particularly valuable for:

- Improving model performance by removing irrelevant or redundant features.
- Facilitating visualization and interpretability in lower-dimensional spaces.
- Enhancing computational efficiency for subsequent analyses.

Dimensionality reduction techniques can be broadly categorized into linear methods, which assume linear relationships between variables, and non-linear methods, which capture complex, manifold-like structures in the data. Among the most widely used approaches are Exploratory Factor Analysis (EFA), Principal Component Analysis (PCA), and t-distributed Stochastic Neighbor Embedding (t-SNE).

7.1 Exploratory Factor Analysis (EFA)

Exploratory Factor Analysis is a statistical method used to uncover latent, or hidden, factors that explain correlations among observed variables. In many cases, it is not possible to measure certain underlying concepts directly, such as social class or risk attitudes. Instead, these latent factors are inferred through related observable variables, for example, occupation, educational background, or property ownership.
EFA aims to describe the relationships between observed variables in terms of a smaller set of underlying factors. By doing so, it reduces redundancy, highlights shared variance, and facilitates interpretation. The method estimates how strongly each observed variable is associated with each latent factor, while accounting for unique variance that is specific to each variable. Factor solutions are often rotated to achieve a simpler and more interpretable structure, with rotations designed to either produce uncorrelated factors (orthogonal rotation) or allow correlated factors (oblique rotation).
EFA can be applied in an exploratory sense, where the goal is to identify potential latent structures without imposing strict assumptions, or in a confirmatory manner, where a predefined factor model is tested against the data. This flexibility makes EFA widely used in fields such as psychology, social sciences, and market research, where researchers seek to identify meaningful constructs underlying observed patterns.

Implementation

To illustrate the application of Exploratory Factor Analysis, we use the classic Iris dataset, which is widely employed in machine learning and statistics for testing and

demonstration purposes. The dataset contains measurements of four continuous variables for 150 iris flowers, corresponding to three different species: Iris setosa, Iris versicolor, and Iris virginica. The variables measured are: Sepal length (cm), Sepal width (cm), Petal length (cm) and Petal width (cm).
In this context, the measurements of the flowers serve as the observed variables, while the species themselves could be thought of as underlying latent groupings, though in EFA we do not use the species label to fit the factors. This makes the Iris dataset suitable for factor analysis because the observed variables are correlated, which allows us to explore the potential latent structure.
We implement EFA with the following configuration:

- Number of factors: 2, chosen to capture the main underlying patterns in the data.
- Rotation method: Varimax, which simplifies the factor structure and improves interpretability by maximizing the variance of the squared loadings for each factor.
- Estimation method: Maximum likelihood (ml), which allows for statistical testing of the factor solution.

The analysis begins by fitting the factor model to the four numeric features of the dataset. After fitting, we extract: Factor loadings, which quantify how strongly each observed variable is associated with each latent factor; Communalities, which represent the proportion of variance in each observed variable explained by the extracted factors; and Factor scores, which provide a lower-dimensional representation of each observation in the space of the latent factors, useful for visualization and further analysis.

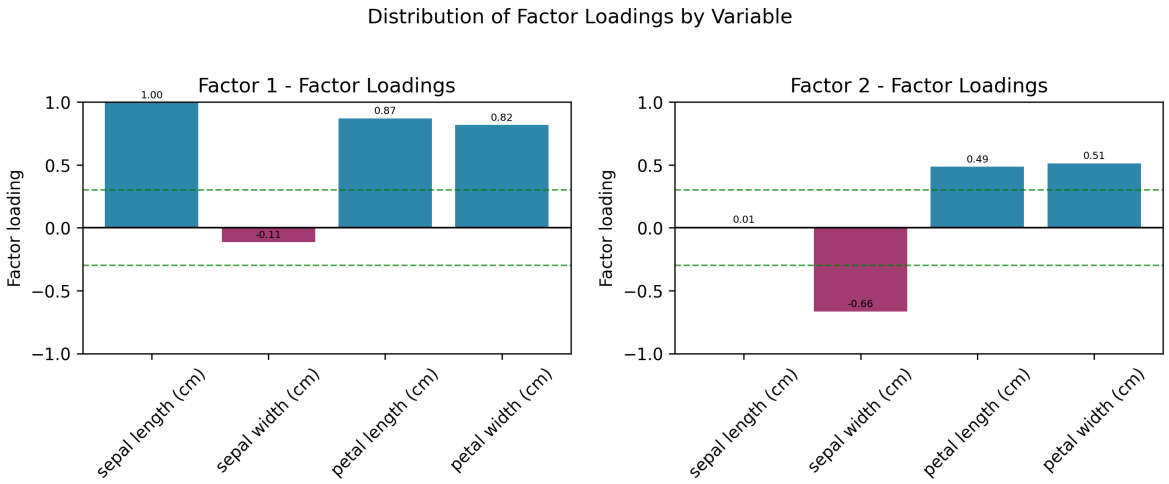


Figure 38: Factor loadings from Exploratory Factor Analysis on the Iris dataset.

In the first stage of the analysis, we examine the factor loadings obtained from the EFA model, which quantify the contribution of each observed variable to the extracted factors. In Figure 38, the loadings for each variable on each factor are displayed.
For Factor 1, the two petal measurements exhibit high positive loadings, with petal length and petal width showing values of approximately 0.82 and 0.87, respectively.

This indicates that Factor 1 is strongly associated with the variation in petal characteristics, and these variables contribute most significantly to this factor. Sepal length also shows a substantial positive loading close to 1, suggesting that it shares some variance with the petal variables and participates in defining Factor 1. In contrast, sepal width presents a small negative loading of around -0.11, indicating that it contributes very little to this factor and its variance is largely independent of the petal-related dimension captured by Factor 1.

In Factor 2, the pattern differs. Sepal width exhibits a moderately strong negative loading of approximately -0.66, highlighting its primary contribution to this factor, while sepal length has a negligible loading of around 0.01, suggesting it does not significantly influence Factor 2. Interestingly, the two petal variables maintain moderate positive loadings of around 0.5, implying that they still retain some shared variance with Factor 2, albeit less than with Factor 1. This dual contribution indicates that petal measurements influence multiple dimensions of variation in the dataset, reflecting their overall importance in differentiating among observations.

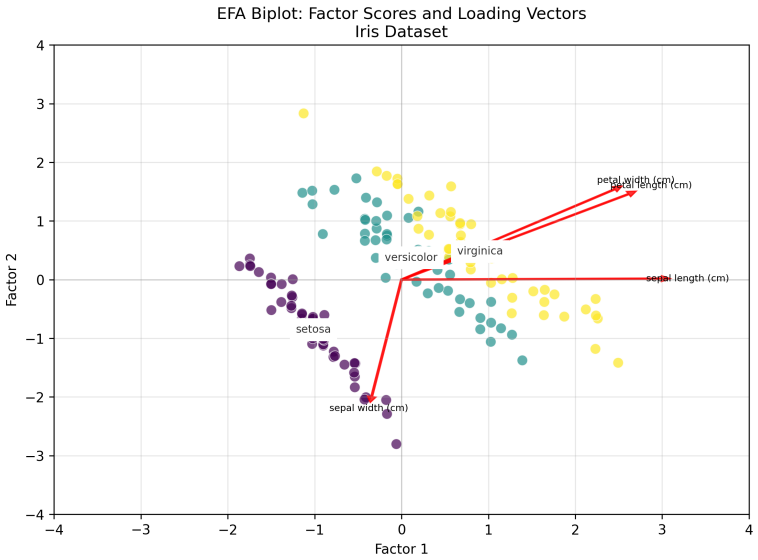


Figure 39: Biplot visualization of Iris species in the latent factor space.

Moving to the biplot visualization (Figure 39), the three iris species are clearly separated into distinct clusters along the two-factor space defined by the EFA model. Iris setosa is positioned in the lower-left quadrant, reflecting its relatively low values on the factor associated with petal dimensions (Factor 1) and its sepal characteristics. This aligns with the previous loading analysis, as setosa has smaller petal lengths and widths, which correspond to lower Factor 1 scores.

The other two species, Iris versicolor and Iris virginica, are located toward the central and right regions of the biplot, with a tendency to occupy higher values on Factor 1, consistent with their larger petal measurements. The separation along Factor 2 further distinguishes these species, primarily due to differences in sepal width, which is strongly associated with Factor 2. This demonstrates how the latent factors identified by EFA capture meaningful dimensions of variability in the dataset, effectively summarizing

multivariate relationships and facilitating the differentiation of species.

7.2 Principal Component Analysis (PCA)

Principal Component Analysis is a linear dimensionality reduction method that transforms a high-dimensional dataset into a smaller set of principal components, which are linear combinations of the original variables. Unlike factor analysis, PCA does not aim to uncover latent causes or underlying constructs; its primary goal is to retain as much variance as possible while simplifying the data.

The first principal component captures the direction of maximum variance in the data, and each subsequent component captures the remaining variance while being orthogonal to the previous components. By projecting the data onto the first few components, PCA produces a lower-dimensional representation that summarizes the most relevant information. This property is particularly useful for identifying dominant patterns, reducing noise, and improving computational efficiency for machine learning models.

PCA is sensitive to the scale of the variables, so standardization is often performed before analysis to ensure that all variables contribute equally. The technique is widely used in exploratory data analysis, image and signal processing, and as a preprocessing step for machine learning algorithms. While PCA components may not have direct interpretability, they provide a mathematically optimal summary of the data, and selecting the top components based on variance explained allows researchers to retain the most informative dimensions while reducing complexity.

In comparison to EFA, PCA focuses on maximizing total variance without assuming hidden causes, whereas EFA seeks to model latent factors that explain correlations between variables. PCA constructs components sequentially and deterministically, and the choice of how many components to retain is often guided by a scree plot or the cumulative variance explained. This makes PCA particularly effective for reducing dimensionality in large datasets while preserving global structure.

Implementation

To implement Principal Component Analysis, we again use the Iris dataset, which contains measurements of four variables—sepal length, sepal width, petal length, and petal width—for 150 flowers across three species: Iris setosa, Iris versicolor, and Iris virginica. The dataset is suitable for PCA because the variables are correlated, allowing for meaningful dimensionality reduction.

Before applying PCA, the data is standardized to ensure that all variables contribute equally to the analysis, given that they are measured on different scales. Standardization centers each variable around zero and scales it to unit variance, preventing variables with larger ranges from dominating the principal components.

The PCA implementation is configured to extract all principal components initially. The explained variance ratio is computed for each component, and the cumulative variance is also calculated to determine how many components are required to capture the majority of the information in the dataset. The result is a set of principal components,

each representing a linear combination of the original variables, which maximizes the variance in the data sequentially.

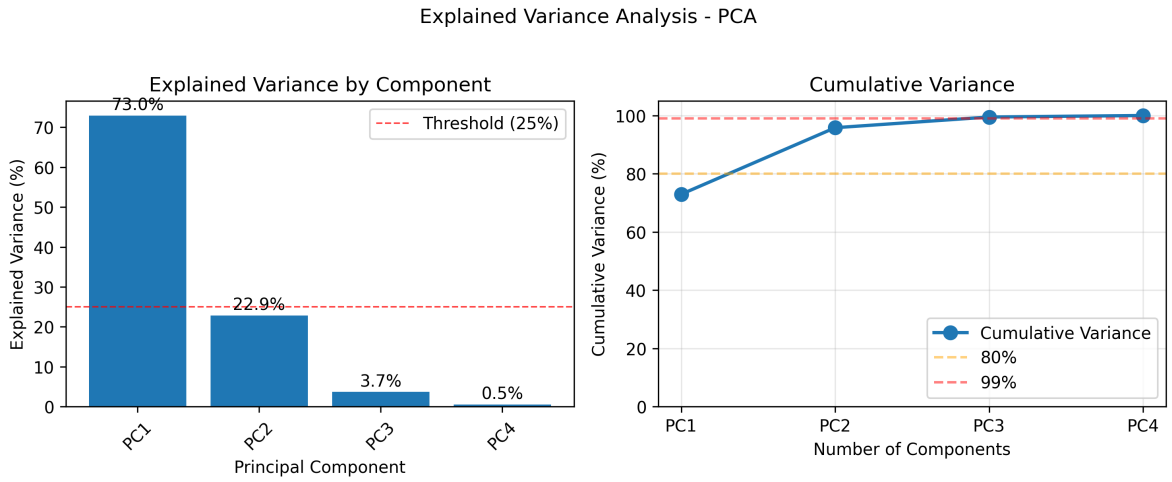


Figure 40: Explained variance and cumulative variance for PCA components.

The first step in analyzing PCA results is to examine the variance explained by each component. As shown in Figure 40, the first principal component captures approximately 73% of the total variance in the dataset, making it by far the most informative component. This indicates that the primary direction of variability in the Iris dataset is strongly aligned with the first component, largely influenced by the variables with the highest variation, particularly the petal measurements.

The second component explains an additional 22.9% of variance, which is also significant. Together, the first two components account for over 95% of the total variance, demonstrating that the dimensionality of the original four-variable dataset can be effectively reduced to two dimensions without losing substantial information. From a practical standpoint, this means that most of the variability and structure in the Iris dataset can be captured and visualized using just these two components, simplifying interpretation while retaining essential patterns.

The cumulative explained variance curve highlights that adding further components beyond the first two yields diminishing returns, as the additional variance explained by the third and fourth components is minimal. This reinforces the decision to focus on the first two principal components for visualization and further analysis.

To explore the relationships among observations and variables in the reduced space, a biplot is generated using the first two principal components (Figure 41). The biplot reveals that PCA effectively captures the structure of the dataset: the three species form distinct clusters, demonstrating that the first two components summarize the essential differences in flower measurements.

Iris setosa is clearly separated from the other two species, located in the lower-left quadrant of the biplot. This aligns with the lower values of petal length and width observed in setosa, corresponding to lower scores on the first principal component. In contrast, Iris versicolor and Iris virginica occupy the central and right portions of the plot, reflecting their larger petal dimensions. There is some overlap between versicolor

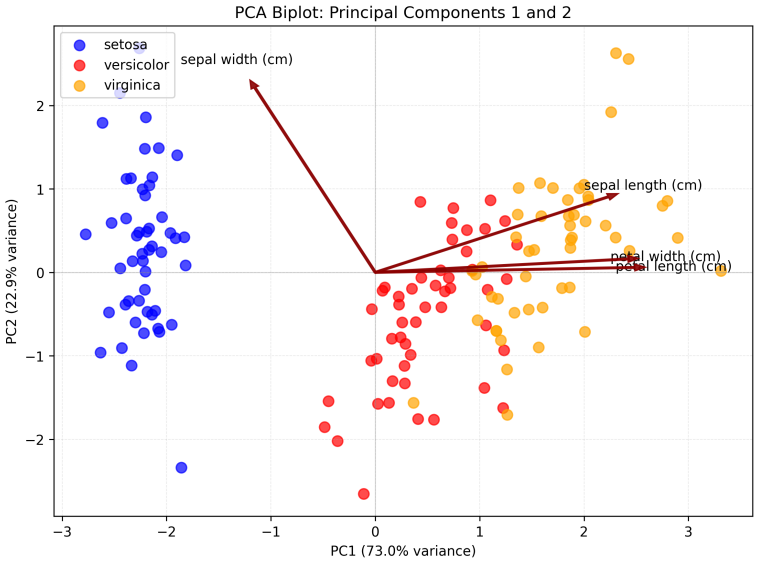


Figure 41: PCA biplot showing variable contributions and species separation.

and virginica, indicating that these species are more similar in the measured features, particularly in petal width and length, which dominate Factor 1.

Examining the directions of the variable vectors in the biplot provides further insight. Petal length and petal width are strongly aligned with the first principal component, indicating that they are the primary contributors to the separation along this axis. Sepal length also contributes positively but to a lesser extent. The placement of versicolor and virginica along this direction confirms their larger petal measurements compared to setosa.

The second principal component is primarily associated with sepal width, which points toward the upper-left quadrant where setosa is located. This suggests that sepal width is a key variable distinguishing setosa from the other species along the second dimension. While sepal length and the petal measurements also contribute slightly to PC2, their influence is less pronounced, which explains why the main separation between versicolor and virginica occurs primarily along PC1.

Overall, the PCA biplot demonstrates that two principal components are sufficient to capture the main structure of the Iris dataset, highlighting both the dominant contribution of petal measurements to species differentiation and the supporting role of sepal width in distinguishing setosa. This combination of explained variance and biplot visualization provides a clear understanding of the relationships among variables and species in the dataset.

7.3 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a non-linear dimensionality reduction method primarily used for visualizing high-dimensional data in two or three dimensions. Unlike PCA, which preserves global variance, t-SNE emphasizes the preservation of local structures, meaning that points that are similar in high-dimensional space are placed close together in the low-

dimensional representation, while dissimilar points are pushed apart.
The algorithm begins by projecting data randomly into a low-dimensional space and then iteratively adjusts the positions of the points to reflect pairwise similarities in the original space. This approach makes t-SNE particularly effective at revealing clusters, patterns, and manifold structures that are not captured by linear methods. However, the resulting axes in t-SNE plots do not have inherent mathematical meaning, and the technique is computationally intensive, making it more suitable for visualization than for feature extraction or predictive modeling.
t-SNE is widely used in applications such as genomics, natural language processing, and image analysis to explore complex, high-dimensional datasets and gain intuitive insights into underlying patterns. It complements linear methods like PCA by focusing on local neighborhoods rather than global variance.

Implementation

To illustrate the application of t-SNE, we use the Digits dataset, which consists of 1,797 grayscale images of handwritten digits from 0 to 9. Each image is represented as a vector of 64 features, corresponding to the 8x8 pixel intensity values. The target variable indicates the digit class, but t-SNE is applied in an unsupervised manner, meaning that the labels are not used during the embedding.
Before applying t-SNE, a comparison with PCA is also included, using two principal components as a baseline linear dimensionality reduction method. For t-SNE, we configure the algorithm to generate a 2D embedding, initializing the positions using PCA, setting a perplexity of 40, and a fixed random state for reproducibility. The resulting embedding is a two-dimensional representation in which the algorithm attempts to preserve local neighborhood relationships from the original 64-dimensional space.

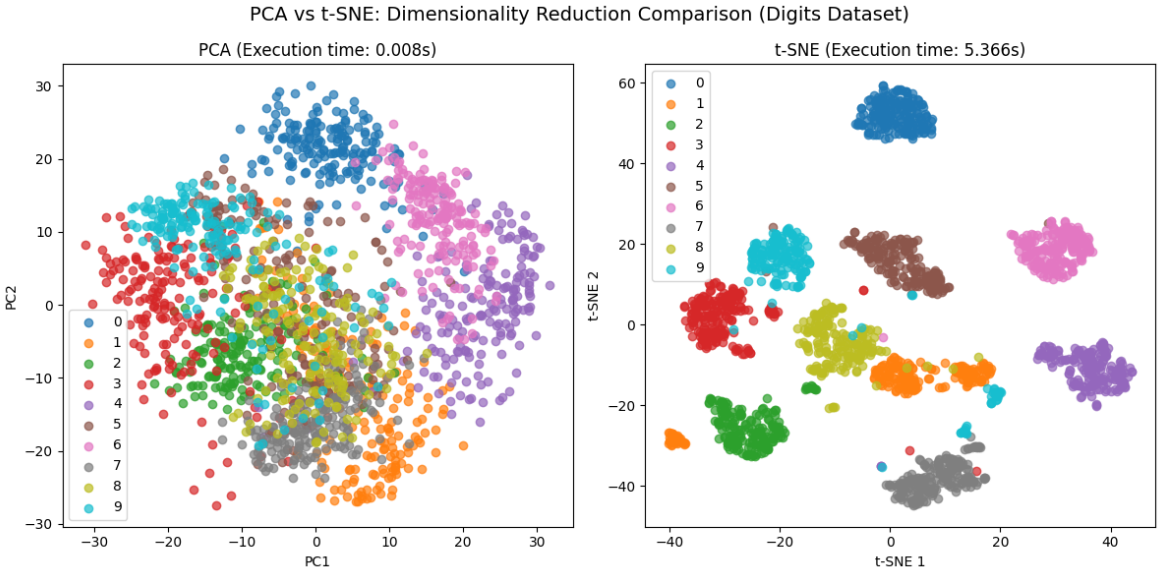


Figure 42: Comparison of PCA and t-SNE visualizations on the Digits dataset.

Figure 42 displays the 2D representations produced by PCA and t-SNE. The t-SNE

embedding shows a nearly perfect separation of all digit classes, with each cluster corresponding clearly to a single digit. This remarkable separation is a result of t-SNE's ability to preserve local similarities: points that are close together in the high-dimensional pixel space remain close in the 2D embedding, while points from different clusters are pushed apart. This allows t-SNE to reveal the intrinsic manifold structure of the data, effectively forming well-defined clusters corresponding to each digit.
In contrast, the PCA projection on the left panel shows substantial overlap among classes. While the first two principal components capture the directions of maximal variance, this linear transformation cannot account for the complex non-linear relationships between pixel intensities, leading to superimposed clusters and poor class separation. This illustrates a key difference between linear and non-linear dimensionality reduction: PCA retains global variance but is insufficient for resolving intricate local structures, whereas t-SNE excels at visualizing local relationships.
However, this improved performance comes at a computational cost. As indicated in the figure titles, t-SNE requires approximately 5.37 seconds to compute the embedding, while PCA takes only about 0.008 seconds. The higher execution time of t-SNE is due to the iterative optimization process, in which the algorithm repeatedly adjusts the positions of all points in the low-dimensional space to minimize the divergence between probability distributions representing pairwise similarities in the high-dimensional and low-dimensional spaces. This iterative, non-linear process is computationally intensive, particularly as the number of observations increases, which contrasts with the deterministic and algebraic computation of PCA.
In summary, t-SNE provides an exceptionally informative visualization of the digits dataset, effectively uncovering clusters that correspond to individual classes. PCA, while faster and suitable for capturing global variance trends, is unable to separate complex, non-linearly distributed data in the same manner.

7.4 Project - Face Recognition

Dimensionality reduction plays a crucial role in many real-world data science and machine learning applications, particularly in domains where data is inherently high-dimensional. Examples include face recognition, topic extraction from text corpora, and recommendation systems. In such contexts, reducing dimensionality is not only essential for computational efficiency, but also for extracting the most informative patterns from complex data.
This project focuses on the application of Principal Component Analysis (PCA) to the task of face recognition. Face images are typically represented as high-dimensional vectors, where each pixel corresponds to a feature. As a result, even a modestly sized image dataset can quickly become computationally expensive and difficult to analyze directly. PCA provides an effective solution by projecting face images onto a lower-dimensional subspace that preserves the most relevant variability across individuals.
The objective of this project is to extend and improve a base implementation of PCA to address a concrete identification problem. Given a dataset of facial images and a new photograph of an individual who is already present in the dataset, the goal is to identify the person in the new image by comparing it against the existing faces. This

comparison is performed in the reduced PCA space, where distances between images reflect meaningful similarities while filtering out noise and redundant information.
By leveraging PCA, the project demonstrates how dimensionality reduction can be used not only for visualization or data compression, but also as a core component of a practical recognition system. The following sections will describe the dataset, the implementation of PCA for face representation, and the identification strategy based on similarity in the reduced feature space.

7.4.1 Implementation

To implement face recognition using PCA, we rely on the Olivetti Faces dataset, which consists of grayscale images of human faces, each resized to a fixed resolution of 64×64 pixels. Each image is therefore represented as a high-dimensional vector of 4,096 pixel intensity values. This high dimensionality makes the dataset particularly suitable for demonstrating the effectiveness of PCA in compressing and representing facial information.
The dataset is divided into a training set and a test set. The training set is used to learn the PCA representation, while the test set is reserved for evaluating reconstruction quality and identification performance. This separation ensures that the PCA model captures general facial structures rather than memorizing individual images. By fitting PCA exclusively on the training faces, we simulate a realistic scenario in which the system must recognize previously seen individuals from new, unseen images.
A critical aspect of PCA-based face recognition is the choice of the number of components to retain. Retaining too few components leads to excessive information loss, while retaining too many undermines the benefits of dimensionality reduction. To address this trade-off, we analyze the cumulative explained variance as a function of the number of components.
The cumulative variance curve shows how much of the total variability in facial images is preserved as more components are included. Thresholds such as 80% and 99% explained variance are commonly used as reference points. This analysis provides an empirical basis for selecting a suitable number of components that balances compression and information retention. In this project, a reduced set of components is chosen to retain most of the facial structure while significantly lowering dimensionality.
Once the number of components is fixed, PCA is fitted to the training images, projecting each face into a lower-dimensional space commonly referred to as the eigenface space. Each face is now represented by a compact vector of PCA coefficients rather than thousands of pixel values. This transformation reduces noise and highlights the dominant patterns that characterize facial variation across individuals.
Both training and test images are projected into this PCA space. Face identification is then performed by comparing the PCA representation of a test image with those of the training images, using a distance-based similarity measure. The closest match in the reduced space is taken as the predicted identity.
To evaluate how well PCA preserves facial information, the reduced representations are projected back into the original image space to obtain reconstructed faces. Comparing the reconstructed images with the original ones provides qualitative insight into how

facial details are preserved as dimensionality is reduced.
In addition to visual inspection, reconstruction quality is assessed quantitatively using the mean squared reconstruction error (MSE). This error measures the average discrepancy between the original and reconstructed pixel values. By computing the MSE for different numbers of PCA components, we observe how reconstruction accuracy improves as more components are retained. The resulting error curve illustrates the diminishing returns of adding additional components and provides further guidance for selecting an optimal dimensionality.

7.4.2 Results

The PCA-based face recognition system is trained using 350 facial images, while the remaining 50 images are reserved for testing. This split ensures that the PCA model is learned exclusively from the training data and evaluated on unseen images, allowing for a realistic assessment of both reconstruction quality and identification performance.

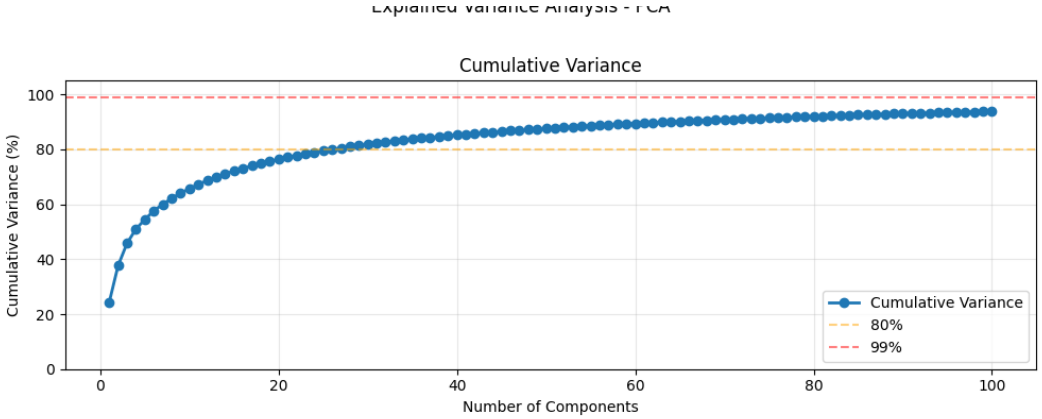


Figure 43: Cumulative explained variance curve for PCA on facial images.

The first step in the analysis focuses on determining an appropriate number of principal components using the cumulative explained variance curve. As shown in Figure 43, the cumulative variance increases rapidly for the first components, reflecting that a large proportion of the facial variability is captured by a relatively small number of dimensions.
Around 25 to 30 components, the cumulative explained variance exceeds the commonly used threshold of 80%, indicating that most of the relevant information in the facial images has already been retained. Beyond this point, the curve exhibits a nearly asymptotic behavior, where each additional component contributes only marginal gains in explained variance. When increasing the number of components to 100, the total explained variance reaches approximately 93.9%, confirming that the remaining components mostly capture fine-grained details or noise rather than essential identity-related information.
This pattern suggests that retaining more than 30 components yields diminishing returns in terms of information preservation, while significantly increasing model complexity.

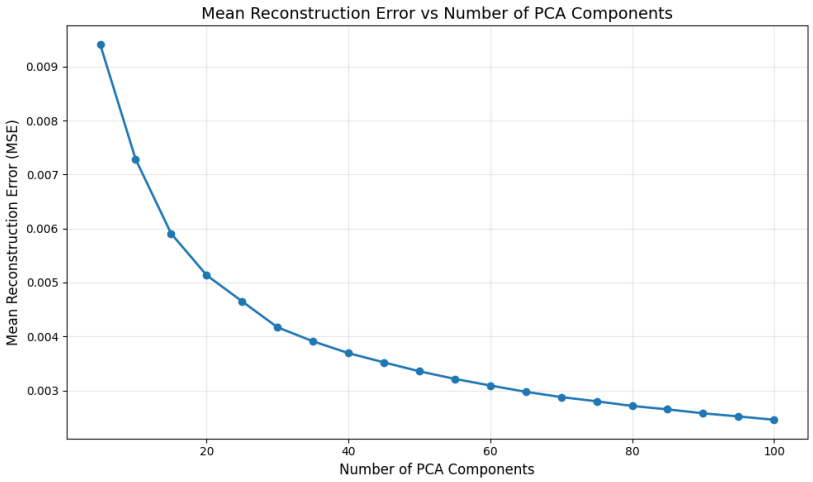


Figure 44: Reconstruction error as a function of the number of PCA components.

To complement the variance-based analysis, the mean squared reconstruction error (MSE) is examined as a function of the number of PCA components, as shown in Figure Y. When using only 5 components, the reconstruction error is relatively high, exceeding 9×10^{-3}, indicating substantial information loss and poor reconstruction quality.

As the number of components increases, the reconstruction error decreases rapidly, reaching values close to 4×10^{-3} around 30 components. This represents a reduction of more than half of the initial reconstruction error. Beyond this point, however, further increases in the number of components lead to only marginal improvements, with the error decreasing slowly toward approximately 3×10^{-3}.

The strong similarity between the explained variance curve and the reconstruction error curve reinforces the conclusion that around 30 principal components provide an effective balance between dimensionality reduction and information preservation. This convergence of both criteria supports the selection of 30 components for the face identification task.

The identification performance of the PCA-based system is illustrated in Figure 45, where three randomly selected test images are compared against their nearest neighbors in the PCA space. The system correctly identifies two out of the three faces, demonstrating that the reduced PCA representation retains sufficient identity-related information for reliable recognition.

The misclassification occurs in the first test image, where the true subject corresponds to label 0, but the system incorrectly predicts subject 23. When examining the reconstruction errors for these cases, it is observed that the reconstruction error remains relatively low for the correctly identified faces, while it is noticeably higher for the misclassified image. This indicates that the identification error is closely linked to the quality of the PCA reconstruction.

The misidentification can be attributed to information loss during dimensionality reduction, particularly the loss of subtle facial features that differentiate visually similar individuals. When these discriminative details are not adequately preserved in the re-

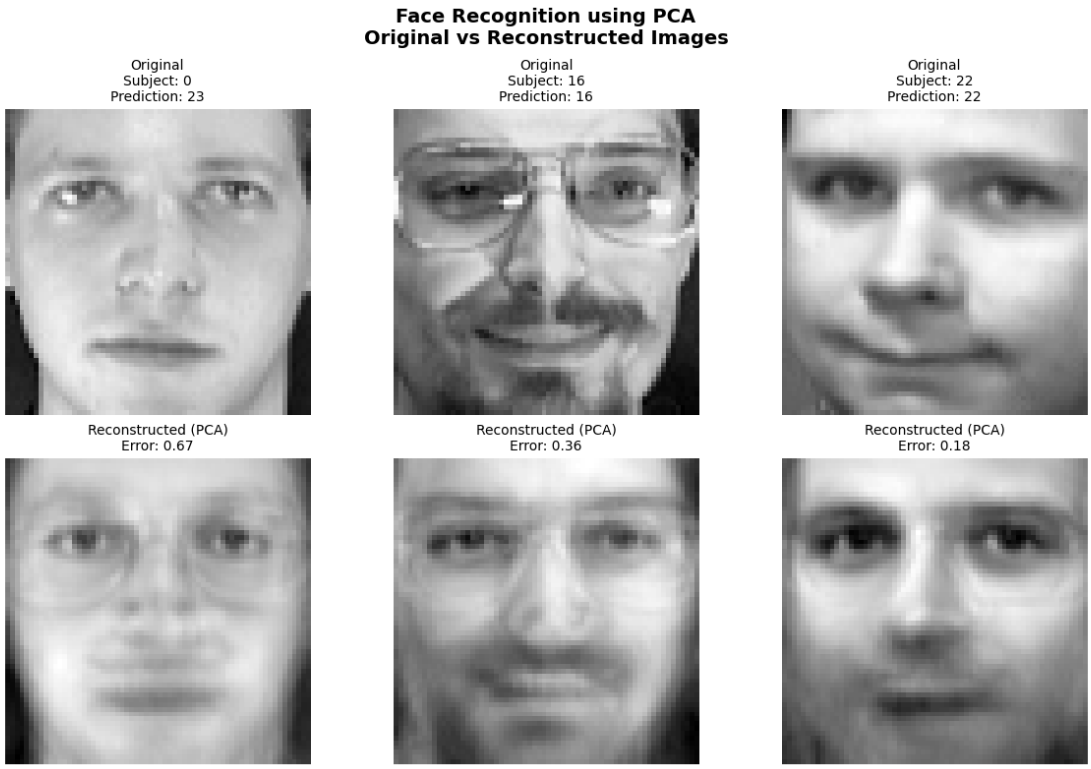


Figure 45: Face identification results using PCA-based nearest neighbor matching.

duced PCA space, the reconstructed representation becomes closer to that of another subject, leading to incorrect nearest-neighbor matching.

Overall, these results highlight the trade-off inherent in PCA-based face recognition systems: while dimensionality reduction enables efficient and effective identification, excessive compression can remove fine-grained identity cues. The chosen configuration with 30 components strikes a reasonable balance, achieving strong reconstruction quality and accurate identification in most cases, while maintaining a compact and computationally efficient representation.

8 Unit 8. Time Series Analysis

8.1 Store Sales Time Series

8.1.1 Preprocessing

The data set used consists of daily sales of different product categories across multiple stores over a five-year period, totaling 1,684 daily records. Each record corresponds to the total sales in dollars recorded on a specific day for a given store and category. Since the objective of this section is to perform a time series analysis, the daily data from all stores and categories were aggregated into a single daily time series, considering only the total sales per day across all stores. This transformation allows for the analysis of the overall sales dynamics over time and facilitates the identification of general patterns and trends.

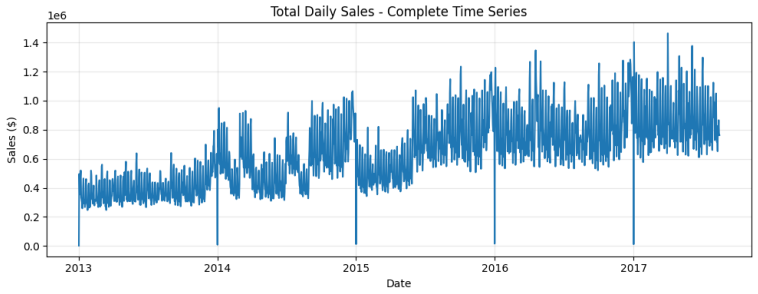


Figure 46: Daily time series of total sales across all stores.

As can be seen in Figure 46, the daily series presents data recorded for each day, showing a high density of measurements. However, the complexity and intrinsic variability of the data make it difficult for simple models to accurately capture the underlying behaviour. The daily series shows considerable daily fluctuations due to factors such as variations in demand, promotions, holidays, and specific events for each store or category, which generates noise in the analysis.

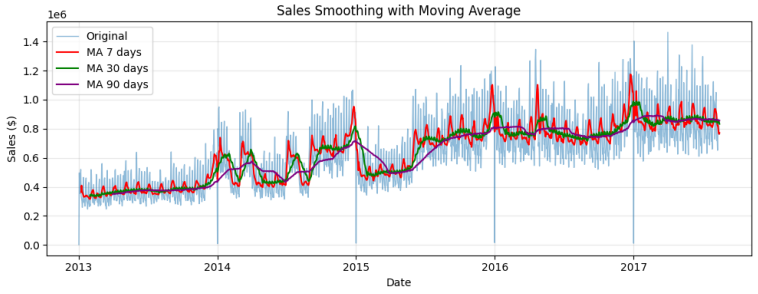


Figure 47: Smoothing of the daily series using moving averages of 7, 30, and 90 days.

To explore the structure of the data, smoothing was applied using a moving average, as shown in Figure 47. This procedure helps reduce short-term variability and highlight broader trends in the data. Despite the smoothing, the daily series still presents peaks and valleys that reflect the complex nature of daily commercial activity.

In order to simplify the analysis and facilitate modeling, it was decided to aggregate the daily series to a monthly level, calculating the total sales per month. This aggregation offers several advantages: it reduces daily noise, highlights more stable behavioral patterns, and allows models to identify trends and cycles more easily. As observed in Figure 48, the monthly series shows a smoother and more predictable pattern, with less abrupt cycles that reflect sales seasonality and sustained growth trends over time.

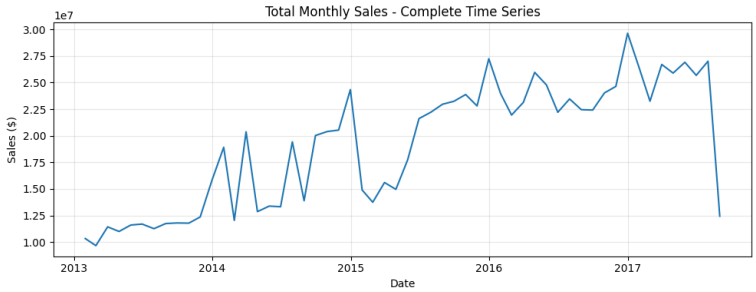


Figure 48: Monthly series of aggregate sales.

Additionally, an analysis of the time series trend was performed using linear regression methods applied to the monthly series. This analysis reveals sustained sales growth over the studied period, with an estimated slope of $293,677.37$ additional dollars per month, indicating that, on average, sales increase steadily each month. This information is crucial, as it allows for anticipating the future behavior of the series and serves as a reference for forecasting models that incorporate trend components.

8.1.2 Decomposition

Time series decomposition is a technique used to break down a series into its fundamental components in order to better understand its underlying patterns. By separating the observed series into trend, seasonal, and residual components. This decomposition is useful as it allows to address each component individually and improve forecast accuracy.

- **Additive Decomposition**: In an additive model, the observed value of the series is assumed to be the sum of its components:

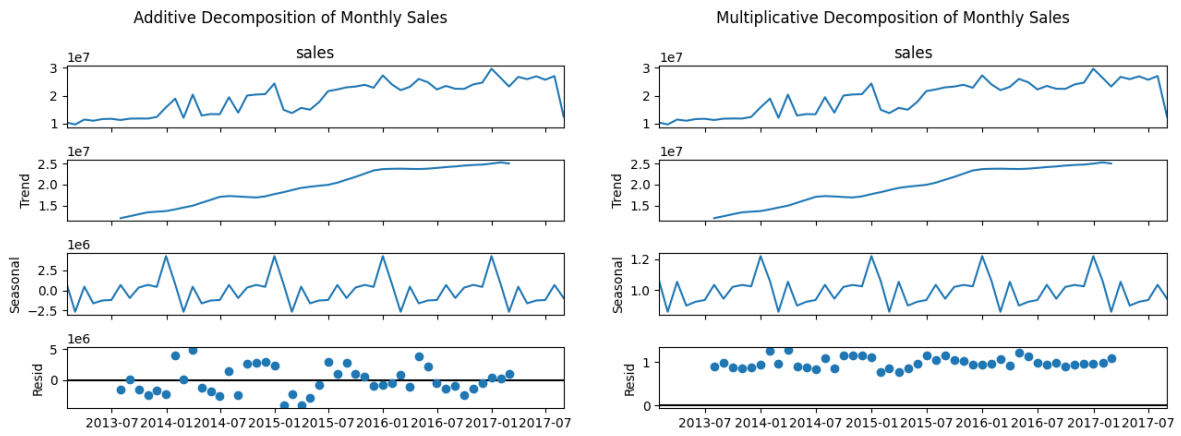
$$y_t = T_t + S_t + R_t,$$

  where T_t represents the trend component, S_t the seasonal component, and R_t the residual (or random) component.

- **Multiplicative Decomposition**: In a multiplicative model, the observed value is assumed to be the product of its components:

$$y_t = T_t \times S_t \times R_t.$$

  This approach is appropriate when the seasonal variations or residuals increase or decrease proportionally with the level of the series.



(a) Additive decomposition of the monthly sales series.

(b) Multiplicative decomposition of the monthly sales series.

Figure 49: Decompositions of the monthly sales series.

Following the previous data, both additive and multiplicative decompositions were applied to the monthly sales series. As shown in Figure 49.a, the additive decomposition clearly separates the series into trend, seasonal, and residual components, capturing the underlying growth pattern and the repeating seasonal fluctuations. On the other hand, the multiplicative decomposition, illustrated in Figure 49.b, models the series by considering proportional effects, which highlights how seasonal variations increase with the overall level of sales.
Despite the differences in approach, both decompositions successfully break down the time series into its main components, allowing a clearer understanding of the trend, seasonality, and residual variability. This confirms that either method can be effectively used to analyze and model the underlying structure of the sales data.

8.1.3 Forecasting

Exponential smoothing is a widely used technique for forecasting time series, which assigns exponentially decreasing weights to past observations. This approach allows the model to give more importance to recent data while still considering historical patterns. There are several variants of exponential smoothing, depending on whether the model accounts for trend and seasonality:

- **Single Exponential Smoothing**: This method only captures the level of the series and does not account for trend or seasonality. The forecast is calculated using a smoothing parameter α ($0 < \alpha < 1$), which determines how quickly the weights of past observations decrease. High values of α give more weight to recent observations, while low values smooth the series more heavily.

- **Double Exponential Smoothing (Holt's Method)**: This method extends single exponential smoothing by capturing trend in addition to level. It introduces a second smoothing parameter, β ($0 < \beta < 1$), which controls the smoothing of

the trend component. Like α, β balances sensitivity to recent changes versus stability of the trend estimate.

- **Triple Exponential Smoothing (Holt-Winters Method)**: This method further extends double exponential smoothing to account for both trend and seasonality It can be applied in additive or multiplicative form, depending on whether seasonal effects are roughly constant (additive) or proportional to the level of the series (multiplicative).

Following the approach used with the monthly sales data, the series was split into training and test sets to evaluate forecasting performance. The last twelve months of data (from September 2016 to August 2017) were reserved as the test set, while the first 44 months (from January 2013 to August 2016) were used for training. This split ensures that the model is evaluated on the most recent data while being trained on a sufficiently long historical period to capture trends and seasonal patterns.

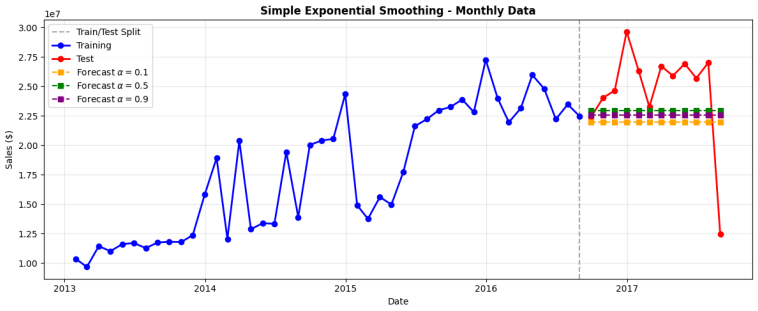


Figure 50: Forecast using Single Exponential Smoothing with various α values.

As shown in Figure 50, the use of Single Exponential Smoothing (SES) produces a relatively simple forecast, essentially a straight line starting from different points depending on the value of α. When α is high, the forecast is closer to the last observed point of the original series. For instance, with $\alpha = 0.9$ the prediction starts almost exactly where the original series ends. Conversely, with $\alpha = 0.1$, the forecast lies further below, reflecting a smoother and more conservative estimate that places less weight on the most recent observation.
In Figure 51, the results from Double Exponential Smoothing (Holt's Method) highlight the effect of accounting for trend. The predictions now incorporate the series' trend, and depending on the values of α and β, the forecast can have a positive or negative slope. Observing the prediction with lower α and β values, the slope is slightly positive, reproducing the overall trend of the series. While the other predictions exhibit a negative slope, this behavior occurs because the trend in the last five observed points (train set) is negative, even though the overall series trend is positive. This demonstrates how Holt's method responds not only to the long-term trend but also to recent changes in the series.
For the Holt-Winters Method, illustrated in Figure 52, the forecast captures both trend and seasonality, providing a more realistic prediction of the series. While the prediction is not perfect, as reflected in the evaluation metrics (MAPE: 17.39%, MAD:

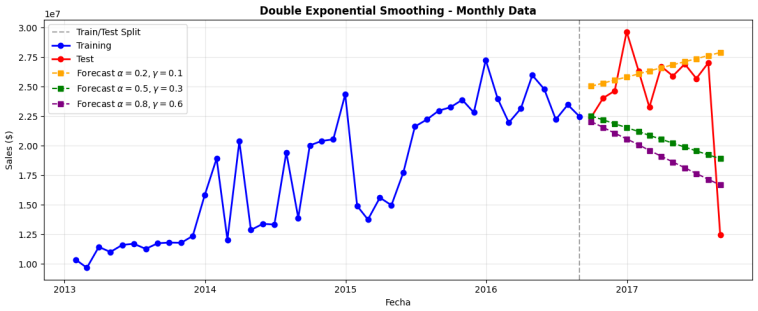


Figure 51: Forecast using Double Exponential Smoothing (Holt's Method) with various α and β values

$\$3,108,348.35$, RMSE: $\$4,807,563.2$), it already reflects the general pattern of the data. The elevated errors are primarily due to a sudden drop in the last point of the test series, which the model could not anticipate because this behavior was not observed in previous data. This anomaly is likely caused by incomplete data for the final month of the series.

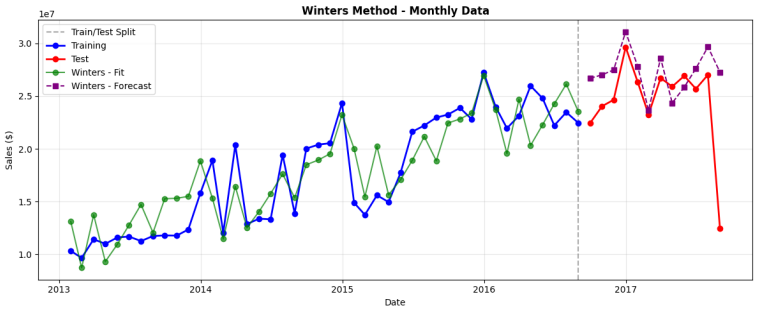


Figure 52: Forecast using Triple Exponential Smoothing (Winters Method).

As expected, the results for the Holt-Winters Method show the model's ability to capture both the trend and seasonal components of the series. The forecast provides a realistic representation of the underlying patterns, smoothing out short-term fluctuations while following the general upward trend and periodic seasonal variations.

8.2 ARIMA and SARIMAX

In this section, we introduce the analysis of time series data using ARIMA and SARIMAX. Time series analysis aims to model temporal dependencies in data, allowing us to forecast future values based on past observations. Unlike standard regression, time series models explicitly account for autocorrelation, trends, and seasonality, which are inherent in temporal data.

8.2.1 Autocorrelation

Autocorrelation Function (ACF) and Correlograms

Autocorrelation measures the degree of similarity between a time series and a lagged

version of itself over successive time intervals. It is a fundamental concept because most time series are not independent over time, past values often influence future values. For example, in financial data, yesterday's stock price is often correlated with today's price.

$$\rho_k = \frac{\text{Cov}(y_t, y_{t-k})}{\sigma^2}$$

The Autocorrelation Function (ACF) plots ρ_k for different lags k. Visualizing the ACF with a correlogram helps identify key patterns in the series. By examining the correlogram, we can detect whether seasonal components must be included in the model.

Partial Autocorrelation Function (PACF)
While the ACF shows the overall correlation between the series and its lags, it cannot differentiate between direct and indirect correlations. For example, if y_t is correlated with y_{t-2} because of the intermediate correlation with y_{t-1}, the ACF will reflect this combined effect. The Partial Autocorrelation Function (PACF) solves this by measuring the correlation between y_t and y_{t-k} after removing the effect of all intermediate lags. The PACF is particularly useful for selecting the order p in an autoregressive (AR) model, as it shows where the direct influence of past values cuts off.

8.2.2 ARIMA Models

Autoregressive (AR) Models
Autoregressive models predict the current value of a series as a linear combination of its previous values plus a stochastic error. An AR model of order p ($AR(p)$) is defined as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2).$$

Here, the coefficients $(\phi_1, \ldots, \phi_p)$ quantify the influence of past observations. AR models assume that past behavior contains information about the future.

Moving Average (MA) Models
Moving Average models represent the current value as a function of past forecast errors, capturing short-term shocks in the series:

$$y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2).$$

MA models are particularly effective when the series is affected by sudden but temporary disturbances. The MA coefficients $(\theta_1, \ldots, \theta_q)$ determine how past errors influence current observations.

ARIMA Models
ARIMA models combine three key components to model time series data. First, the autoregressive (AR) component uses past values of the series to predict the current value. Second, the moving average (MA) component accounts for past forecast errors to adjust predictions. Finally, differencing (I for Integration) is applied to remove trends and achieve stationarity in non-stationary series.

This combination allows ARIMA models to capture both short-term fluctuations and longer-term patterns. The choice of the model parameters (p, d, q) is guided by analyzing autocorrelation and partial autocorrelation plots and testing for stationarity.

8.2.3 Seasonal ARIMA (SARIMA) Models

SARIMA models extend ARIMA to handle seasonal patterns in time series data. In addition to the autoregressive, moving average, and differencing components of ARIMA, SARIMA incorporates seasonal autoregressive, seasonal moving average, and seasonal differencing terms. These seasonal components allow the model to capture patterns that repeat over fixed intervals. By including seasonality explicitly, SARIMA improves forecast accuracy for series with predictable repetitive behavior.

SARIMAX Models
SARIMAX further extends SARIMA by allowing the inclusion of exogenous variables, which are external factors that can influence the series. By incorporating these additional predictors, SARIMAX can account for both the temporal and seasonal structure of the series as well as external influences, providing more accurate and robust forecasts for real-world applications.

9 Course Feedback

From a personal perspective, this course has been a very positive learning experience. Given my academic background as a graduate in Data Science, many of the core concepts covered throughout the course, such as supervised and unsupervised learning, model evaluation, and basic statistical learning techniques, were already familiar to me. This prior knowledge significantly facilitated my understanding of the material and allowed me to engage with the content in a more critical and reflective manner.

Despite this initial familiarity, I consider the course to be well-structured, comprehensive, and intellectually stimulating. Rather than merely reinforcing known concepts, it provided deeper insights into both the theoretical foundations and the practical implications of statistical learning methods. In particular, the course succeeded in connecting the theoretical content with real-world applications, which greatly enhanced my overall understanding.

Among the topics covered, I found the section on dimensionality reduction techniques especially interesting. Beyond their technical implementation, the emphasis on how these methods contribute to data interpretability and explainability was particularly valuable. Understanding how techniques such as PCA or factor analysis can be used not only to reduce complexity but also to uncover latent structures in data added an important analytical dimension to the course.

Additionally, the part dedicated to time series forecasting captured my interest, especially in relation to modeling temporal dynamics and extracting meaningful patterns from sequential data. This topic complemented the rest of the course well and highlighted the relevance of statistical learning methods in practical forecasting scenarios.

Overall, even with a strong prior background, the course enabled me to consolidate existing knowledge, gain new perspectives, and deepen my interest in key areas of statistical learning.

References

Grasas, A., Juan, A. A., Faulin, J., De Armas, J., & Ramalhinho, H. (2017). Biased randomization of heuristics using skewed probability distributions: A survey and some applications. *Computers & Industrial Engineering*, *110*, 216–228.