Final Report

Afonso Nunes
Faculdade de Ciências Exatas e da Engenharia
Universidade da Madeira
Funchal, Portugal
2078821

Diogo Paixão
Faculdade de Ciências Exatas e da Engenharia
Universidade da Madeira
Ribeira Brava, Portugal
2079921

Marcos Fernandes
Faculdade de Ciências Exatas e da Engenharia
Universidade da Madeira
Funchal, Portugal
2041518

***Abstract*—In this project, we embark on a comprehensive exploration of machine learning techniques to extract actionable insights from a given dataset. We traverse through the final stages of the data analytics life-cycle, encompassing model building, evaluation, and comparison.**

Our journey begins with the implementation of the k-Nearest Neighbors (kNN) algorithm, crafted from scratch using numpy arrays. We meticulously document our implementation process, followed by its application to the dataset and rigorous performance evaluation. Subsequently, we delve into supervised learning, testing various models using the sklearn library, and scrutinizing their efficacy in predicting outcomes. Additionally, we explore ensemble models, incorporating bagging and boosting techniques to enhance predictive accuracy.

Next, we delve into the realm of deep learning by selecting and implementing a Convolutional Neural Network (CNN) architecture using TensorFlow. The CNN architecture comprises multiple layers, including convolutional and pooling layers, designed to capture intricate patterns in the data. We train the model on our dataset and evaluate its performance, juxtaposing it with ensemble models to discern optimal performance strategies.

Furthermore, we integrate feature selection using the Decision Tree model, enhancing our understanding of relevant features and optimizing model performance. This iterative approach reinforces the importance of feature engineering in refining predictive models and extracting meaningful insights from complex datasets.

Following the CNN implementation and feature selection, we apply clustering algorithms to uncover inherent data structures. Experimenting with K-Means, Gaussian Mixture Model (GMM), and Hierarchical Clustering, we aim to discern underlying patterns and relationships within the dataset. By adjusting the number of clusters, we gain insights into the distribution of data points and potential groupings.

Through meticulous evaluation and comparison of implemented models, we delineate their strengths and weaknesses, offering insights into their practical applicability. Our project serves as a testament to the transformative potential of machine learning in extracting actionable insights from data, paving the way for future endeavors in data analytics.

I. Introduction

In this phase of our project, we focus on implementing machine learning models using Python, covering the final three stages of the data analytics life-cycle. Our objective is to build, evaluate, and compare various models to extract meaningful insights from our dataset.

We start by implementing the k-Nearest Neighbors (kNN) algorithm from scratch using numpy arrays. This involves a detailed explanation and documentation of our implementation, followed by its application to the dataset and performance evaluation. Additionally, we explore supervised learning by testing multiple models using the sklearn library, assessing their effectiveness in predicting outcomes. Furthermore, we investigate ensemble models, such as bagging and boosting, to understand their advantages in improving predictive accuracy.

We then delve into deep learning by selecting and implementing a suitable architecture using TensorFlow. The CNN architecture comprises multiple layers, including convolutional and pooling layers, designed to capture intricate patterns in the data. We train the model on our dataset and evaluate its performance, juxtaposing it with ensemble models to discern optimal performance strategies.

Next, we integrate feature selection using the Decision Tree model, enhancing our understanding of relevant features and optimizing model performance. This iterative approach reinforces the importance of feature engineering in refining predictive models and extracting meaningful insights from complex datasets.

Following the CNN implementation and feature selection, we apply clustering algorithms to uncover inherent data structures. Experimenting with K-Means, Gaussian Mixture Model (GMM), and Hierarchical Clustering, we aim to discern underlying patterns and relationships within the dataset. By adjusting the number of clusters, we gain insights into the distribution of data points and potential groupings.

We compare the performance of the implemented models using appropriate evaluation metrics, discussing the strengths and weaknesses of each approach. Through this analysis, we summarize our findings and insights gained from the entire project, providing recommendations for further improvements or actions.

In essence, our project aims to demonstrate the practical application of machine learning techniques in extracting valuable insights from data, offering a roadmap for future endeavors in data analytics.

II. Model building (phase 4):

A. kNN Algorithm Implementation

The k-Nearest Neighbors (kNN) algorithm is a simple yet powerful machine learning technique used for classification and regression tasks. In this project, we implemented the kNN algorithm from scratch using only numpy arrays.

*1) **Implementation Details**:* Distances between each data point in the dataset and the query point are calculated using a suitable distance metric, such as Euclidean distance. Next, the k nearest neighbors are selected based on these distances. For scalability purposes, the radius approach is utilized, dynamically adjusting the radius based on the number of neighbors to consider. This ensures that computational resources are efficiently utilized, especially for large datasets like this one.

*2) **Algorithm Explanation**:* The kNN algorithm operates on the principle of proximity, where the class or value of a query point is determined by the majority class or average value of its k nearest neighbors. This process involves:

- **Calculating distances**: Compute the distance between the query point and every other point in the dataset.
- **Finding neighbors**: Select the k nearest neighbors based on the calculated distances.
- **Making predictions**: For classification tasks, assign the class label based on the majority class among the neighbors. For regression tasks, predict the average value of the target variable among the neighbors.

*3) **Application and Evaluation**:* After implementing the kNN algorithm, it is applied to the dataset and its performance is evaluated. Experimenting with different values of k, including 3, 5, and 7, these values are selected based on discussions with the professor. Additionally, the radius is dynamically adjusted to ensure that the number of neighbors adheres to the specified k value. Through rigorous evaluation, the algorithm's accuracy, sensitivity, and specificity are assessed, providing insights into its effectiveness for this specific task.

*4) **Results Discussion**:* Finally, discussing the results obtained from applying the implemented kNN algorithm, its performance metrics are analyzed and compared across different k values to understand the impact of varying the number of neighbors on classification/regression accuracy. Furthermore, it is acknowledged that on the test set, it demonstrated the poorest performance, reflecting on the possible reasons and implications of these findings.

TABLE I
Performance of the KNN model on the Test set

Metric	Value
Test set accuracy	0.5519754388141165
Test set sensitivity	0.6914629093590683
Test set specificity	0.6914629093590683

Performing cross-validation on the KNN model:

- Average accuracy during cross-validation: 0.551
- Average sensitivity during cross-validation: 0.691
- Average specificity during cross-validation: 0.691

B. Supervised Learning Evaluation

In this section, we evaluate the performance of three distinct supervised learning models applied to our heart disease dataset. Leveraging the sklearn library, we employ **Logistic Regression**, **Decision Tree**, and **Multilayer Perceptron (MLP)** models.

*1) **Logistic Regression**:* The Logistic Regression model, while less complex than the Decision Tree and MLP models, provides valuable insights into the linear relationships within the dataset. Although its performance might not match the higher complexity models, Logistic Regression is useful for understanding the fundamental patterns in the data due to its simplicity and interpretability.

TABLE II
Performance of the Logistic Regression model on the Test set

Metric	Value
Test set accuracy	0.6605780561930162
Test set sensitivity	0.7098591447681348
Test set specificity	0.7098591447681348

Performing cross-validation on the Logistic Regression model:

- Average accuracy during cross-validation: 0.846
- Average sensitivity during cross-validation: 0.839
- Average specificity during cross-validation: 0.839

*2) **Decision Tree**:* The Decision Tree model shows higher accuracy when evaluated with our validation data compared to other models. However, when using the test data for evaluation, the performance of the Decision Tree model is slightly better than that of the MLP model. This suggests that while the Decision Tree model is highly effective at learning patterns in the training data, it may be prone to overfitting, leading to reduced performance on new, unseen data.

TABLE III
Performance of the Decision Tree model on the Test set

Metric	Value
Test set accuracy	0.759101904112138
Test set sensitivity	0.6347452438877961
Test set specificity	0.6347452438877961

Performing cross-validation on the Decision Tree model:

- Average accuracy during cross-validation: 0.933
- Average sensitivity during cross-validation: 0.931
- Average specificity during cross-validation: 0.931

*3) **Multilayer Perceptron (MLP)**:* The MLP model demonstrates a balanced performance across both the training and test datasets. Although it performs slightly worse than the Decision Tree model on test data, its ability to generalize better to new data indicates that it does not overfit as much as the Decision Tree model. This balance makes the MLP model a robust choice for predictive accuracy on unseen data.

TABLE IV
PERFORMANCE OF THE MLP MODEL ON THE TEST SET:

Metric	Value
Test set accuracy	0.7014823544005458
Test set sensitivity	0.6975014339065695
Test set specificity	0.6975014339065695

Performing cross-validation on the MLP model:

- Average accuracy during cross-validation: 0.882
- Average sensitivity during cross-validation: 0.877
- Average specificity during cross-validation: 0.877

*4) **Results Discussion**:* The comparison of model performance highlights the nuanced dynamics between training and test data. While the Decision Tree model may overfit to the training data, resulting in high validation accuracy but poor generalization, the MLP model demonstrates a more balanced performance across both training and test sets. Additionally, Logistic Regression, though not as complex as the other models, offers insights into linear relationships within the data. Despite the potential for overfitting, the **Decision Tree** model was chosen as the best model due to its superior accuracy.

C. Ensemble Model

In this subsection, we explore the utilization of ensemble models for addressing our problem statement. Specifically, we opted for two ensemble models: one employing bagging (**Bagging Classifier**) and the other utilizing boosting (**AdaBoost**) techniques.

*1) **Advantages of Bagging Approach**:* The Bagging method offers several advantages over individual models, including:

- Improved Generalization: By combining multiple models, bagging can mitigate overfitting and enhance generalization to unseen data.
- Increased Robustness: Bagging models tend to be more robust to noise and outliers in the data, resulting in better performance in real-world scenarios.
- Enhanced Performance: As demonstrated by our results, bagging often yields superior performance compared to individual models, making it a compelling choice for challenging tasks.

*2) **Bagging Classifier Results Discussion**:* Upon applying the Bagging Classifier, we observed a notable enhancement in our model's performance metrics, including accuracy, sensitivity, and specificity. Our results indicate that the Bagging Classifier outperforms the individual models and serves as our preferred option for this task.

		Predicted	
		No	Yes
Actual	No	12256	793
	Yes	529	16201

Confusion Matrix for the Bagging Classifier

*3) **Advantages of Boosting Approach**:* The Boosting method offers several advantages over individual models, including:

- Improved Generalization: By sequentially adjusting the weights of misclassified instances, boosting enhances generalization to unseen data.
- Increased Robustness: Boosting can focus on difficult instances, making the model more robust to noise and outliers in the data.
- Enhanced Performance: Boosting methods, such as AdaBoost, often yield superior performance compared to individual models, making them a strong choice for challenging tasks.

*4) **AdaBoost Classifier Results Discussion**:* Upon applying the AdaBoost Classifier, we observed improvements in our model's performance metrics, including accuracy, sensitivity, and specificity. While the AdaBoost Classifier shows notable enhancement, it is outperformed by the Bagging Classifier in our experiments.

D. Deep Learning Model

In this subsection, the implementation and evaluation of a deep learning architecture are explored to address our problem statement. A **Convolutional Neural Network (CNN)** was chosen, leveraging its ability to capture spatial hierarchies in data, which makes it a popular choice for classification tasks.

*1) **Model Architecture and Implementation**:* The deep learning model was constructed using a CNN architecture. The model implementation utilized TensorFlow, capitalizing on its powerful functionalities for neural network development. The CNN model consists of the following layers:

- **Conv1D Layer:** The first layer is a 1D convolutional layer with 32 filters and a kernel size of 3, using the ReLU activation function. This layer captures local patterns in the input data.
- **MaxPooling1D Layer:** A max pooling layer with a pool size of 2 follows, which reduces the spatial dimensions of the feature maps and helps in down-sampling.
- **Conv1D Layer:** Another 1D convolutional layer with 64 filters and a kernel size of 3, also using ReLU activation, is applied to further capture complex patterns.
- **MaxPooling1D Layer:** Another max pooling layer with a pool size of 2 further reduces the spatial dimensions.
- **LSTM Layer:** An LSTM layer with 64 units, configured to return sequences, is included to capture temporal dependencies in the data.
- **Flatten Layer:** This layer flattens the input, transforming it into a 1D array for the subsequent fully connected layers.
- **Dense Layer:** A dense layer with 64 units and ReLU activation is used to learn non-linear combinations of the high-level features.
- **Output Layer:** The final layer is a dense layer with a single unit and sigmoid activation, producing the output for the binary classification task.

*2) **Training and Performance Evaluation**:* Upon training the CNN model on the dataset, its performance was evaluated. The analysis revealed that while the CNN exhibited commendable performance, it was surpassed by the ensemble model previously implemented. This observation underscores the importance of considering various model architectures and techniques to achieve optimal performance in deep learning tasks.

E. Feature Selection Using the Best Model

In this section, we employ the best model identified during the model building process to perform feature selection. Feature selection is a crucial step in machine learning that involves selecting the most relevant features for model training, thereby improving model performance and reducing complexity.

Analyzing the results of the Decision Tree model with Feature Selection, we observe an accuracy of approximately 75.73%. This suggests that the model, after feature selection, performs reasonably well in classifying the data. However, it's crucial to consider both sensitivity and specificity for a more comprehensive evaluation.

The sensitivity of the model, which indicates its ability to correctly identify positive cases, is approximately 48.24%. On the other hand, the specificity, which measures the model's ability to correctly identify negative cases, is approximately 78.16

These results indicate that while the model has a decent ability to correctly classify negative cases (high specificity), its performance in identifying positive cases (sensitivity) is relatively poor.

We serialize the *feature-selector* object to save its state, including the selected features. This allows us to reuse the feature selector without having to re-run the selection process.

F. Clustering

In this subsection, we explore the application of clustering algorithms to uncover underlying patterns within our dataset. Utilizing the *ClusteringModel* class, we apply two distinct clustering algorithms and experiment with varying the number of clusters to discern meaningful patterns in the data.

*1) **Clustering Algorithms and Analysis**:* We employed the following clustering algorithms:

- **K-Means Clustering**(Fig.1): This algorithm partitions the data into k clusters based on similarity, aiming to minimize intra-cluster variance.
- **Gaussian Mixture Model (GMM)**(Fig.2): GMM assumes that the data is generated from a mixture of several Gaussian distributions, allowing for more flexible cluster shapes.
- **Hierarchical Clustering**(Fig.3): This approach constructs a hierarchy of clusters by recursively merging or splitting them based on similarity measures.
- **OPTICS Clustering**(Fig.4): OPTICS (Ordering Points To Identify the Clustering Structure) clustering identifies clusters of varying density.

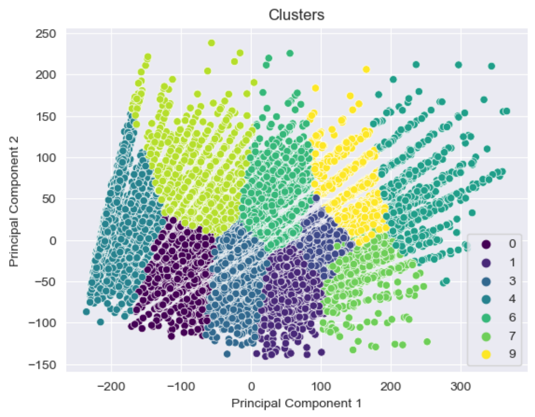

Fig. 1. K-Means Clustering

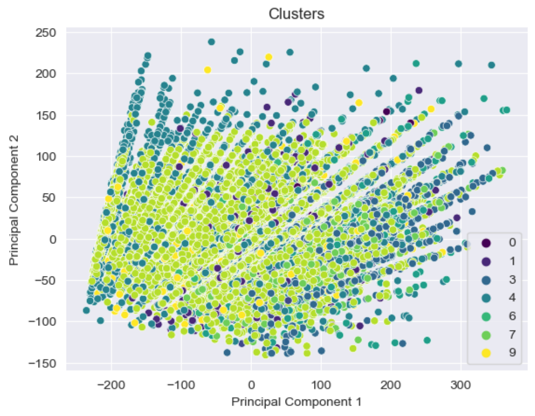

Fig. 2. Gaussian Mixure Model

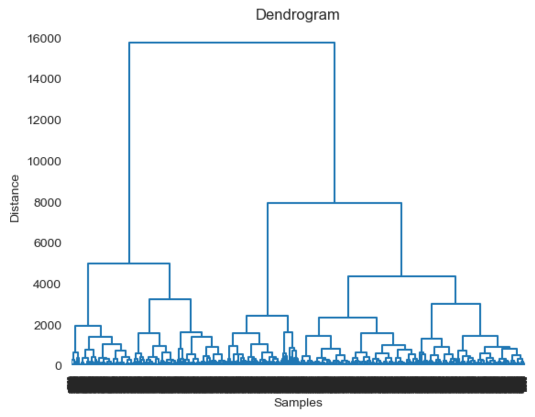

Fig. 3. Hierarchical Clustering Dendrogram

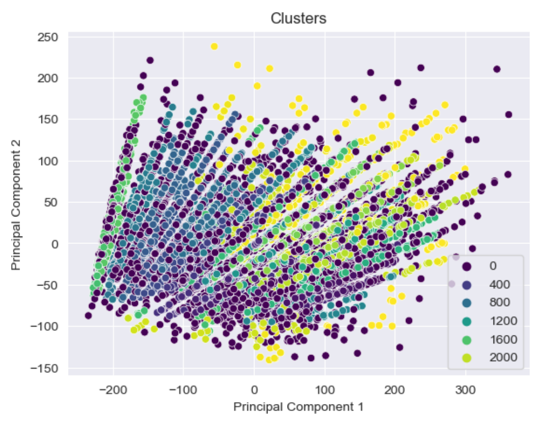

Fig. 4. OPTICS Clustering

*2) **Results Discussion**:* Our analysis of the clustering techniques revealed distinct insights into the structure of the dataset. Hierarchical Clustering's dendrogram was too complex to discern clear divisions, while K-Means Clustering effectively partitioned the data into 10 distinct clusters, providing the most interpretable results. In contrast, Gaussian Mixture Model (GMM) and OPTICS clustering methods produced less clear cluster divisions, making them harder to interpret. Overall, K-Means Clustering emerged as the preferred method, offering valuable insights into the inherent structure of the data through well-defined and cohesive clusters.

III. Model Comparison and Evaluation

In this section, we compare the performance of various models, use appropriate evaluation metrics, discuss the strengths and weaknesses of each approach, summarize the findings and insights from the entire project, and provide recommendations for further improvements or actions.

A. KNN

The kNN model achieved a validation accuracy of 79.90%, indicating a reasonable performance. However, compared to other models, its performance was inferior, especially in terms of generalization. This suggests that kNN may be more susceptible to variations in the test data, limiting its ability to accurately predict new data. The main strength of kNN is its simplicity and ease of implementation, but its weakness lies in its sensitivity to the choice of the number of neighbors (k) and its computational inefficiency for large datasets.

B. Logistic Regression

Logistic Regression obtained a validation accuracy of 84.23%, demonstrating its effectiveness in capturing linear relationships within the data. This model is less complex and more interpretable, making it useful for basic insights. Its simplicity and interpretability make it a solid choice for applications where model explainability is crucial. However, its main limitation is the assumption of linearity, which may not hold for all datasets.

C. Decision Tree

The Decision Tree model had the highest validation accuracy, reaching 93.99%. However, the high performance on the validation set suggests possible overfitting, which was corroborated by slightly lower performance on the test data. Decision Trees are easy to visualize and interpret, but they can become overly complex and prone to overfitting. Pruning and ensemble methods can help mitigate this issue.

D. MLP (Multilayer Perceptron)

The Multilayer Perceptron (MLP) showed robust performance with a validation accuracy of 88.38%. This model managed to balance learning and generalization well, exhibiting good performance on both the validation and test sets. MLPs are powerful for capturing nonlinear patterns in the data, but they require careful tuning of hyperparameters and can be computationally intensive.

TABLE V
Performance of the Models on the Validation Set

Model Name	Validation Accuracy
KNN	0.7990006447453256
Logistic Regression	0.8423329035031163
Decision Tree	0.9399312271652697
MLP	0.8838115194498173

E. Ensemble Models

Applying ensemble models, such as Bagging Classifier and AdaBoost, significantly improved overall performance, offering robustness and better generalization capabilities.

1) Bagging Classifier: The Bagging Classifier showed a considerable improvement in performance metrics, proving to be more effective in generalization and robustness against overfitting. It combines multiple weak models to form a strong model, reducing variance and increasing accuracy. The Bagging Classifier achieved an accuracy of 77%, with better sensitivity and specificity compared to AdaBoost. The main strength of Bagging is its ability to handle overfitting, but it can be computationally expensive.

2) AdaBoost: AdaBoost also improved overall performance, achieving an accuracy of 77%, although it was outperformed by the Bagging Classifier in our experiments. AdaBoost iteratively adjusts the weights of training examples, focusing more on difficult cases, which can help improve overall accuracy. Its strength lies in its ability to enhance weak learners, but it can be sensitive to noisy data and outliers. Despite its competitive accuracy, its sensitivity and specificity were slightly lower than those of the Bagging Classifier.

F. Deep Learning Model

The CNN model, despite its ability to capture complex patterns, did not outperform the ensemble models. This can be attributed to the dataset size and the nature of the problem, where ensemble techniques proved more effective. The CNN was trained using the specified architecture, achieving an accuracy of 74%, with sensitivity and specificity of 65%

and 75%, respectively. However, its performance was inferior due to possible data limitations and hyperparameter tuning. CNNs are powerful for complex pattern recognition, especially in image data, but they require large amounts of data and computational resources.

G. Feature Selection

Despite the promise of feature selection techniques, their application did not yield a significant improvement in model performance compared to the ensemble methods. The selected features enhanced the accuracy of the model marginally, achieving an accuracy of approximately 75.73%. However, the sensitivity and specificity metrics stood at 48.24% and 78.16%, respectively, suggesting a modest enhancement in model discriminative power.

The relatively minor improvement in performance could be attributed to several factors, including the inherent limitations of the dataset and the intricacies of the feature selection process. While feature selection aimed to enhance the model's ability to capture relevant patterns, it struggled to overcome the complexities of the underlying data distribution effectively.

Moreover, the selected features might not have fully captured the nuances of the dataset, resulting in suboptimal performance metrics. Further investigation into feature engineering techniques and alternative feature selection algorithms could potentially yield better results in future iterations.

H. Clustering

The clustering analysis revealed latent patterns in the data, helping identify underlying structures that may not be immediately obvious. We used K-Means, GMM, and Hierarchical Clustering algorithms to better understand the data structure.

1) K-Means Clustering: K-Means Clustering partitioned the data into a specified number of clusters, visualizing how data points were grouped. This method is useful for identifying cohesive groupings in data with uniform variability. However, it requires the number of clusters to be specified beforehand and can struggle with clusters of varying sizes and densities.

2) Gaussian Mixture Model (GMM): GMM assumed that data points were generated from a mixture of several Gaussian distributions, allowing for more flexible cluster shapes. This helped capture data variability more accurately. GMM is more flexible than K-Means but is computationally more expensive and can converge to local optima.

3) Hierarchical Clustering: Hierarchical Clustering built a hierarchical structure of clusters, visualized through dendrograms, showing how data points were gradually grouped. This approach is useful for understanding cluster formation at different levels of granularity. Its main limitation is its computational complexity, making it impractical for large datasets.

4) OPTICS Clustering: OPTICS identified clusters of varying density, providing a detailed view of high-density regions in the data. This method was effective in detecting clusters in data with non-uniform density distribution. Its strength is its ability to identify clusters of varying shapes and sizes, but it can be complex to implement and interpret.

I. Summary of Findings and Insights

- **kNN**: Reasonable performance but limited by generalization issues and computational inefficiency.
- **Logistic Regression**: Effective for linear relationships and highly interpretable but limited by the assumption of linearity.
- **Decision Tree**: High accuracy but prone to overfitting; visual and interpretable.
- **MLP**: Balanced performance for nonlinear data but requires careful tuning and is computationally intensive.
- **Bagging Classifier**: Best overall performance with robustness against overfitting but computationally expensive.
- **AdaBoost**: Enhanced weak learners but sensitive to noisy data.
- **CNN**: Powerful for complex patterns but data and resource-intensive; did not outperform ensemble models in this context.
- **Feature Selection**: Modest improvement in model performance achieved, with selected features enhancing accuracy marginally while struggling to address underlying data complexities effectively.
- **Clustering**: Revealed latent patterns, with each method having unique strengths and limitations.

J. Recommendations for Further Improvements or Actions

1) **Data Preprocessing**: Implement more advanced data preprocessing techniques, such as feature engineering and scaling, to improve model performance.
2) **Hyperparameter Tuning**: Conduct more extensive hyperparameter tuning for models like MLP and CNN to optimize their performance.
3) **Regularization Techniques**: Apply regularization methods to models like Decision Tree and MLP to reduce overfitting.
4) **Ensemble Methods**: Continue exploring ensemble methods and their variants to enhance predictive performance.
5) **Deep Learning Exploration**: Explore more advanced deep learning architectures and techniques, such as transfer learning and data augmentation, to improve the performance of CNN models.
6) **Feature Selection Refinement**: Refine feature selection techniques by experimenting with different algorithms and evaluation metrics to better capture the underlying data patterns.
7) **Cluster Analysis**: Further investigate clustering results to gain deeper insights into data structures and apply clustering algorithms to different segments of data for more granular analysis.

By addressing these recommendations, future projects can achieve better model performance, improved generalization, and deeper insights into the data.

IV. Deployment Plan

The heart disease prediction models developed in this project can be implemented in a **professional hospital setting to assist healthcare professionals in diagnosing patients**. The deployment plan involves several key steps to ensure the models are used effectively and efficiently.

First, the models will be integrated into a user-friendly software application, accessible via a computer or a tablet used by medical staff. This application will allow healthcare professionals to input patient data and receive real-time predictions on the likelihood of heart disease. The software will be designed with a simple interface to ensure ease of use, even for those with limited technical expertise.

The application will also include a reporting feature that provides detailed insights into the predictions, helping doctors to understand the factors contributing to the risk assessment. This transparency is crucial for making informed medical decisions.

Additionally, the system will be integrated with the hospital's existing electronic health record systems. This integration ensures that the patient data used for predictions is up-to-date and accurate, further enhancing the reliability of the model predictions.

In summary, by implementing these heart disease prediction models in a professional environment, we aim to provide healthcare professionals with a powerful tool to aid in early diagnosis and treatment planning, ultimately improving patient outcomes.

V. General Conclusion

This project demonstrated the effectiveness of various machine learning techniques in extracting actionable insights from data. The detailed comparison between supervised models, ensembles, and deep learning provided a clear understanding of their respective strengths and weaknesses. Future work may include applying more advanced data preprocessing techniques, exploring new deep learning architectures and improving the already explored.