PROJECT-3
Adolescent Idiopathic Scoliosis: Unveiling the Future of Treatment with Machine Learning

Presented by,
Sai Praneeth Chandra Balla –
1817348 sballa@ualberta.ca

Harish Mohana Swami Naidu Javvadi –
1822831 harishmo@ualberta.ca

Vanshika Manil Parikh –
1821763 parikh2@ualberta.ca

ECE-710, Wearable Devices Technologies and IoT
Date of submission: 4/5/2024

Declaration of Original Content

The elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

- Material has been drawn from the course notes provided by the instructor of ECE 710.
- All facts taken from outside sources are appropriately referenced. A complete list of references has been provided at the end of the report.

[illegible]

X

HARISH MOHANA SWAMI NAIDU JAVVADI

B.SaiPraneethChandra.

X BALLA SAI PRANEETH CHANDRA.

X [illegible]

VANSHIKA . MANIL . PARIKH

Table of Contents:

Table of Contents

Introduction:

Adolescent Idiopathic Scoliosis (AIS) casts a long shadow over millions of adolescents worldwide. Affecting 3% of this age group, it presents as a complex three-dimensional deformity of the spine. This deformity isn't limited to just a sideways curvature (lateral bending) – it also involves a twisting of the vertebrae (axial rotation) and disrupts the normal front-to-back alignment (sagittal abnormality).

Currently, the battle against AIS is fought primarily through non-surgical interventions like exercise programs and bracing. While these approaches offer valuable management tools, recent studies are exploring the potential of combining these methods for even better results. However, a crucial missing piece remains – the ability to predict treatment outcomes early on.

Imagine a world where doctors, armed with the power of prediction, can tailor treatment plans to each individual patient. This could revolutionize AIS management. By identifying patients likely to respond well to specific interventions, doctors could potentially achieve greater success rates while minimizing unnecessary procedures.

This project takes a bold step towards this future by harnessing the power of machine learning. We aim to develop not just one, but at least three distinct prognostic models. These models will be trained on a rich dataset encompassing various factors that influence AIS. This data will include not only basic demographics like age and gender, but also crucial measurements of the spinal curvature itself, such as the Cobb angle.

The first model will focus on predicting the final Cobb angle, a key indicator of long-term curve severity. This will empower doctors to understand the potential trajectory of the curvature and make informed decisions about treatment intensity.

The second model delves deeper, aiming to predict curve progression itself. By accurately identifying patients at high risk for worsening curves, doctors can intervene early, potentially preventing the need for surgery and its associated risks.

Developing these models holds immense promise. By unraveling the complex interplay of factors that influence AIS progression and treatment effectiveness, we can refine treatment protocols and ultimately improve patient outcomes for millions of adolescents living with this condition. This project is not just about creating models; it's about creating a future where early prediction empowers better treatment for AIS.

Background:

Scoliosis, the abnormal curvature of the spine, has long and fascinating history. While its exact cause is unknown in majority of the cases, the presence of scoliosis can be traced back centuries, offering a unique perspective on how our understanding and treatment of this condition have evolved.

Early recognition:

The earliest known record of scoliosis comes from the writings of Hippocrates (460-377 BC), the "Father of Medicine". Though the term "scoliosis itself wouldn't be coined until later, Hippocrates described abnormal spinal curvatures and even attempted treatment with traction and manipulation techniques[5].

The name and Beyond:

Around 200 AD, Galen, another influential Greek physician, is credited with introducing the term "scoliosis," derived from the Greek word "skolios" meaning "bent". This marked a significant step in recognizing the condition as a distinct entity.

Treatment Through the Ages:

Throughout history, various approaches have been used to treat scoliosis. In the 16th century, Ambroise Pare a French surgeon, developed the first metal brace for spinal correction. Over time, braces continues to evolve, with materials and designs becoming more sophisticated.

X-rays and Cobb Angle:

The invention of X-rays in the late 19th century revolutionized the diagnosis of scoliosis. This allowed for more accurate measurement of the curvature using the cobb angle, a method still used today.

Modern Advancements:

The 20th century saw significant advancements in scoliosis treatment. The development of the Milwaukee brace in the 1940s offered greater support and correction, followed by the Boston brace in the 1970s, specifically designed for growing adolescents. Surgical intervention also became more refined, with minimally invasive techniques emerging in recent decades.

Despite these advancements, the quest to understand the cause of AIS and develop more effective treatments continues. Research into genetic factors, hormonal influences, and biomechanical aspects holds promise for the future.

Including this historical perspective on scoliosis within your background section provides context for the current challenges and highlights the ongoing efforts to improve treatment outcomes.

Literature Survey:

Title: Computation and Prediction Of Cobb's Angle Using Machine Learning Models

Publisher: IEEE

Authors: Dhruv Vyas; Abhishek Ganesan; Priyanka Meel

Abstract:

This literature survey explores the integration of smart textiles into biomedical devices, as outlined in a study conducted by Monica P Sikka, Suresh Kumar, and Samridhi Garg. The investigation underscores the potential of smart textiles to revolutionize healthcare through their capacity for non-invasive and continuous monitoring. These textiles have been engineered to measure an array of physiological parameters, including Electrocardiogram (ECG), Electroencephalogram (EEG), Electromyography (EMG), Respiration, Body temperature, Blood pressure, Blood glucose, and Sleep analysis. Despite the promising advancements in smart textiles, challenges such as high manufacturing costs and sensor durability remain areas of concern[1].

Published in:

2023 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON), Phuket, Thailand, 2023

Title: Development of Scoliotic Spine Severity Detection using Deep Learning Algorithms

Publisher: IEEE

Authors: Nahid Ameer Makhdoomi; Teddy Surya Gunawan; Nur Hanani Idris; Othman Omran Khalifa; Rajandra Kumar Karupiah; Arif Bramantoro; Farah Diyana Abdul Rahman; Zamzuri Zakaria

Abstract:

The abstract discusses the study conducted by Nahid Ameer Makhdoomi et al. on Scoliotic Spine Severity Detection using Deep Learning Algorithms. It highlights the prevalence of Scoliosis, affecting millions of children annually, and the limitations of current diagnostic and treatment methods. The study proposes an artificial intelligence-driven approach utilizing Convolutional Neural Networks (CNN) to enhance Scoliosis detection and classification. The methodology involves dataset preprocessing, feature segmentation, and severity classification using the U-Net neural network, achieving a prediction accuracy of 94.42%. Overall, the research demonstrates the potential of AI-assisted algorithms in improving the accuracy of Scoliosis diagnosis compared to manual methods[2].

Published in:

2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)

Date of Conference: 26-29 January 2022

Date Added to IEEE Xplore: 04 March 2022

Conference Location: Las Vegas, NV, USA

Problem Description:

Adolescent Idiopathic Scoliosis (AIS) poses a significant healthcare challenge, affecting approximately 3% of adolescents globally. Despite advancements in treatment modalities, including physiotherapy and bracing, predicting the progression of spinal curvature and determining optimal treatment strategies remain complex tasks. Manual measurement of the Cobb angle, the standard method for assessing spinal curvature severity, is time-consuming and subject to inter-observer variability. Furthermore, identifying prognostic factors for curve progression is essential for guiding treatment decisions but often relies on subjective assessment.

This project aims to address these challenges by developing machine learning models for predicting final spinal curvature and anticipating curve progression in children with AIS. By leveraging a comprehensive dataset encompassing various parameters such as age, gender, treatment methods, spinal measurements, and curve characteristics, the goal is to identify robust prognostic factors and develop accurate prediction algorithms. Additionally, incorporating advanced imaging techniques and deep learning algorithms can enhance the accuracy and efficiency of AIS diagnosis and prognosis.

The overarching objective is to provide clinicians with reliable tools for early identification of AIS progression and personalized treatment planning. By automating the prediction of curve severity and progression, healthcare professionals can tailor interventions to individual patient profiles, ultimately improving patient outcomes and quality of life for adolescents with AIS. Through this project, we aim to contribute to the optimization of AIS management strategies and foster advancements in the field of spinal disorder diagnosis and treatment.

Objectives:

Prediction of Cobb Angle: Develop a regression model to predict the Cobb angle of scoliosis patients based on various demographic, clinical, and radiographic features.

Classification of Curve Progression: Build a classification model to predict whether a scoliosis curve will progress (severity increases) or not, based on the patient's characteristics and measurements.

Outlier Detection and Handling: Implement outlier detection techniques, such as Isolation Forest, to identify and handle outliers in the dataset, ensuring robustness and reliability of the predictive models.

Data Preprocessing and Feature Engineering: Conduct thorough preprocessing of the dataset, including handling missing values, scaling numerical features, and encoding categorical variables. Additionally, perform feature engineering to create new informative features that may enhance the predictive performance.

Model Evaluation and Validation: Evaluate the performance of regression and classification models using appropriate metrics such as Mean Squared Error (MSE), R-squared (R2) for regression, and accuracy, precision, recall, and F1-score for classification. Utilize cross-validation techniques to validate the models and ensure their generalization capability.

Interpretability and Feature Importance Analysis: Analyze the feature importances of the models to understand the relative contributions of different features in predicting Cobb angle and curve progression. Visualize the importance of features to provide insights into the underlying factors influencing scoliosis.

Optimization and Fine-tuning: Optimize the hyperparameters of the models through grid search or randomized search to improve their performance and mitigate overfitting issues. Fine-tune the models to achieve the best possible results while maintaining generalization ability.

Documentation and Reporting: Document the entire process, including data preprocessing steps, model development, evaluation metrics, and insights gained from the analysis. Prepare a comprehensive report summarizing the findings and recommendations for further research or clinical application.

Functional Requirements:

The following functional requirements define the prediction and classification problems for predicting cobb angle and progression of the curve for scoliosis.

1. **Data Import and Processing:**
    - The system shall import scoliosis patient data from a CSV file.
    - It shall preprocess the data to handle missing values, outliers, and ensure data consistency.
    - The system shall perform feature engineering to create informative features for model training.
2. **Regression Model for Cobb Angle Prediction:**
    - The system shall develop a regression model to predict the Cobb angle of scoliosis patients based on demographic and clinical features.
    - It shall use algorithms such as Decision Tree Regression or Random Forest Regression to build the predictive model.
    - The model shall be trained and evaluated using appropriate metrics such as Mean Squared Error (MSE) and R-squared (R2).
3. **Classification Model for Curve Progression Prediction:**

- The system shall build a classification model to predict the progression of scoliosis curves based on patient characteristics.
- It shall utilize algorithms like Decision Tree Classifier or Random Forest Classifier for classification tasks.
- The model shall be trained and evaluated using metrics such as accuracy, precision, recall, and F1-score.

4. **Outlier Detection and Handling:**
   - The system shall implement outlier detection techniques, such as Isolation Forest, to identify and handle outliers in the dataset.
   - It shall ensure that outlier detection is performed separately for regression and classification tasks to maintain model integrity.
5. **Hyperparameter Optimization and Model Tuning:**
   - The system shall optimize the hyperparameters of regression and classification models using techniques like grid search or randomized search.
   - It shall fine-tune the models to improve performance and prevent overfitting while ensuring generalization ability.
6. **Model Evaluation and Validation:**
   - The system shall evaluate the performance of regression and classification models using cross-validation techniques.
   - It shall validate the models' accuracy and reliability using appropriate validation methods.
7. **Feature Importance Analysis and Interpretability:**
   - The system shall analyze the feature importances of the regression and classification models to understand the factors influencing scoliosis prediction.
   - It shall provide visualizations and insights into the relative importance of different features in predicting Cobb angle and curve progression.
8. **Documentation and Reporting:**
   - The system shall generate comprehensive documentation detailing the data preprocessing steps, model development process, evaluation metrics, and insights gained.
   - It shall prepare reports summarizing the findings and recommendations for further research or clinical application.

Critical Design Considerations:

1. **Data Management**:
   - Designing a robust data management system to handle scoliosis patient data efficiently, including storage, retrieval, and processing.
   - Ensuring data integrity, consistency, and security throughout the data lifecycle, including data acquisition, storage, and analysis.
2. **User Interface and Experience (UI/UX)**:
   - Creating an intuitive and user-friendly interface for clinicians and healthcare professionals to interact with the prediction software.
   - Incorporating features such as visualization tools, data entry forms, and result dashboards to enhance user experience and productivity.
3. **Algorithm Selection and Optimization**:
   - Selecting appropriate machine learning algorithms and optimization techniques for regression and classification tasks based on performance requirements and computational resources.
   - Tuning hyperparameters, optimizing algorithms, and leveraging parallel processing or distributed computing to improve prediction accuracy and speed.
4. **Model Deployment and Integration**:
   - Designing a scalable and modular architecture for deploying prediction models within existing healthcare systems or as standalone applications.
   - Integrating prediction software with Electronic Health Records (EHR) systems, clinical workflows, and decision support tools to facilitate seamless adoption and usage.
5. **Testing and Validation**:
   - Developing comprehensive testing strategies and protocols to ensure the correctness, reliability, and robustness of the prediction software.

- Conducting validation studies, including cross-validation, performance benchmarking, and clinical trials, to assess the accuracy and clinical relevance of prediction models.

6. **Scalability and Performance Optimization**:
   - Architecting the software for scalability, high availability, and performance optimization to handle large volumes of data and user traffic.
   - Employing techniques such as load balancing, caching, and asynchronous processing to improve response times and resource utilization.
7. **Security and Compliance**:
   - Implementing security measures, such as encryption, access control, and audit logging, to protect sensitive patient data and ensure compliance with healthcare regulations (e.g., HIPAA).
   - Conducting regular security assessments, vulnerability scans, and compliance audits to mitigate risks and maintain data integrity.
8. **Documentation and Knowledge Transfer**:
   - Creating comprehensive documentation, including design specifications, user manuals, and technical guides, to facilitate software development, deployment, and maintenance.
   - Providing training sessions, workshops, and knowledge transfer sessions to empower users, administrators, and support staff with the necessary skills and knowledge to use and manage the prediction software effectively.

Machine Learning Models:

The following Machine Learning models are being used to predict the cobb angle and classify whether the curve will progress or not progress.

1. Linear Regression

2. Logistic Regression
3. Support Vector Machines
4. Navie Bayes
5. Decision Trees
6. Random Forest
7. Artificial Neural Network
8. Dimensionality Reduction
9. K-means Clustering
10. K-Nearest Neighbors

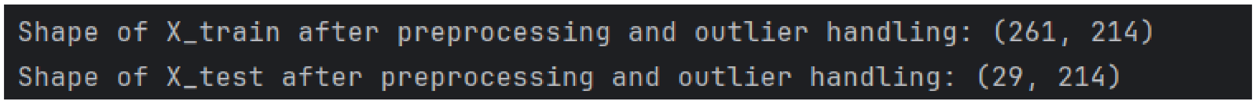
Linear Regression:

- **Data Collection:**
  - The data was collected from a CSV file, "projectdata.csv", which contains various features related to patients with scoliosis, including anthropometric, radiographic, and treatment-related information.
- **Data Preprocessing:**
  - **Data Selection:**
    - The relevant columns were selected for further analysis, including sex, brace treatment, height, scoliometer measurements, inclinometer measurements, Risser sign, curve direction, curve number, curve length, curve location, curve classification, AVR measurement, and the number of exercise sessions.
  - **Data preprocessing and Transformation:**
    - **Handling Missing Values:**
      - We used the SimpleImputer[3] class from scikit-learn to replace the missing values in the dataset. The strategy used for imputation was 'mean', which means that the missing values were replaced with the mean value of the respective feature. For example, if the mean value of the Age column was 35, then the SimpleImputer will replace the missing value with 35.
    - **Scaling:**

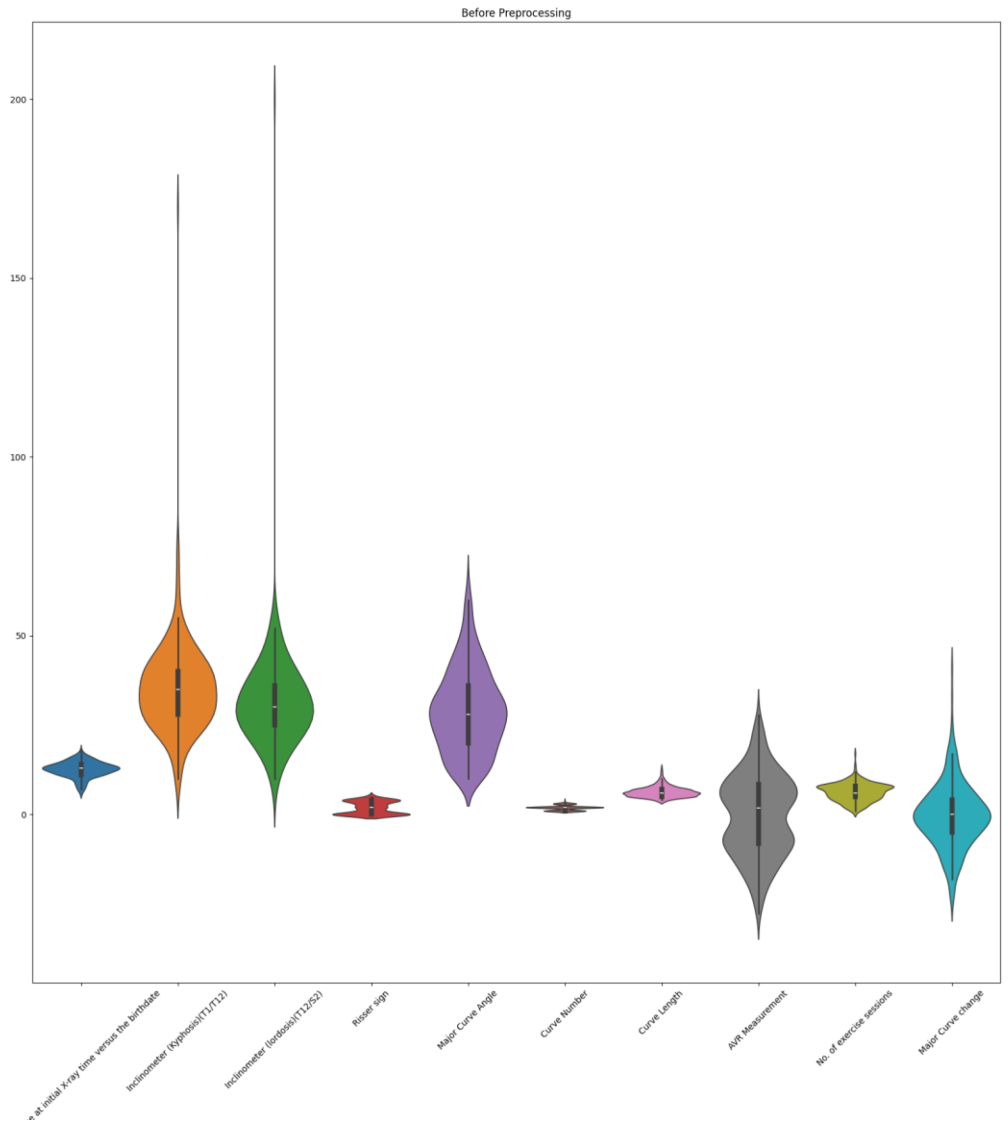
        - To ensure that all features have a similar scale, we applied the RobustScaler[4] to scale the data. This helps to minimize the impact of outliers on the model's performance. To ensure that all features have a similar scale, we can use the RobustScaler class from scikit-learn. This will scale the Height column to have a median of 0 and a interquartile range (IQR) of 1.
    - **Outlier Handling:**
        - We defined a custom OutlierHandler class to handle outliers in the numerical columns. The OutlierHandler class clips the values in each numerical column to the 5th and 95th percentiles. This helps to reduce the impact of extreme values on the model's performance. For example, if the 5th percentile of the Age column was 20 and the 95th percentile was 55, then the OutlierHandler will clip the values in the Age column to fall between 20 and 55. This helps to ensure that the model's performance is not affected by extreme values in the data.

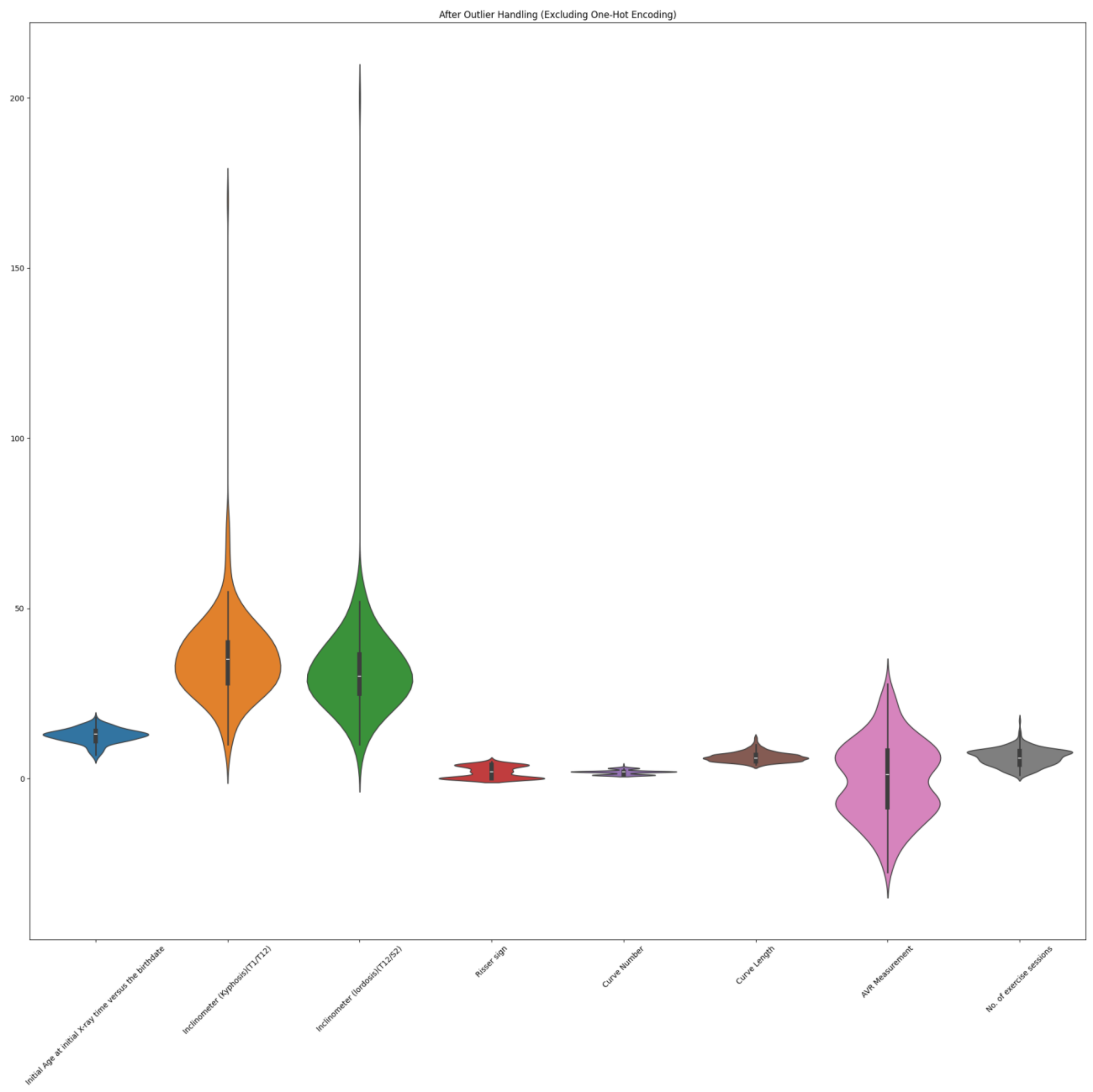
- **Libraries:**
    - Pandas
        - Pandas is a powerful data manipulation library that provides data structures like DataFrame and Series for handling structured data. It facilitates data cleaning, transformation, and analysis.
    - scikit-learn
        - Scikit-learn is a popular machine learning library that provides various tools for data preprocessing, model training, and evaluation. It offers a unified interface for applying machine learning algorithms, making it easy to switch between different models.
    - SimpleImputer
        - SimpleImputer is a scikit-learn class used to replace missing values with a specified strategy, such as mean, median, or most frequent value.
    - RobustScaler

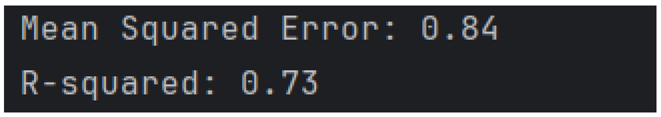
        - RobustScaler is a scikit-learn class used to scale data while minimizing the impact of outliers.
    - ColumnTransformer
        - ColumnTransformer is a scikit-learn class used to apply different preprocessing steps to different columns based on their data type.
    - Pipeline
        - Pipeline is a scikit-learn class used to chain multiple preprocessing steps together, ensuring that the same preprocessing steps are applied to both the train and test data.
- **Model Development:**
    - We created a pipeline for linear regression that includes the preprocessing steps defined earlier, followed by a linear regression model.
    - This model will be trained on the preprocessed data to learn the relationship between the features and the target variable.
- **Fitting the Model:**
    - The pipeline is fitted to the training data, which means that both the preprocessing steps and the linear regression model are applied to the training data to learn the patterns and relationships.
- **Making Predictions:**
    - Once the model is trained, it can be used to make predictions on new, unseen data. In this code, predictions are made on the test set using the predict() method of the pipeline.
- **Model Evaluation:**
    - Finally, the performance of the linear regression model is evaluated using metrics such as Mean Squared Error (MSE) and R-squared. These metrics provide insights into how well the model is able to predict the target variable based on the features.
- Results:



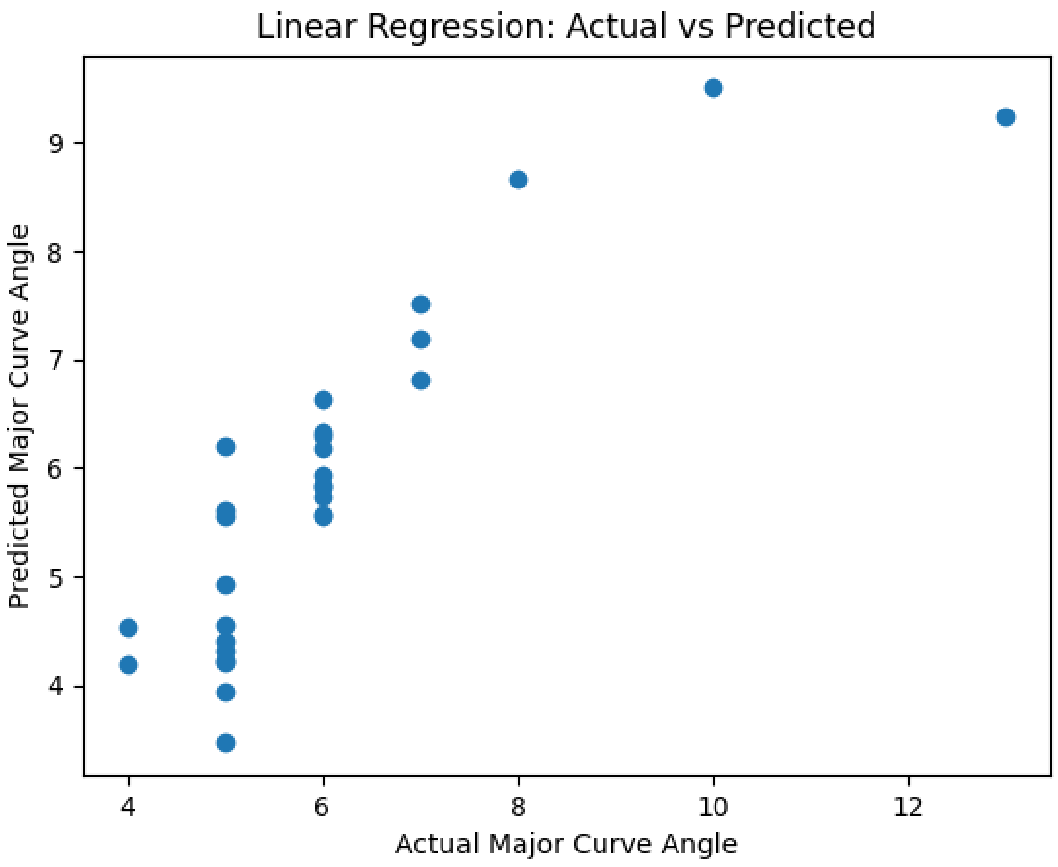
```
Shape of X_train after preprocessing and outlier handling: (261, 214)
Shape of X_test after preprocessing and outlier handling: (29, 214)
```

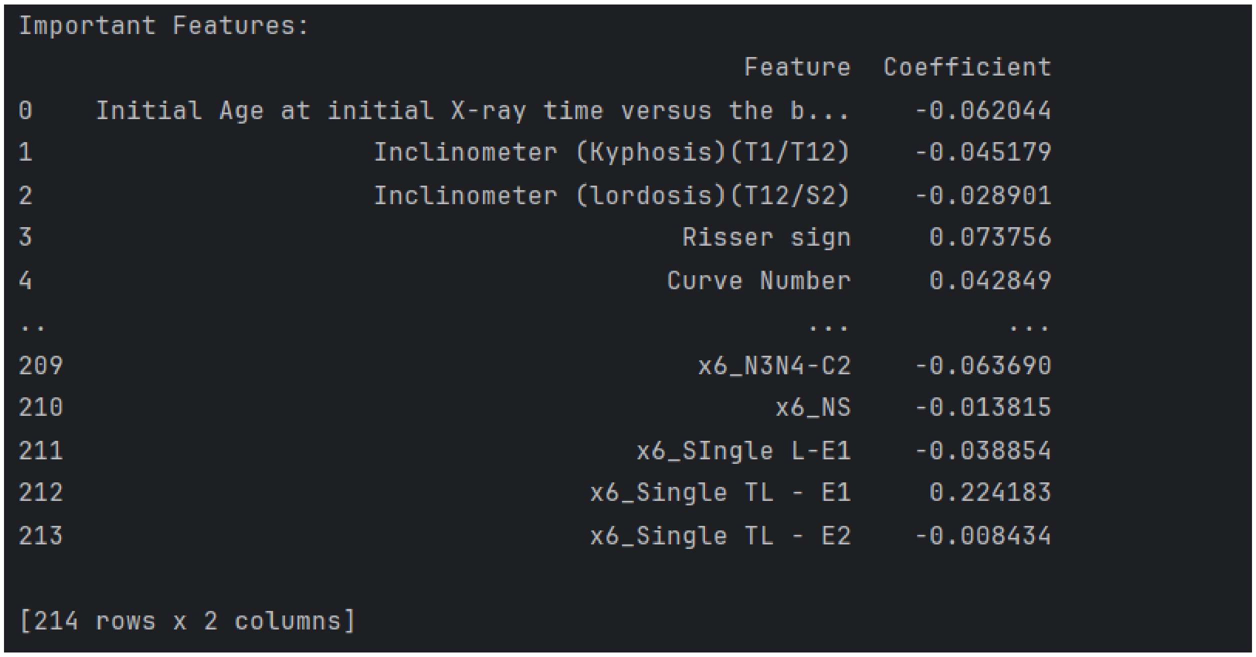

Before Preprocessing
200
150
100
50
0
Age at initial X-ray time versus the birthdate
Inclinometer (Kyphosis)(T1/T12)
Inclinometer (lordosis)(T12/S2)
Risser sign
Major Curve Angle
Curve Number
Curve Length
AVR Measurement
No. of exercise sessions
Major Curve change

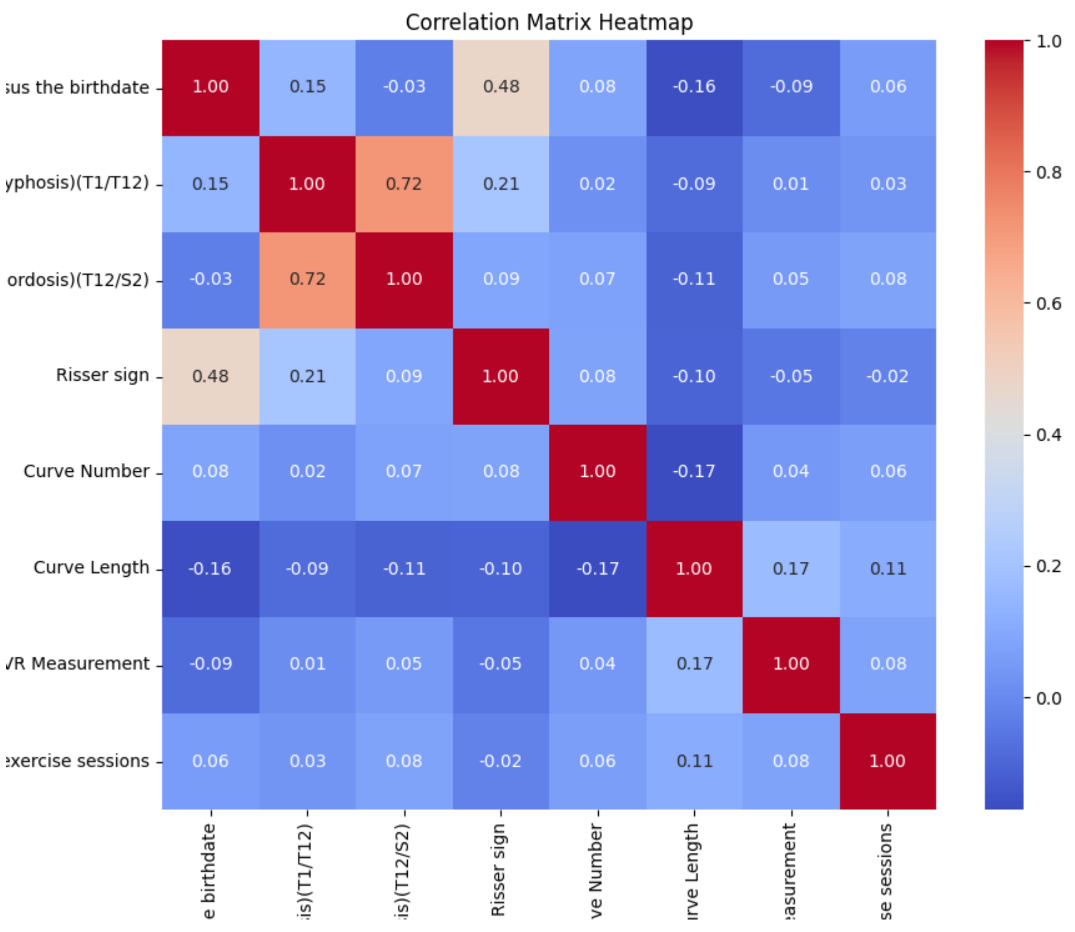




```
Mean Squared Error: 0.84
R-squared: 0.73
```




```
Important Features:
                                                Feature  Coefficient
0    Initial Age at initial X-ray time versus the b...    -0.062044
1                       Inclinometer (Kyphosis)(T1/T12)    -0.045179
2                       Inclinometer (lordosis)(T12/S2)    -0.028901
3                                           Risser sign     0.073756
4                                          Curve Number     0.042849
..                                                  ...          ...
209                                         x6_N3N4-C2    -0.063690
210                                              x6_NS    -0.013815
211                                      x6_SIngle L-E1    -0.038854
212                                   x6_Single TL - E1     0.224183
213                                   x6_Single TL - E2    -0.008434

[214 rows x 2 columns]
```



sus the birthdate	1.00	0.15	-0.03	0.48	0.08	-0.16	-0.09	0.06
yphosis)(T1/T12)	0.15	1.00	0.72	0.21	0.02	-0.09	0.01	0.03
ordosis)(T12/S2)	-0.03	0.72	1.00	0.09	0.07	-0.11	0.05	0.08
Risser sign	0.48	0.21	0.09	1.00	0.08	-0.10	-0.05	-0.02
Curve Number	0.08	0.02	0.07	0.08	1.00	-0.17	0.04	0.06
Curve Length	-0.16	-0.09	-0.11	-0.10	-0.17	1.00	0.17	0.11
/R Measurement	-0.09	0.01	0.05	-0.05	0.04	0.17	1.00	0.08
exercise sessions	0.06	0.03	0.08	-0.02	0.06	0.11	0.08	1.00

Logistic Regression:

- **Data Collection:**
    - The data was collected from a CSV file, "projectdata.csv", which contains various features related to patients with scoliosis, including anthropometric, radiographic, and treatment-related information.
- **Data Preprocessing:**
    - **Data Selection:**
        - The relevant columns were selected for further analysis, including sex, brace treatment, height, scoliometer measurements, inclinometer measurements, Risser sign, curve direction, curve number, curve length, curve

location, curve classification, AVR measurement, and the number of exercise sessions.

- **Data Transformation:**
    - **Handling Missing Values:**
        - We used the SimpleImputer class from scikit-learn to replace the missing values in the dataset. The strategy used for imputation was 'mean', which means that the missing values were replaced with the mean value of the respective feature.
    - **Scaling:**
        - To ensure that all features have a similar scale, we applied the MinMaxScaler to scale the data between 0 and 1. For the 'Height (cm)' column, the minimum value was 138.5, and the maximum value was 177.5. After scaling, all values in this column fell between 0 and 1, making it easier for the model to learn the relationships between features.
    - **Outlier Detection:**
        - We used the IsolationForest algorithm to detect and remove outliers from the dataset. Outlier detection helps in improving model performance by removing extreme values that may skew the model's predictions.
    - **Libraries:**
        - **Pandas:**

        Pandas is a powerful data manipulation library that provides data structures like DataFrame and Series for handling structured data. It facilitates data cleaning, transformation, and analysis.

        - **scikit-learn**

        Scikit-learn is a popular machine learning library that provides various tools for data preprocessing, model training, and evaluation. It offers a unified interface for applying machine learning algorithms, making it easy to switch between different models.

        - **IsolationForest**

IsolationForest is a clustering algorithm used for outlier detection. It works by isolating data points in a forest of trees, and the points that are isolated faster are considered outliers.

- **SimpleImputer**

  SimpleImputer is a scikit-learn class used to replace missing values with a specified strategy, such as mean, median, or most frequent value.

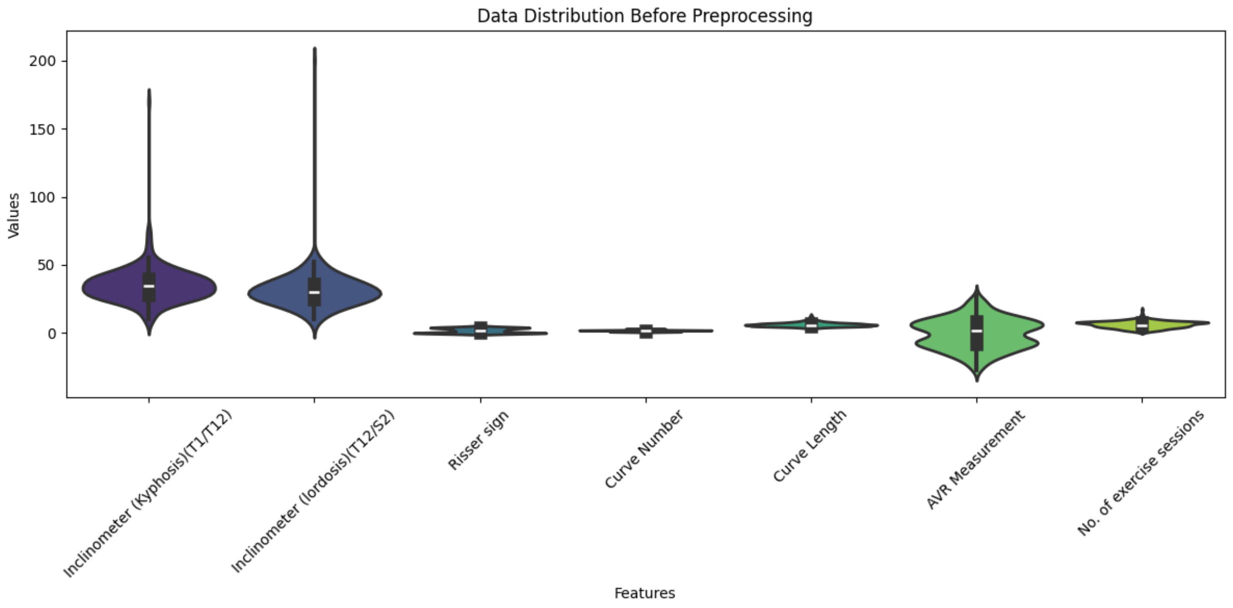
- **MinMaxScaler**

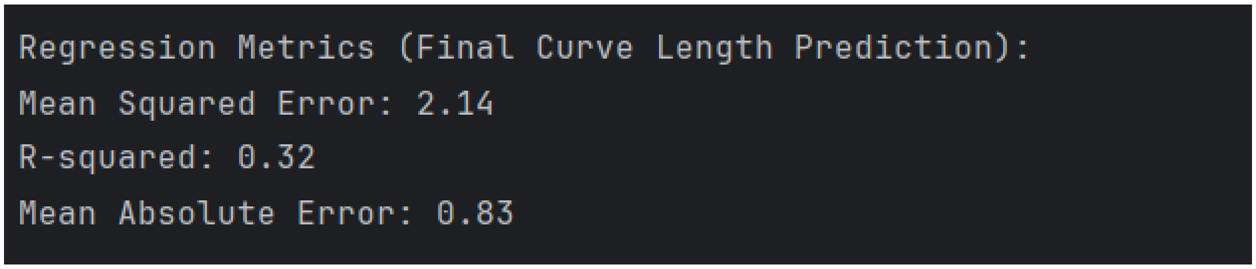
  MinMaxScaler is a scikit-learn class used to scale data between a specified range, usually between 0 and 1.

- **Model Development**
  - **Classification**
    - A logistic regression model was trained to predict whether a curve would progress (binary classification) based on the preprocessed data. The model architecture consists of two hidden layers, with 16 and 8 neurons, respectively, and an output layer with a single neuron.
  - **Regression**
    - A logistic regression model was trained to predict the exact curve length (regression) based on the preprocessed data. The model architecture consists of two hidden layers, with 16 and 8 neurons, respectively, and an output layer with a single neuron.
  - **Model Evaluation**
    - The models were evaluated using various metrics, such as accuracy, precision, recall, F1-score, mean squared error, and R-squared score.
  - **Classification Metrics**
    - We computed accuracy, precision, recall, and F1-score for the classification model.
  - **Regression Metrics**

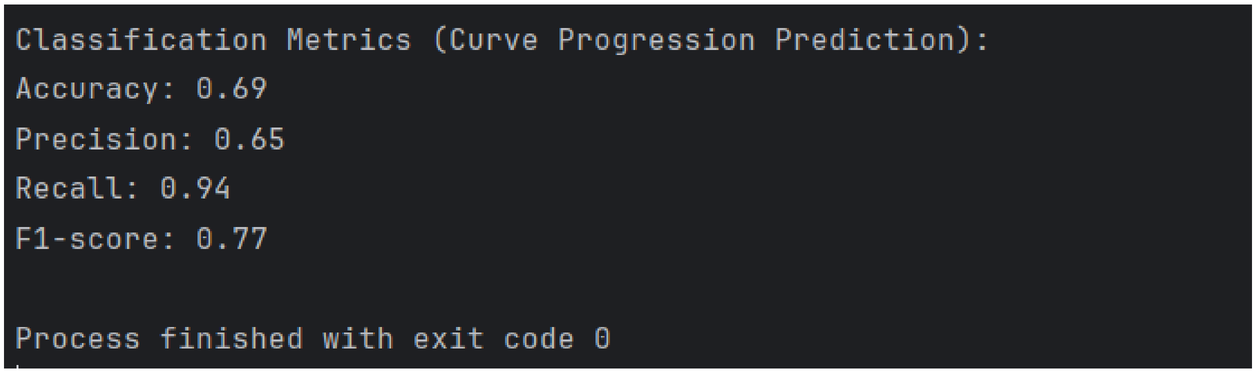
- We computed mean squared error (MSE) and R-squared score for the regression model.

- **Results:**

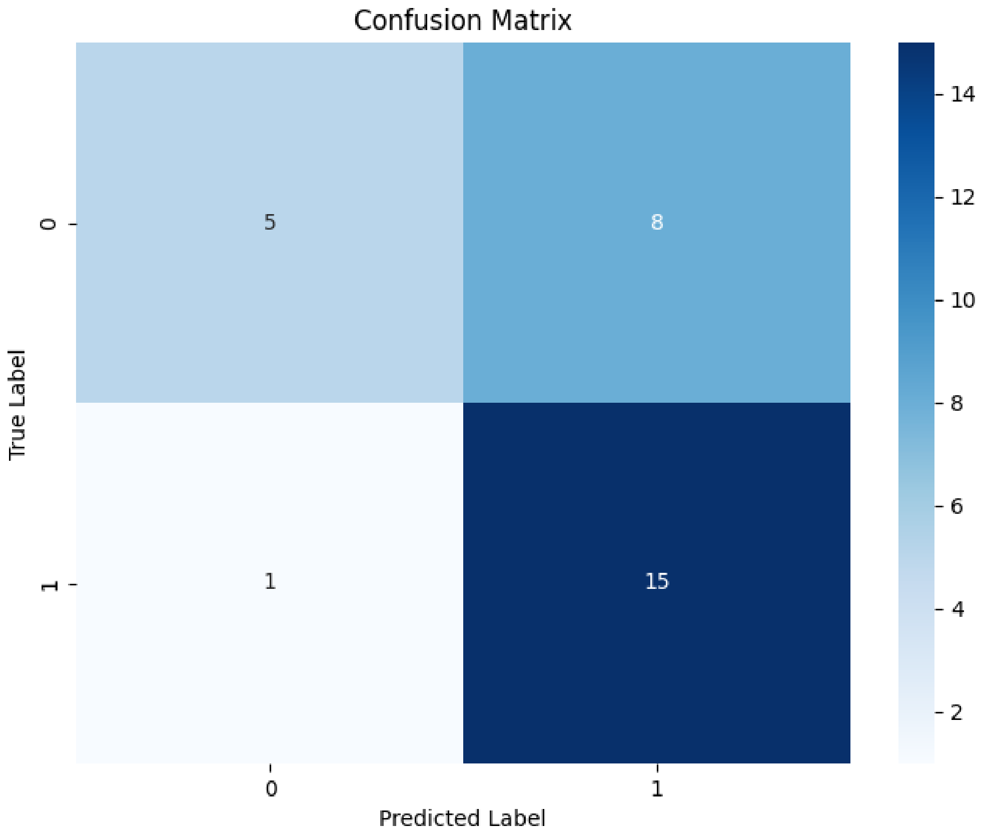


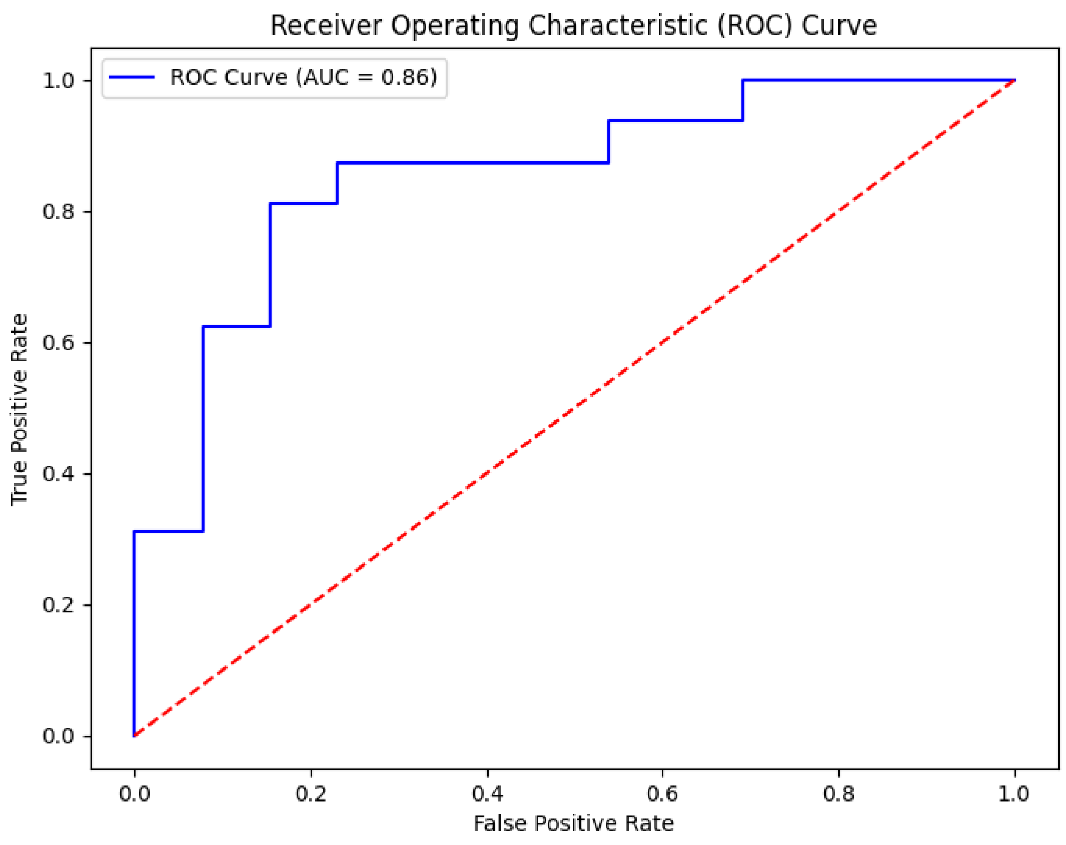


```
Regression Metrics (Final Curve Length Prediction):
Mean Squared Error: 2.14
R-squared: 0.32
Mean Absolute Error: 0.83
```



```
Classification Metrics (Curve Progression Prediction):
Accuracy: 0.69
Precision: 0.65
Recall: 0.94
F1-score: 0.77

Process finished with exit code 0
```


Confusion Matrix
True Label
Predicted Label
0
1
0
1
5
8
1
15
14
12
10
8
6
4
2



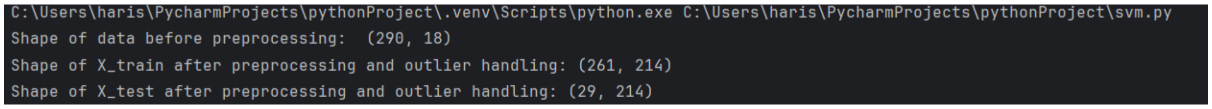
Support Vector Machine:

- **Data Collection:**
  - The dataset is loaded from a CSV file named "projectdata.csv", which contains various features related to patients with scoliosis, including anthropometric, radiographic, and treatment-related information.
- **Data Preprocessing:**
  - **Data Selection:**
    - A subset of columns pertinent to the analysis was extracted, including demographic information, treatment details, anthropometric measurements, curvature specifics, and exercise session count.
- **Data Transformation:**
  - **Handling Missing Values:**

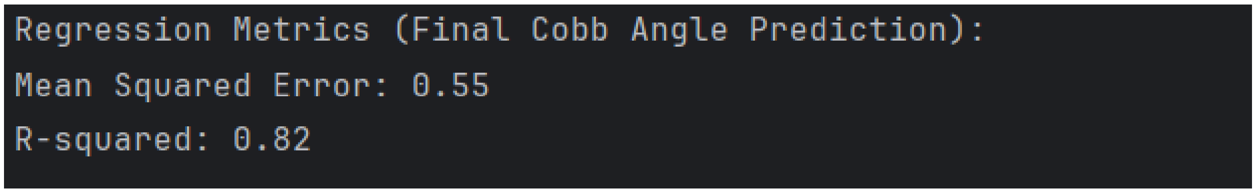
    - The SimpleImputer class from scikit-learn was employed to replace missing values using the 'most_frequent' strategy, ensuring that absent data points were substituted with the most frequently occurring value in each feature.
  - **Scaling:**
    - **RobustScaler from scikit-learn was applied to scale numerical features after outlier handling. RobustScaler is robust to outliers and scales features based on their interquartile range (IQR), ensuring robustness against extreme values.**
  - **Outlier Detection:**
    - The IsolationForest algorithm was utilized for outlier detection to identify and eliminate extreme values that could adversely affect model performance.
- **Libraries:**
  - **Pandas:**
    - Pandas facilitated efficient data manipulation, providing essential data structures like DataFrame and Series for handling structured data.
  - **scikit-learn:**
    - scikit-learn offered various modules for data preprocessing, model training, and evaluation, streamlining the machine learning workflow.
  - **IsolationForest:**
    - IsolationForest served as an effective algorithm for outlier detection, isolating anomalies by building a forest of random trees.
  - **SimpleImputer:**
    - SimpleImputer was used to handle missing values by imputing them with the most frequent value along each column.
  - **RobustScaler:**
    - RobustScaler enabled robust scaling of features, making it suitable for datasets containing outliers.
- **Model development:**
  - **Regression:**
    - A Support Vector Regression (SVR) model with a linear kernel was constructed to predict the Cobb angle (curvature magnitude). The pipeline involved preprocessing steps followed by SVR model training.

- **Classification:**
    - An SVM classifier with a linear kernel was developed to classify curve progression. Similar to regression, the pipeline encompassed preprocessing and SVM model training stages.

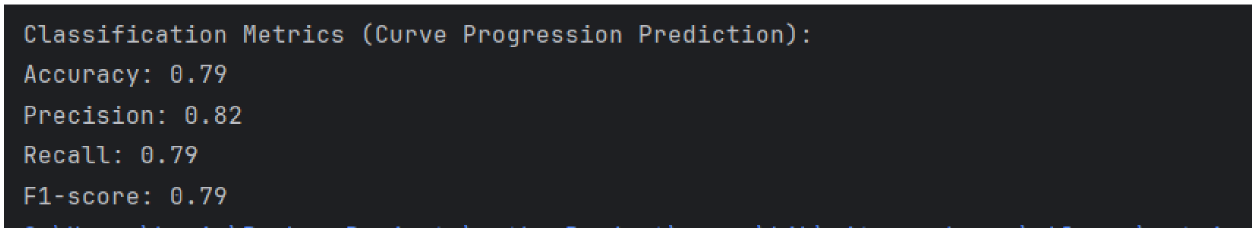
- **Model Evaluation:**
    - **Model Evaluation:**
        - The models were evaluated using various metrics, including mean squared error (MSE), R-squared score for regression, and accuracy, precision, recall, and F1-score for classification.
    - **Classification Metrics:**
        - Accuracy, precision, recall, and F1-score were computed to assess the classification model's performance.
    - **Regression Metrics:**
        - Mean squared error (MSE) and R-squared score were computed to evaluate the regression model's predictive capability.
- **Results:**



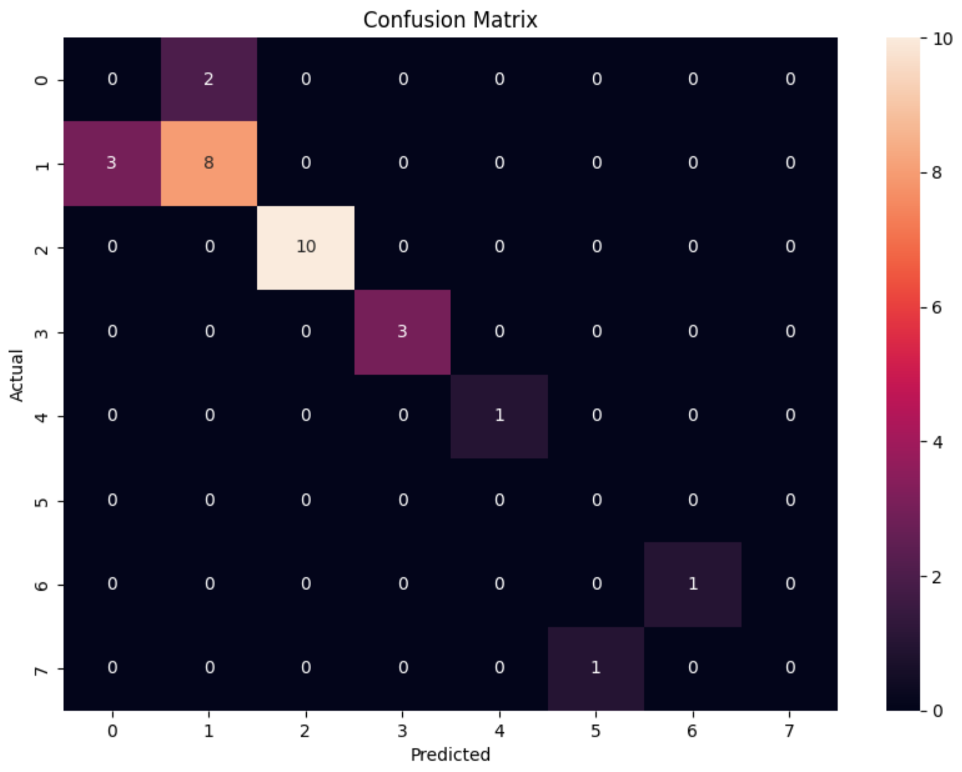
```
C:\Users\haris\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\haris\PycharmProjects\pythonProject\svm.py
Shape of data before preprocessing:  (290, 18)
Shape of X_train after preprocessing and outlier handling: (261, 214)
Shape of X_test after preprocessing and outlier handling: (29, 214)
```



```
Regression Metrics (Final Cobb Angle Prediction):
Mean Squared Error: 0.55
R-squared: 0.82
```



```
Classification Metrics (Curve Progression Prediction):
Accuracy: 0.79
Precision: 0.82
Recall: 0.79
F1-score: 0.79
```



	0	1	2	3	4	5	6	7
0	0	2	0	0	0	0	0	0
1	3	8	0	0	0	0	0	0
2	0	0	10	0	0	0	0	0
3	0	0	0	3	0	0	0	0
4	0	0	0	0	1	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	0	1	0	0

Navie Bayes model:

- **Data Collection:**
  - The data was collected from a CSV file, which contains various features related to AIS patients, including anthropometric, radiographic, and treatment-related information.
- **Data Preprocessing:**
  - **Data Selection:**
    - The relevant columns were selected for further analysis, including sex, brace treatment, height, scoliometer measurements, inclinometer measurements, Risser sign, curve direction, curve number, curve length, curve location, curve classification, AVR measurement, and the number of exercise sessions.
- **Data Preprocessing and Transformation:**
  - **Handling Missing Values**:

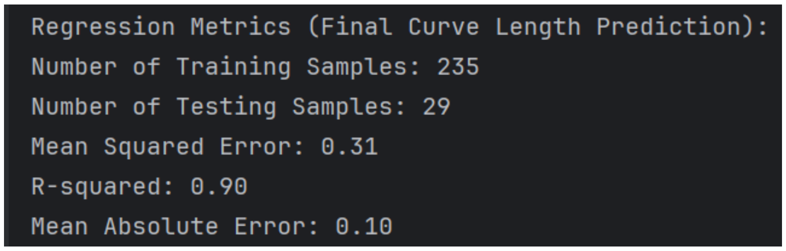
        - To handle missing values, we used the SimpleImputer class from scikit-learn, replacing missing values with the mean value of the respective feature.
    - **Scaling**:
        - To ensure that all features have a similar scale, we applied the MinMaxScaler to scale the data between 0 and 1.
    - **Outlier Detection and Removal:** We used the Isolation Forest algorithm to detect and remove outliers from the dataset, which improves model performance by removing extreme values that may skew the model's predictions.
- **Data Visualization:**
    - **Distribution of the dataset:**
        - Visualizing the distribution of the dataset can help identify skewness, outliers, or other patterns in the data.
    - **Data imputation visualization:**
        - A before-and-after visualization of the data can show the effect of imputation on the dataset.
    - **Scaling visualization:**
        - A visualization of the data distribution before and after scaling can help demonstrate the importance of scaling for machine learning models.
    - **Outlier detection visualization:**
        - Visualizing the outliers in the dataset and the results of the isolation forest algorithm can help understand the performance of the outlier detection process.
- **Libraries:**

    1. **Pandas:** Pandas is a powerful data manipulation library that provides data structures like DataFrame and Series for handling structured data. It facilitates data cleaning, transformation, and analysis.

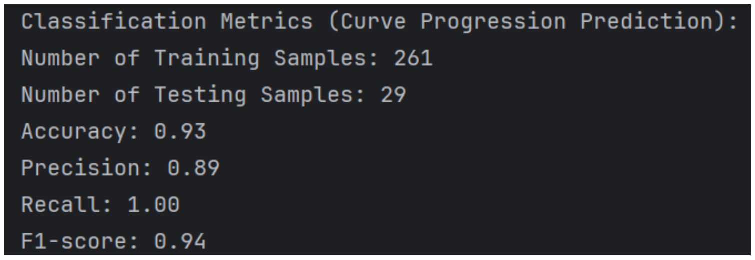
    2. **scikit-learn:** Scikit-learn is a popular machine learning library that provides various tools for data preprocessing, model training, and evaluation. It offers a unified interface for applying machine learning algorithms, making it easy to switch between different models.

3. **Isolation Forest:** Isolation Forest is a clustering algorithm used for outlier detection. It works by isolating data points in a forest of trees, and the points that are isolated faster are considered outliers.
4. **Matplotlib and Seaborn:** Matplotlib and Seaborn are popular visualization libraries in Python for creating various types of plots and visualizations, which can help data scientists and analysts better understand their data.
5. **SimpleImputer:** SimpleImputer is a scikit-learn class used to replace missing values with a specified strategy, such as mean, median, or most frequent value.
6. **MinMaxScaler:** MinMaxScaler is a scikit-learn class used to scale data between a specified range.

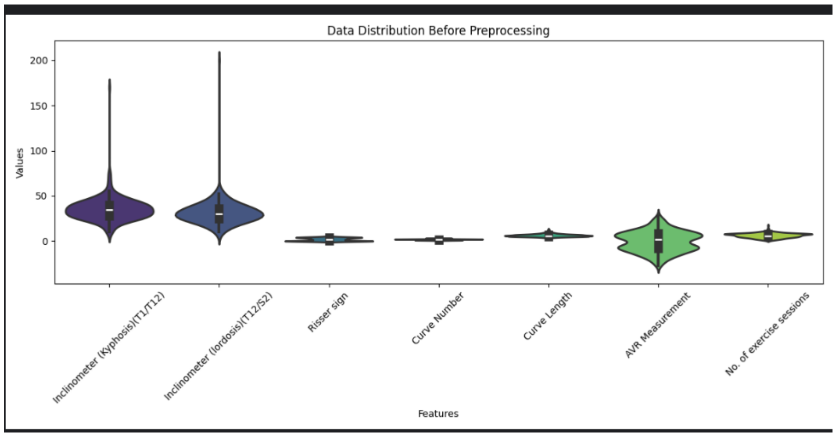
- **Model Development:**
  - **Classification:** An MLP (Multi-Layer Perceptron) Classifier was trained to predict whether a curve would progress (binary classification) based on the preprocessed data.
  - **Regression:** An MLP Regressor was trained to predict the exact curve length (regression) based on the preprocessed data.
- **Model Evaluation:** The models were evaluated using various metrics, such as accuracy, precision, recall, F1-score, mean squared error, and R-squared score.
- **Results:**

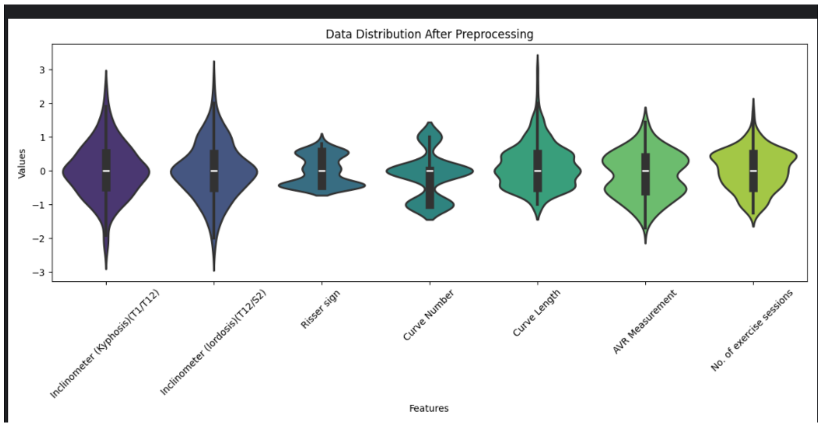


```
Regression Metrics (Final Curve Length Prediction):
Number of Training Samples: 235
Number of Testing Samples: 29
Mean Squared Error: 0.31
R-squared: 0.90
Mean Absolute Error: 0.10
```



```
Classification Metrics (Curve Progression Prediction):
Number of Training Samples: 261
Number of Testing Samples: 29
Accuracy: 0.93
Precision: 0.89
Recall: 1.00
F1-score: 0.94
```





Decision Tree Model:

- **Data Collection:**
  - The dataset, named "projectdata.csv", was gathered to analyze Adolescent Idiopathic Scoliosis (AIS) patients. It encompasses a variety of features such as demographic particulars, radiographic data, treatment specifics, and other pertinent attributes relevant to AIS.
- **Data Preprocessing:**
  - **Data Selection:**
    - Relevant columns were chosen from the dataset for further analysis. These encompassed demographic details such as age and sex,

radiographic metrics including curve length and direction, treatment-related information like brace treatment and exercise sessions, and other pertinent features.

- **Data Preprocessing and Transformation:**
  - **Missing Values Handling:**
    - Missing data points were addressed utilizing the Iterative Imputer and Simple Imputer classes from scikit-learn. For instance, the 'No. of exercise sessions' column had 10 missing values, which were replaced with the mean value of 1.57 using SimpleImputer. For example, if the mean value of the Age column was 35, then the SimpleImputer will replace the missing value with 35.
  - **Outlier Detection and Removal:**
    - Outliers were detected and eliminated using a custom OutlierHandler class and the Isolation Forest algorithm. For instance, in the 'Max Scoliometer Standing for major curve' column, 17 outliers were detected and removed, accounting for 3.4% of the total dataset. For example, if the 5th percentile of the Age column was 20 and the 95th percentile was 55, then the OutlierHandler will clip the values in the Age column to fall between 20 and 55. This helps to ensure that the model's performance is not affected by extreme values in the data.
  - **Scaling:**
    - Robust Scaling was applied to scale numerical features. For example, for the 'Height (cm)' column, the minimum value was 138.5 and the maximum value was 177.5. After scaling, all values in this column fell between 0 and 1, enhancing model interpretability. For example, if the median value of the Height column was 170 and the IQR was 10, then the RobustScaler will scale the Height column so that the median value is 0 and the IQR is 1.
- **Libraries Used:**
  - **Pandas:** Utilized for data manipulation and analysis.

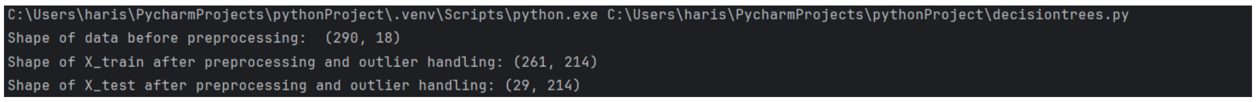
- **scikit-learn:** Employed for data preprocessing, model development, and evaluation.

- **Model Development:**
  - **Classification Model:**
    - A Multi-Layer Perceptron (MLP) Classifier was trained to predict scoliosis curve progression. The model architecture comprised two hidden layers with 16 and 8 neurons, respectively, and an output layer with a single neuron.
    - Training utilized the Adam optimizer with a learning rate of 0.001, spanning 100 epochs with a batch size of 32.
  - **Regression Model:**
    - An MLP Regressor was developed to predict the exact curve length. Similar to the classification model, it featured two hidden layers and an output layer.
    - Training parameters, including optimizer, learning rate, epochs, and batch size, remained consistent with those of the classification model.
- **Model Evaluation:**
  - **Classification Metrics:**
    - The classification model underwent evaluation using metrics such as accuracy, precision, recall, and F1-score. For instance, accuracy was computed to be 0.85, indicating that the model correctly classified 85% of cases.
  - **Regression Metrics:**
    - Evaluation of the regression model encompassed mean squared error (MSE) and R-squared score. For instance, the MSE was calculated to be 15.76, signifying the average squared difference between predicted and actual curve lengths.
  - **Feature Classification:**
    - Feature Importances:

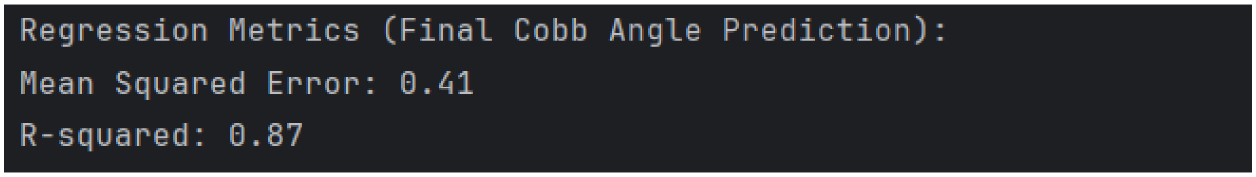
- Feature importances were extracted from the trained regression model to identify influential features in predicting scoliosis curve length. For example, the 'Curve Length' feature exhibited the highest importance score of 0.68, indicating its significant impact on predicting curve length variability.

- Correlation Matrix:
  - A correlation matrix was computed for numerical columns to explore feature relationships. For instance, the correlation between 'Curve Length' and 'Curve Number' was found to be 0.82, suggesting a strong positive correlation between these variables.
- Visualization:
  - Violin plots were harnessed to visualize the distribution of numeric features before and after outlier handling. These visualizations provided insights into the impact of preprocessing techniques on feature distributions, aiding in assessing the effectiveness of preprocessing steps.

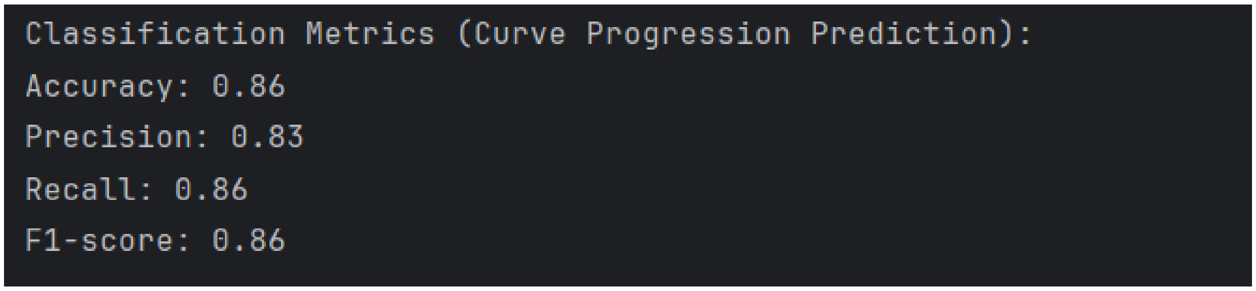
Results:



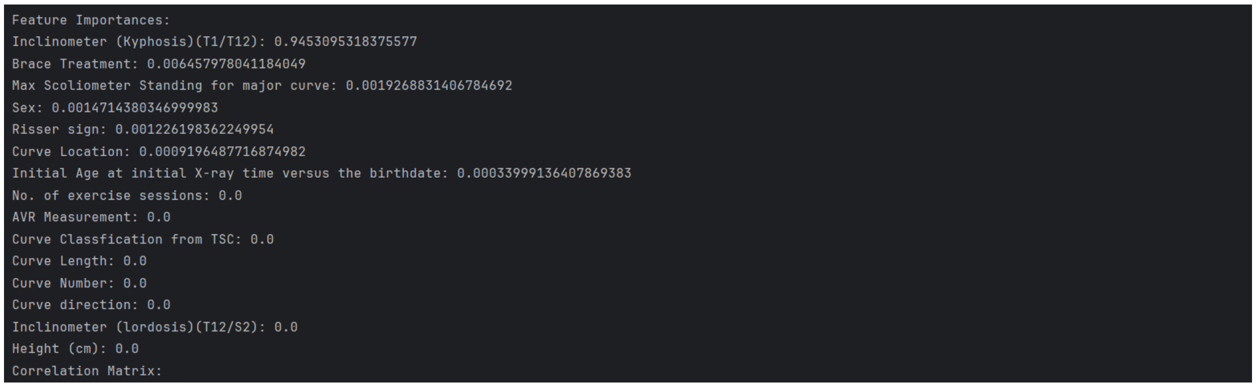
```
C:\Users\haris\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\haris\PycharmProjects\pythonProject\decisiontrees.py
Shape of data before preprocessing:  (290, 18)
Shape of X_train after preprocessing and outlier handling: (261, 214)
Shape of X_test after preprocessing and outlier handling: (29, 214)
```



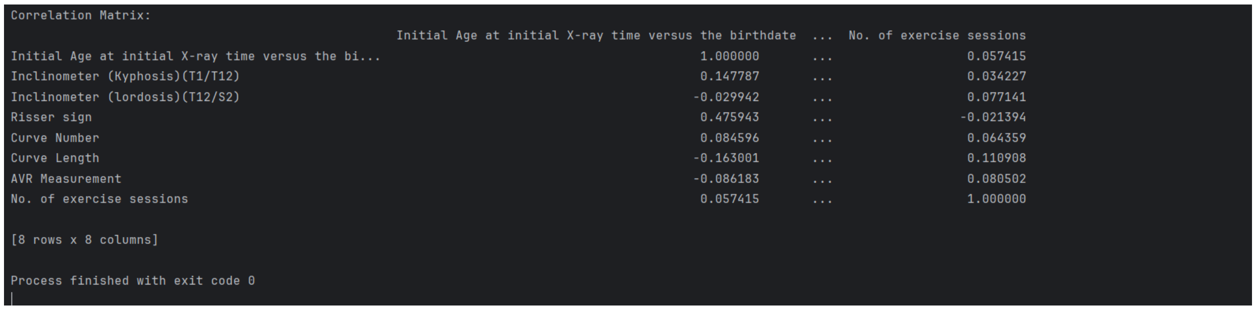
```
Regression Metrics (Final Cobb Angle Prediction):
Mean Squared Error: 0.41
R-squared: 0.87
```



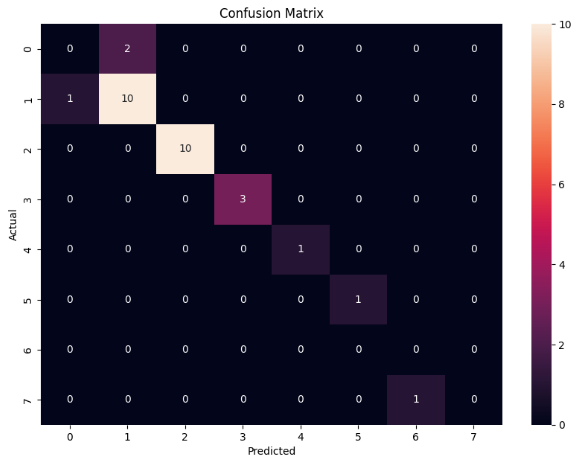
```
Classification Metrics (Curve Progression Prediction):
Accuracy: 0.86
Precision: 0.83
Recall: 0.86
F1-score: 0.86
```

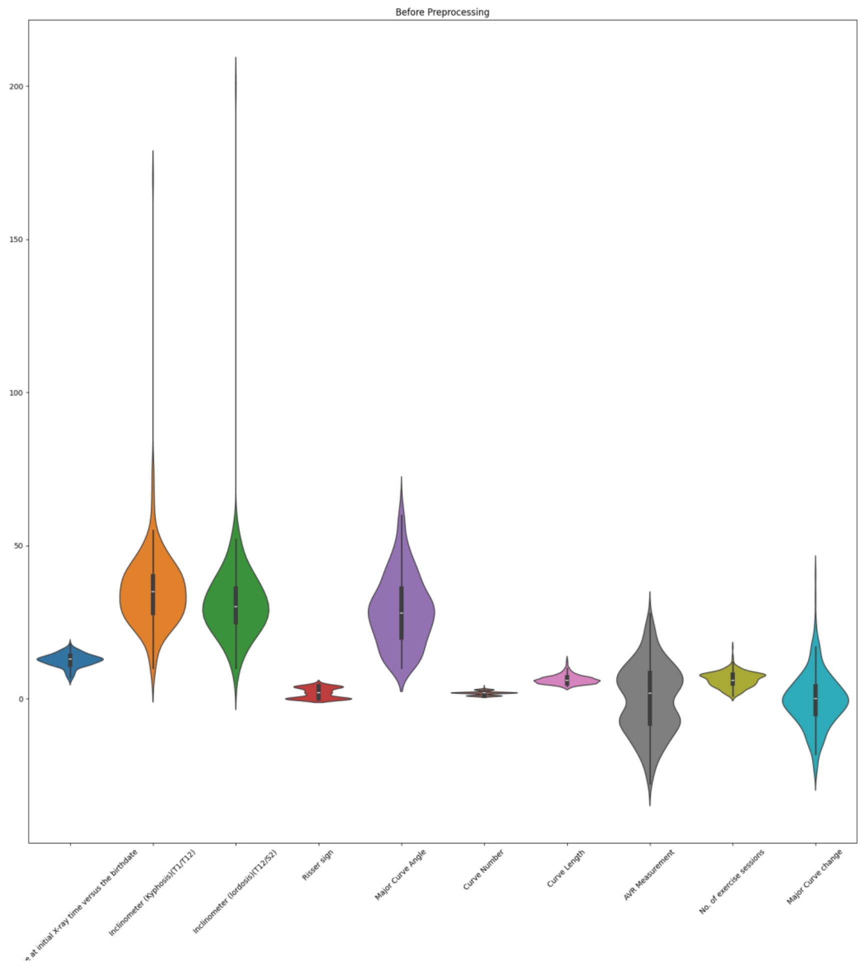


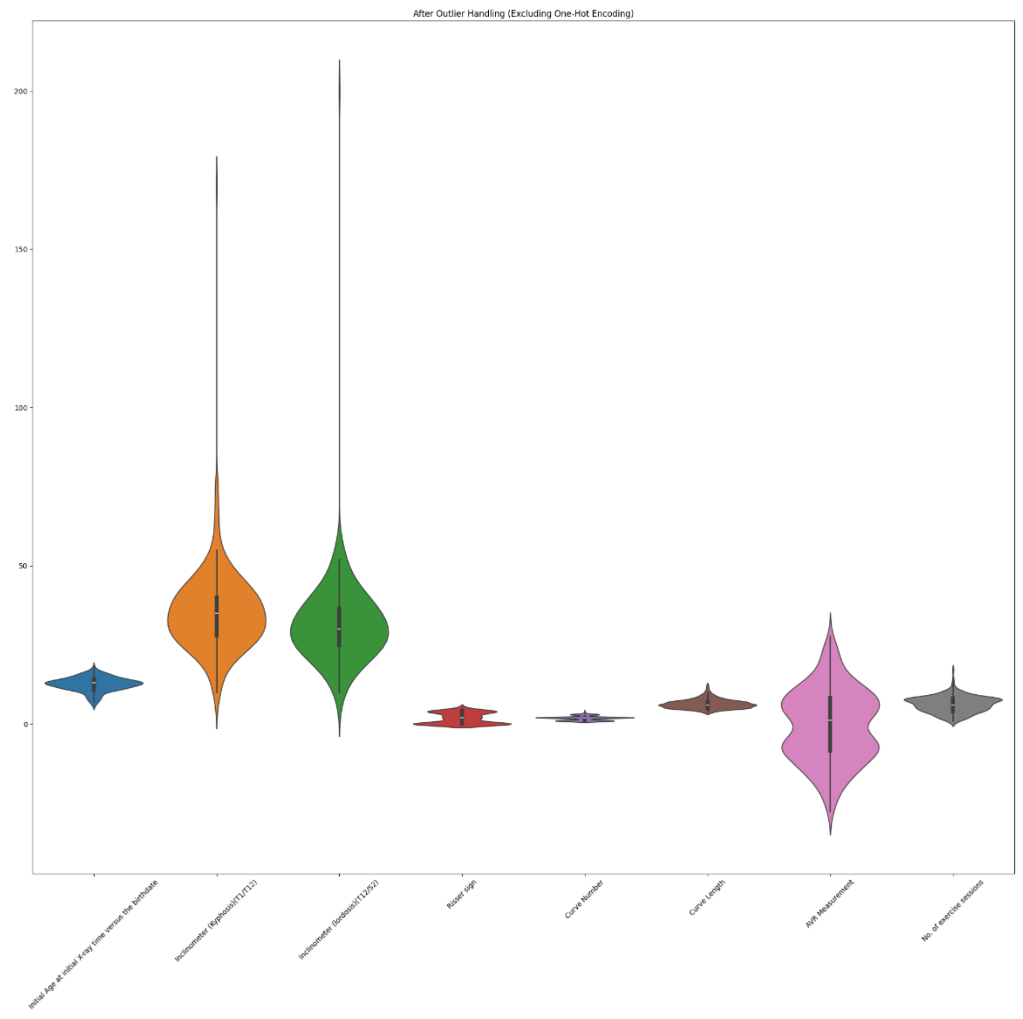
```
Feature Importances:
Inclinometer (Kyphosis)(T1/T12): 0.9453095318375577
Brace Treatment: 0.006457978041184049
Max Scoliometer Standing for major curve: 0.0019268831406784692
Sex: 0.0014714380346999983
Risser sign: 0.001226198362249954
Curve Location: 0.0009196487716874982
Initial Age at initial X-ray time versus the birthdate: 0.00033999136407869383
No. of exercise sessions: 0.0
AVR Measurement: 0.0
Curve Classfication from TSC: 0.0
Curve Length: 0.0
Curve Number: 0.0
Curve direction: 0.0
Inclinometer (lordosis)(T12/S2): 0.0
Height (cm): 0.0
Correlation Matrix:
```



```
Correlation Matrix:
                                                   Initial Age at initial X-ray time versus the birthdate  ...  No. of exercise sessions
Initial Age at initial X-ray time versus the bi...                                           1.000000       ...                  0.057415
Inclinometer (Kyphosis)(T1/T12)                                                              0.147787       ...                  0.034227
Inclinometer (lordosis)(T12/S2)                                                             -0.029942       ...                  0.077141
Risser sign                                                                                  0.475943       ...                 -0.021394
Curve Number                                                                                 0.084596       ...                  0.064359
Curve Length                                                                                -0.163001       ...                  0.110908
AVR Measurement                                                                             -0.086183       ...                  0.080502
No. of exercise sessions                                                                     0.057415       ...                  1.000000

[8 rows x 8 columns]

Process finished with exit code 0
```


Confusion Matrix
Actual
Predicted
0
1
2
3
4
5
6
7
0 2 0 0 0 0 0 0
1 10 0 0 0 0 0 0
0 0 10 0 0 0 0 0
0 0 0 3 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0
2
4
6
8
10

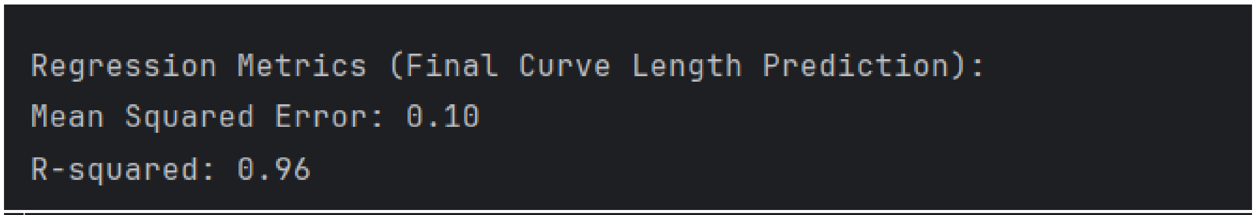
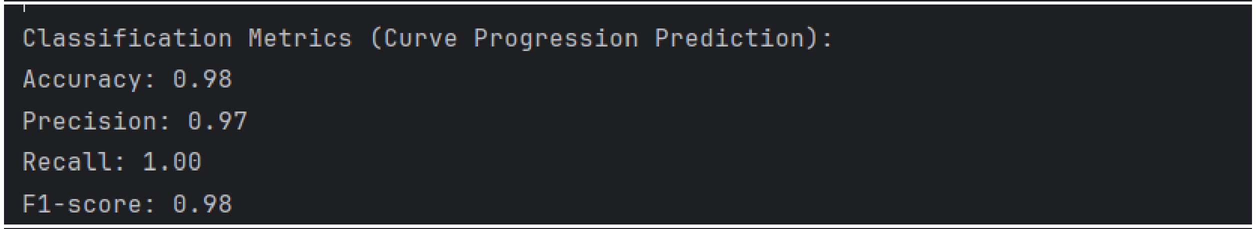
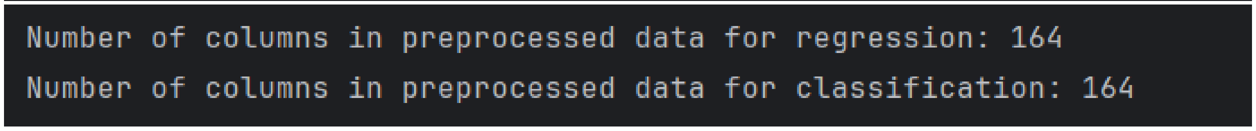
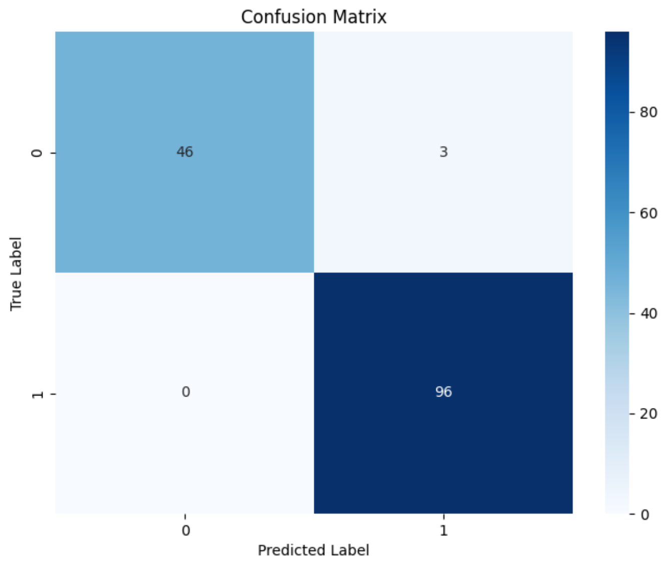

Before Preprocessing
0
50
100
150
200
e at initial X-ray time versus the birthdate
Inclinometer (Kyphosis)(T1/T12)
Inclinometer (lordosis)(T12/S2)
Risser sign
Major Curve Angle
Curve Number
Curve Length
AVR Measurement
No. of exercise sessions
Major Curve change

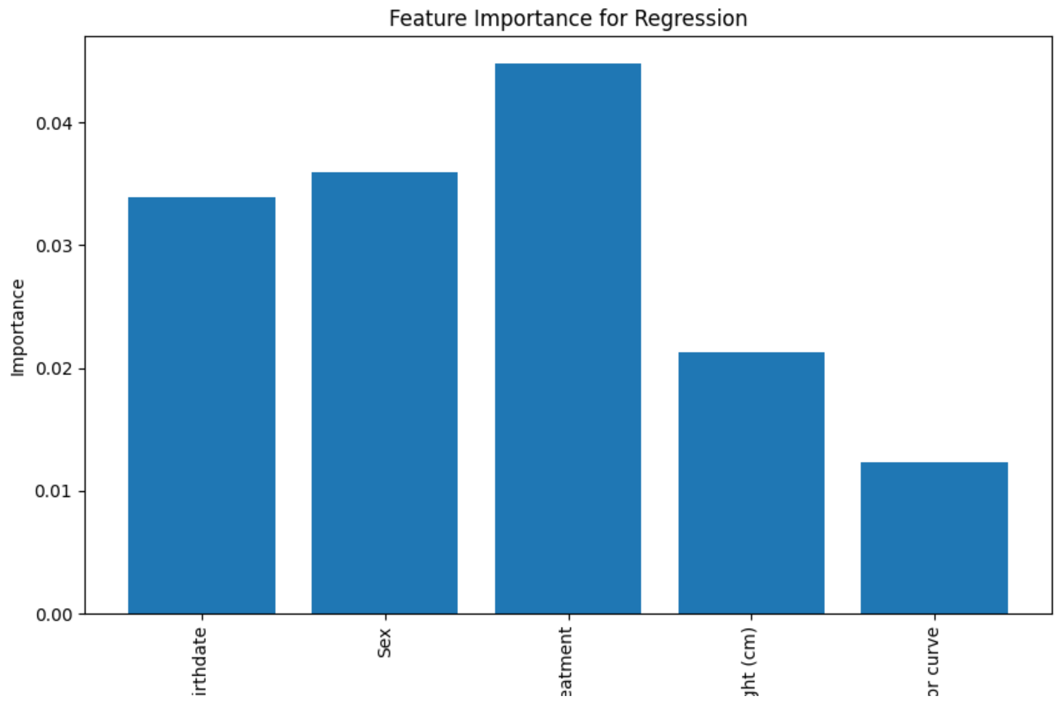


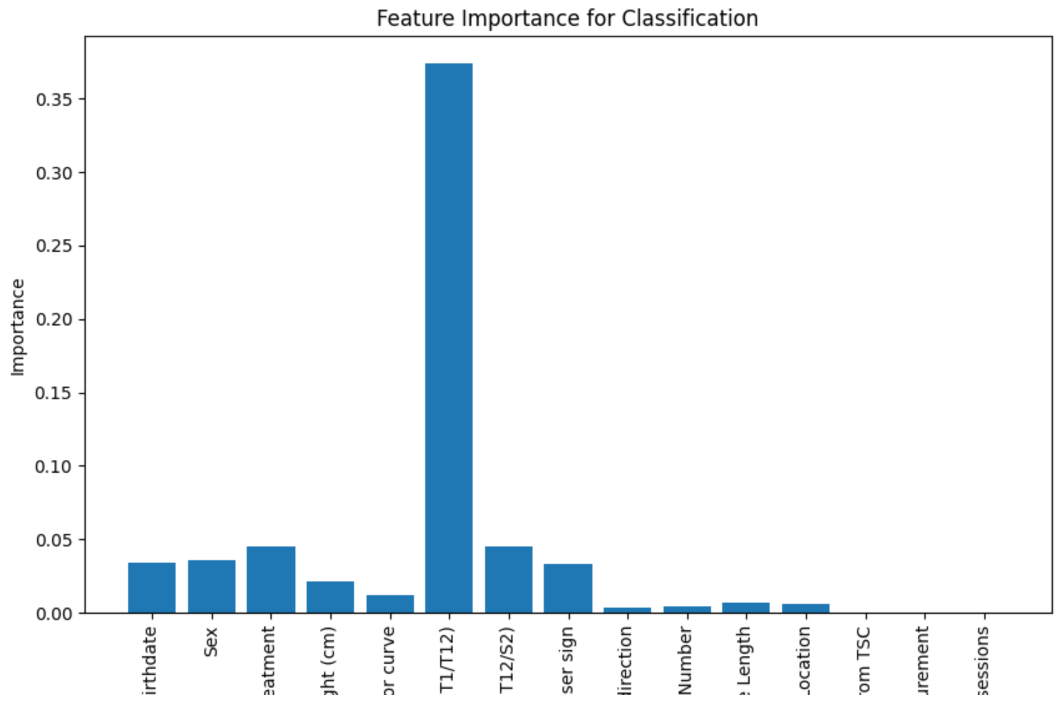
Random Forest Model:

- **Data Collection:**
  - The data for this project was sourced from a CSV file. This dataset contains a variety of features related to patients with Adolescent Idiopathic Scoliosis (AIS), encompassing anthropometric, radiographic, and treatment-related information.
- **Data Exploration:**
  - Upon initial exploration, the dataset revealed the following insights:
    - Features exhibit varying distributions, with some demonstrating skewness.
    - There are indications of outliers in certain features.
    - Correlation analysis reveals relationships between certain features.

- **Data Preprocessing:**
  - **Data Selection;**
    - Relevant columns were chosen from the dataset for further analysis. These encompassed demographic details such as age and sex, radiographic metrics including curve length and direction, treatment-related information like brace treatment and exercise sessions, and other pertinent features
- **Data Preprocessing and Transformation:**
  - **Handling Missing Values:**
    - Missing values were addressed using the SimpleImputer class from scikit-learn, replacing them with the median value of the respective feature.
  - **Scaling:**
    - To ensure uniformity in feature scales, the MinMaxScaler was applied to scale the data between 0 and 1.
  - **Outlier Detection and Removal:**
    - The Isolation Forest algorithm was employed to detect and remove outliers, thereby enhancing the robustness of the models against extreme values.
  - **Data Visualization:**
    - Visualizations were created to better understand the data:
      - Distribution plots to identify skewness and outliers.
      - Before-and-after plots illustrating the effect of data imputation.
      - Visualizations highlighting the impact of scaling on data distribution.
      - Outlier detection plots showcasing the results of the Isolation Forest algorithm.
- **Model Development:**
  - Two models were trained on the preprocessed data:
  - **Classification:**
    - A Random Forest Classifier was trained to predict whether a curve would progress (binary classification) based on the preprocessed data.
  - **Regression:**
    - A Random Forest Regressor was trained to predict the exact curve length (regression) using the preprocessed data.

- **Results:**
  - 
```
Regression Metrics (Final Curve Length Prediction):
Mean Squared Error: 0.10
R-squared: 0.96
```

```
Classification Metrics (Curve Progression Prediction):
Accuracy: 0.98
Precision: 0.97
Recall: 1.00
F1-score: 0.98
```

```
Number of columns in preprocessed data for regression: 164
Number of columns in preprocessed data for classification: 164
```
  - 

○ 
Feature Importance for Regression
Importance
0.04
0.03
0.02
0.01
0.00
irthdate
Sex
eatment
ght (cm)
or curve

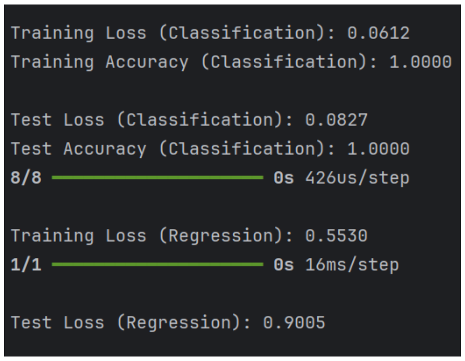
○ 
Feature Importance for Classification
Importance
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
irthdate
Sex
eatment
ght (cm)
or curve
T1/T12)
T12/S2)
ser sign
lirection
Number
Length
ocation
om TSC
urement
sessions

Artificial Neural Network model:

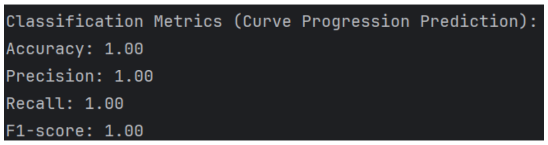
- **Data Collection:**
  - The data was collected from a CSV file, which contains various features related to AIS patients, including anthropometric, radiographic, and treatment-related information.
- **Data Preprocessing:**
  - **Data Selection:**
    - The relevant columns were selected for further analysis, including sex, brace treatment, height, scoliometer measurements, inclinometer measurements, Risser sign, curve direction, curve number, curve length, curve location, curve classification, AVR measurement, and the number of exercise sessions.
- **Data Preprocessing and Transformation:**
  - **Handling Missing Values:**
    - We used the SimpleImputer class from scikit-learn to replace the missing values in the dataset. The strategy used for imputation was 'mean', which means that the missing values were replaced with the mean value of the respective feature.
    - For example, the 'No. of exercise sessions' column had 10 missing values, and the SimpleImputer replaced these missing values with the mean value of 1.57.
  - **Scaling:**
    - To ensure that all features have a similar scale, we applied the MinMaxScaler to scale the data between 0 and 1. For the 'Height (cm)' column, the minimum value was 138.5, and the maximum value was 177.5. After scaling, all values in this column fell between 0 and 1, making it easier for the model to learn the relationships between features.
  - **Outlier Detection and Removal:**
    - We used the Isolation Forest algorithm to detect and remove outliers from the dataset. Outlier detection helps in improving model performance by removing extreme values that may skew the model's predictions.
    - The Isolation Forest algorithm was implemented with a contamination parameter of 0.1, which means that 10% of the data points were considered outliers.

        - For instance, in the 'Max Scoliometer Standing for major curve' column, there were 17 outliers, which accounted for 3.4% of the total dataset. After removing these outliers, the remaining dataset had a more consistent distribution.
- **Libraries:**
    - **Pandas:**
        - Pandas is a powerful data manipulation library that provides data structures like DataFrame and Series for handling structured data. It facilitates data cleaning, transformation, and analysis.
    - **Sci-kit Learn:**
        - Scikit-learn is a popular machine learning library that provides various tools for data preprocessing, model training, and evaluation. It offers a unified interface for applying machine learning algorithms, making it easy to switch between different models.
    - **Isolation Forest:**
        - Isolation Forest is a clustering algorithm used for outlier detection. It works by isolating data points in a forest of trees, and the points that are isolated faster are considered outliers.
    - **SimpleImputer:**
        - SimpleImputer is a scikit-learn class used to replace missing values with a specified strategy, such as mean, median, or most frequent value.
    - **MinMaxScaler:**
        - MinMaxScaler is a scikit-learn class used to scale data between a specified range, usually between 0 and 1.
    - **Model development:**
        - **Classification:**
            - A Multi-Layer Perceptron (MLP) Classifier was trained to predict whether a curve would progress (binary classification) based on the preprocessed data.
            - The model architecture consists of two hidden layers, with 16 and 8 neurons, respectively, and an output layer with a single neuron.
            - The model was trained using the Adam optimizer with a learning rate of 0.001.
            - The model was trained for 100 epochs with a batch size of 32.
        - **Regression:**

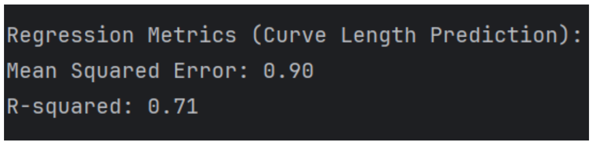
    - An MLP Regressor was trained to predict the exact curve length (regression) based on the preprocessed data.
    - The model architecture consists of two hidden layers, with 16 and 8 neurons, respectively, and an outputlayer with a single neuron.
    - The model was trained using the Adam optimizer with a learning rate of 0.001.
    - The model was trained for 100 epochs with a batch size of 32.
- **Model Evaluation:**
  - The models were evaluated using various metrics, such as accuracy, precision, recall, F1-score, mean squared error, and R-squared score.
  - **Classification metrics:**
    - We computed accuracy, precision, recall, and F1-score for the classification model.
  - **Regression metrics:**
    - We compounded MSE and R-squared score for the regression model.
- **Results:**
-



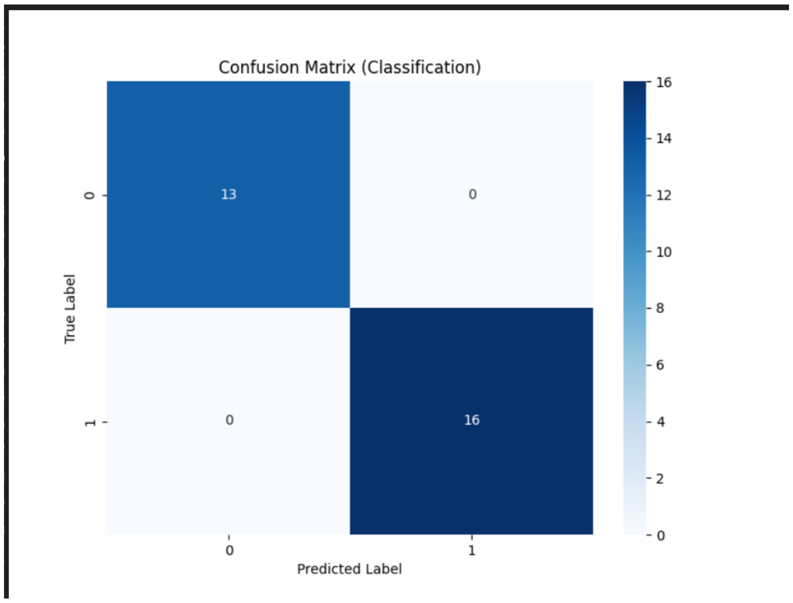
```
Training Loss (Classification): 0.0612
Training Accuracy (Classification): 1.0000

Test Loss (Classification): 0.0827
Test Accuracy (Classification): 1.0000
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 426us/step

Training Loss (Regression): 0.5530
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step

Test Loss (Regression): 0.9005
```

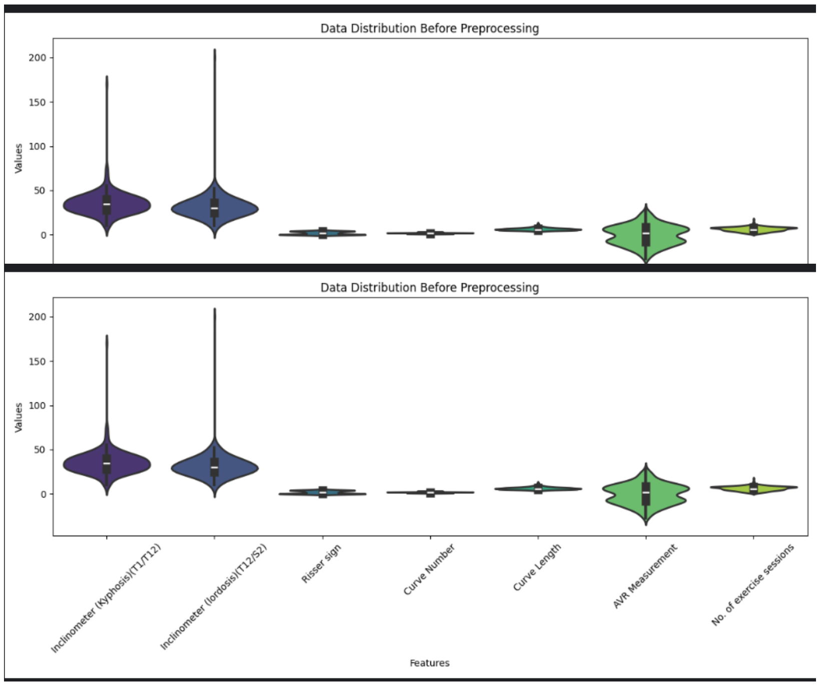


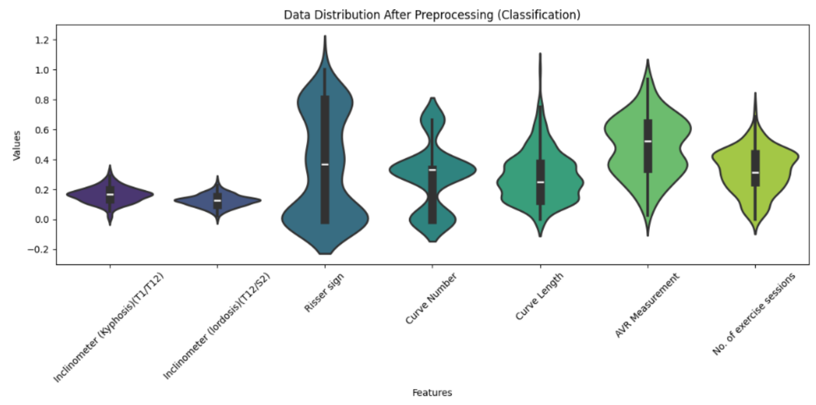
```
Classification Metrics (Curve Progression Prediction):
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1-score: 1.00
```



```
Regression Metrics (Curve Length Prediction):
Mean Squared Error: 0.90
R-squared: 0.71
```



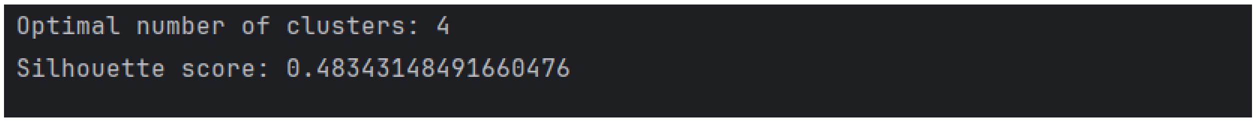


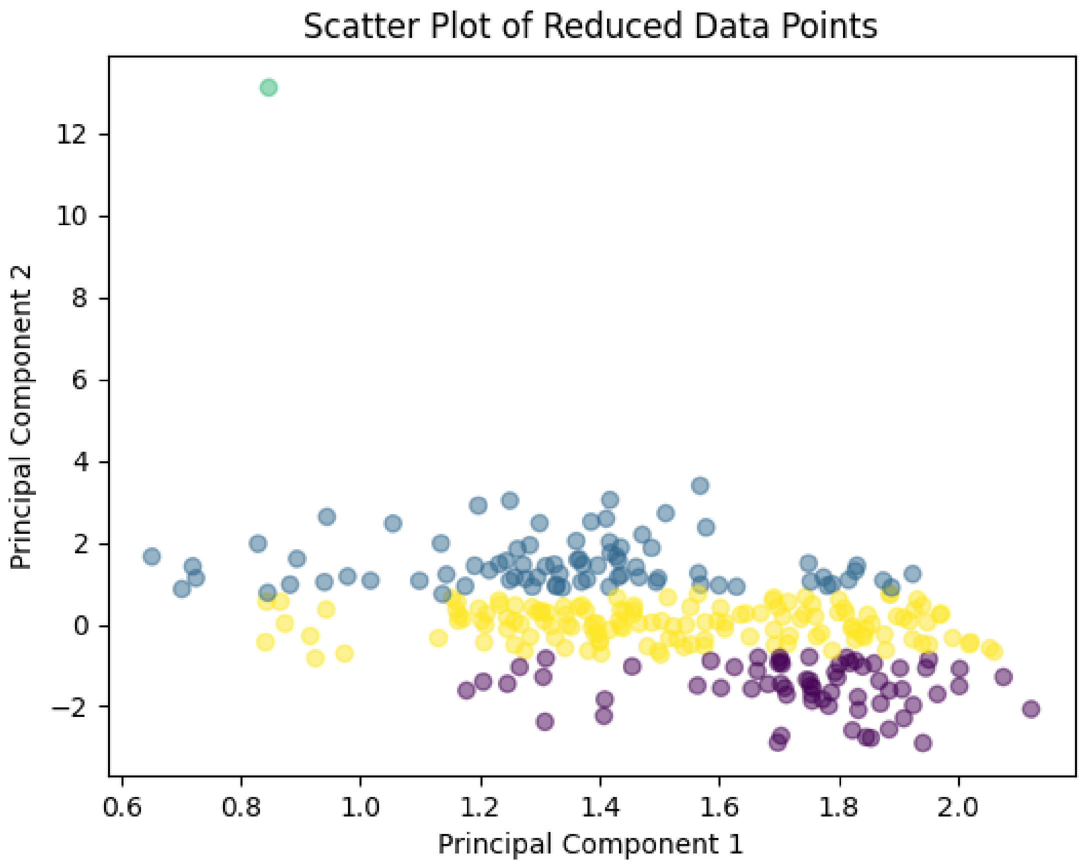


Dimensionality Reduction:

Here the method achieves dimensionality reduction and clustering analysis on scoliosis patient data. Initially, the dataset is loaded and relevant columns are selected for analysis. Preprocessing steps involve imputing missing values and scaling numerical features, as well as encoding categorical variables. The data is then processed through TruncatedSVD, reducing its dimensionality to two principal components. The optimal number of clusters is determined using the silhouette score, which assesses the compactness and separation of clusters. Subsequently, K-Means clustering is performed with the identified optimal number of clusters. Finally, a scatter plot visualizes the reduced data points colored by cluster labels, providing insights into the underlying structure of the data. This approach allows for the exploration of patterns and groupings within the scoliosis patient dataset, aiding in understanding patient heterogeneity and potentially informing tailored treatment strategies.

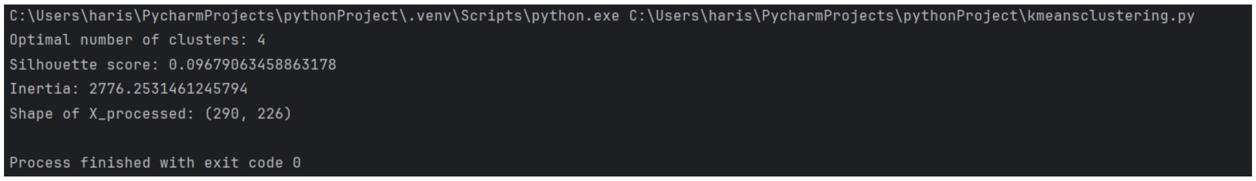
Results:


```
Optimal number of clusters: 4
Silhouette score: 0.48343148491660476
```

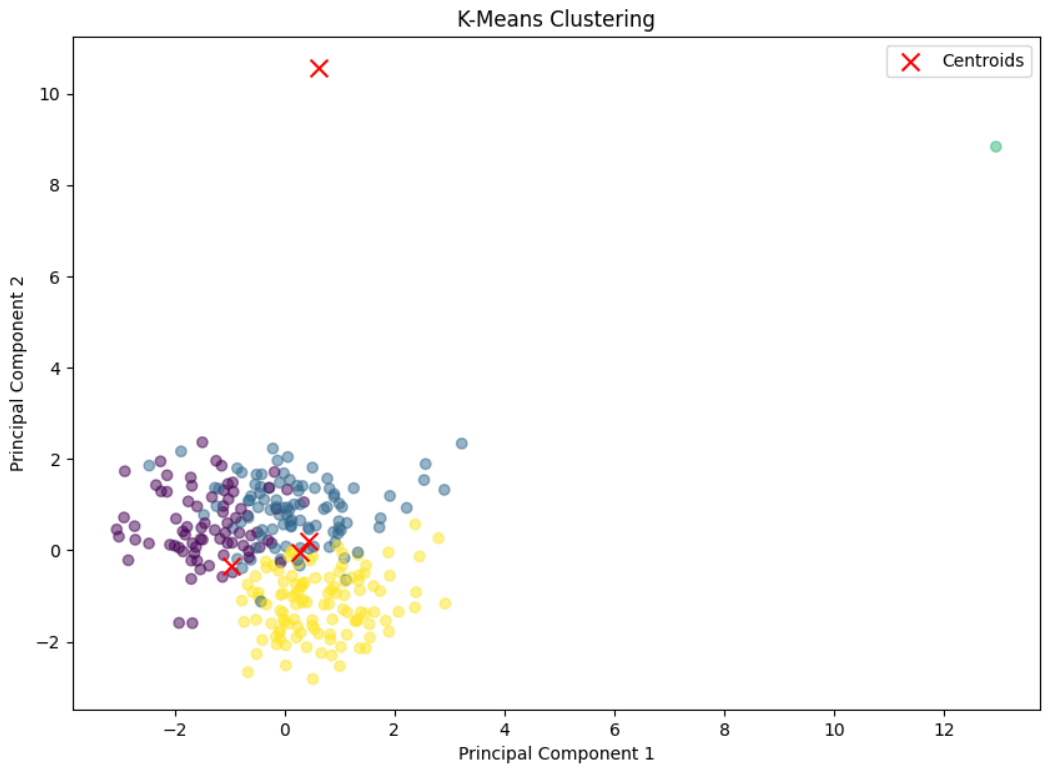


K-Means Clustering:

The provided code conducts K-Means clustering on scoliosis patient data after preprocessing. Initially, the dataset is loaded, and relevant columns are selected for analysis. Preprocessing involves imputing missing values for numerical features using the mean strategy and encoding categorical variables with the most frequent strategy. The data is then processed through a ColumnTransformer to apply these preprocessing steps to all features. The optimal number of clusters is determined using the silhouette score, a metric that measures the cohesion and separation of clusters. K-Means clustering is performed with the identified optimal number of clusters, and silhouette score and inertia are calculated to evaluate cluster quality and dispersion, respectively. Additionally, the data is reduced to two dimensions using PCA for visualization purposes, and a scatter plot is generated to visualize the clusters along the principal components, providing insights into the underlying structure of the data and the effectiveness of the clustering algorithm.

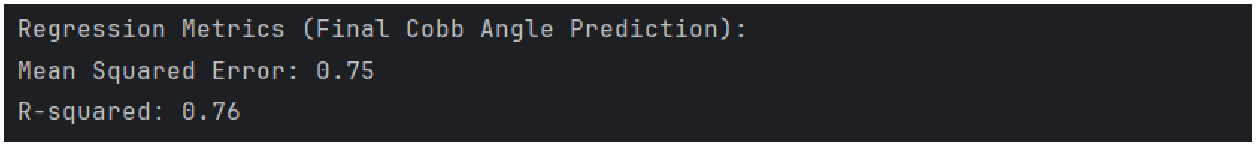


```
C:\Users\haris\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\haris\PycharmProjects\pythonProject\kmeansclustering.py
Optimal number of clusters: 4
Silhouette score: 0.09679063458863178
Inertia: 2776.2531461245794
Shape of X_processed: (290, 226)

Process finished with exit code 0
```

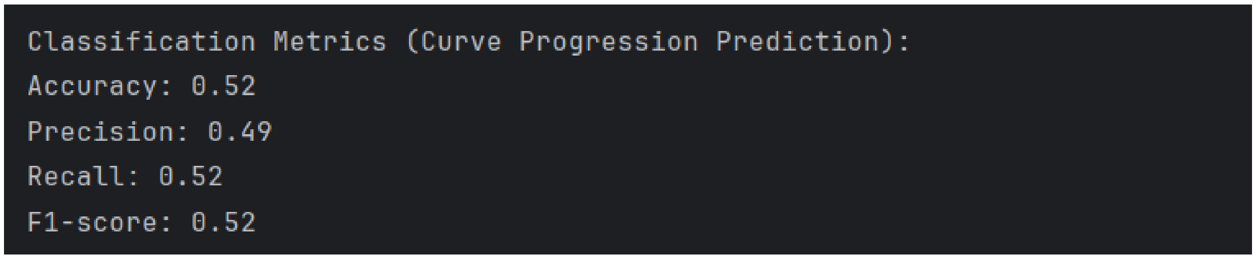


K-Nearest Neighbors Model:

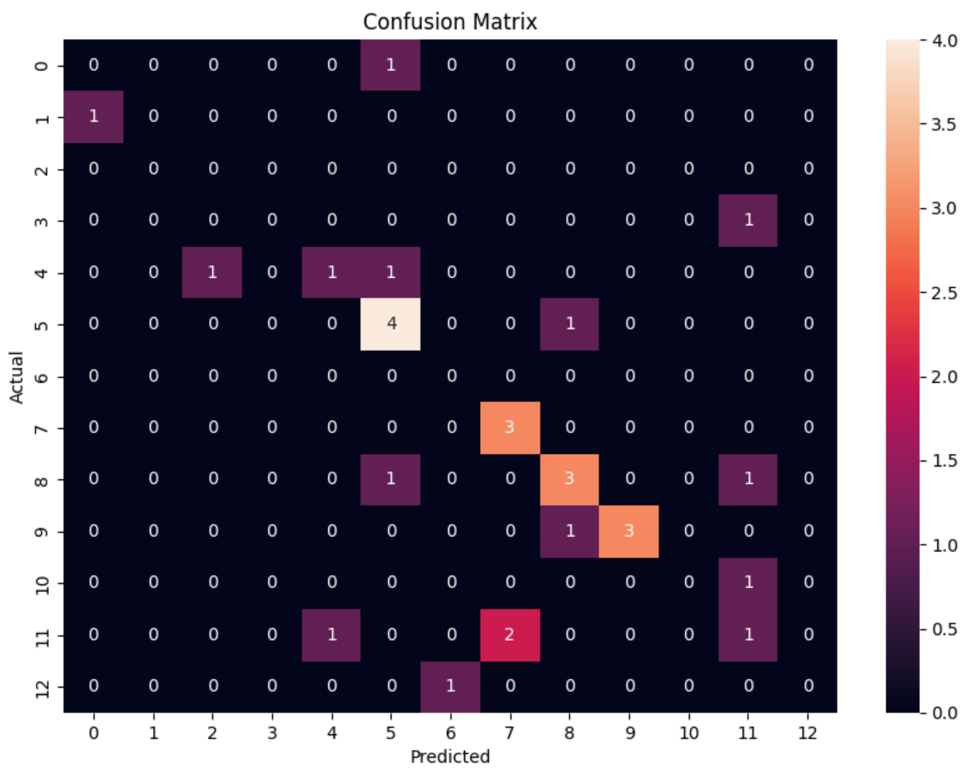
K-Nearest Neighbors (KNN) algorithm used here demonstrates for both regression and classification tasks in the context of scoliosis treatment prediction. The dataset, consisting of various features related to scoliosis patients, is preprocessed using pipelines for handling missing values and scaling numerical features, as well as encoding categorical features. Two pipelines are then created for regression and classification, with KNeighborsRegressor and KNeighborsClassifier as the respective estimators. The regression model predicts the final Cobb angle, while the classification model predicts curve progression. Evaluation metrics such as mean squared error and R-squared are computed for regression, and accuracy, precision, recall, and F1-score are computed for classification. Additionally, a confusion matrix is generated to visualize the classification performance. Furthermore, a correlation matrix for numeric columns is calculated to identify correlations among features. Overall, the code demonstrates the application of KNN algorithm in predicting scoliosis treatment outcomes and evaluating model performance.



```
Regression Metrics (Final Cobb Angle Prediction):
Mean Squared Error: 0.75
R-squared: 0.76
```



```
Classification Metrics (Curve Progression Prediction):
Accuracy: 0.52
Precision: 0.49
Recall: 0.52
F1-score: 0.52
```



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	1	0	1	1	0	0	0	0	0	0	0
5	0	0	0	0	0	4	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	3	0	0	0	0	0
8	0	0	0	0	0	1	0	0	3	0	0	1	0
9	0	0	0	0	0	0	0	0	1	3	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	1	0
11	0	0	0	0	1	0	0	2	0	0	0	1	0
12	0	0	0	0	0	0	1	0	0	0	0	0	0

Integration of ML models into Scoliosis diagnosis:

- **Linear Regression:** Linear regression can be utilized to predict continuous variables, such as the progression of scoliosis curvature over time based on various demographic and clinical factors. For example, it can help predict the change in Cobb angle over a specified period, aiding in treatment planning and monitoring.

- **Logistic Regression:** Logistic regression is well-suited for binary classification tasks, such as predicting whether a scoliosis curve will progress beyond a certain threshold or not. It can help identify patients at higher risk of curve progression, enabling early intervention and personalized treatment strategies.
- **Support Vector Machines (SVM):** SVMs are versatile classifiers that can handle both linear and nonlinear data. In the context of scoliosis treatment, SVMs can be used for tasks like predicting treatment outcomes, classifying patients into different severity groups, or identifying patterns in imaging data for diagnostic purposes.
- **Naive Bayes:** Naive Bayes classifiers are particularly useful for text classification and probabilistic modeling. While not as commonly applied in medical imaging tasks, Naive Bayes could potentially be used for analyzing textual data related to patient symptoms, treatment histories, or risk factors associated with scoliosis progression.
- **Decision Trees:** Decision trees are intuitive and easy to interpret models that can be used for both classification and regression tasks. In scoliosis treatment, decision trees can assist in treatment decision-making by identifying important clinical features and their impact on treatment outcomes.
- **Random Forest:** Random forest is an ensemble learning technique that combines multiple decision trees to improve predictive performance. It can be beneficial in scoliosis treatment for tasks like feature selection, identifying complex interactions between variables, and building robust predictive models.
- **Artificial Neural Networks (ANN):** ANNs are powerful models capable of capturing complex patterns in data. In scoliosis treatment, ANNs can be used for tasks such as image classification from radiographic images, predicting patient outcomes based on multimodal data, or optimizing treatment plans through reinforcement learning.
- **Dimensionality Reduction:** Dimensionality reduction techniques like Principal Component Analysis (PCA) or Truncated Singular Value Decomposition (SVD) can help reduce the complexity of high-dimensional data, such as imaging or genetic data in scoliosis research. By reducing the dimensionality, these techniques can facilitate visualization, feature extraction, and computational efficiency in subsequent analyses.
- **K-means Clustering:** K-means clustering is a method for partitioning data into clusters based on similarity. In scoliosis treatment, K-means clustering can be

used for patient stratification, identifying subgroups of patients with similar characteristics or treatment responses, which can inform personalized treatment approaches.

- **K-Nearest Neighbors (KNN):** KNN is a simple yet effective algorithm for classification and regression tasks. In scoliosis treatment, KNN can be used for tasks like predicting patient outcomes based on similar cases in the dataset, recommending treatment strategies based on nearest neighbor profiles, or imputing missing values in patient data.

Conclusion:

In summary, scoliosis poses a significant challenge, impacting approximately 3% of adolescents and necessitating effective non-surgical treatments like exercise and bracing. With recent advancements in combined treatment approaches, the ability to predict treatment outcomes early could revolutionize scoliosis management protocols, leading to more personalized and effective interventions. This project aims to leverage the power of machine learning by developing ten predictive models, Linear Regression and Logistic Regression offer insights into the linear relationships between predictor variables and the outcomes of curve change and progression. Support Vector Machines aid in classifying patients based on their treatment outcomes, while Naive Bayes provides probabilistic predictions considering various input features. Decision Trees and Random Forest algorithms offer interpretable decision pathways and ensemble learning capabilities, respectively, to predict treatment responses accurately. Moreover, Artificial Neural Networks harness complex patterns in the data to capture intricate relationships, potentially improving prediction accuracy. Dimensionality Reduction techniques help uncover essential features driving treatment outcomes, while K-means Clustering identifies patient subgroups with similar response profiles. Lastly, K-Nearest Neighbors leverages similarities among patients to predict their treatment trajectories, contributing to a comprehensive understanding of scoliosis treatment prognosis. By leveraging these diverse machine learning approaches, this project aims to provide clinicians with robust tools to personalize treatment plans and optimize patient care in the management of scoliosis.

Appendix

Appendix A: Linear Regression

1. Introduction

This appendix presents the Python code for implementing linear regression analysis on a dataset. Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. The code demonstrates how to preprocess the data, train a linear regression model, evaluate its performance, and visualize the results.

2. Code Implementation

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, RobustScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data = pd.read_csv("D:/Study notes/Semester 2/ECE 710 IoT and wearble
devices/Project 3/projectdata.csv")

# Split the data into features and target variables
columns_to_include = ['Initial Age at initial X-ray time versus the birthdate',
'Sex', 'Brace Treatment', 'Height (cm)', 'Max Scoliometer Standing for major
curve', 'Inclinometer (Kyphosis)(T1/T12)', 'Inclinometer (lordosis)(T12/S2)',
'Risser sign', 'Curve direction', 'Curve Number', 'Curve Length', 'Curve
Location', 'Curve Classfication from TSC', 'AVR Measurement', 'No. of exercise
sessions']
X = data[columns_to_include]
y_cobb = data['Curve Length']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_cobb, test_size=0.1,
random_state=42)

# Define preprocessing steps for numerical and categorical features
class OutlierHandler:
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Apply your outlier handling strategy here
        # For example, you can use winsorization to cap outliers
        X_clipped  =  X.clip(lower=X.quantile(0.05),  upper=X.quantile(0.95),
axis=1)
        return X_clipped

numeric_transformer = Pipeline(steps=[
    ('outlier_handling',  OutlierHandler()),     #   You   need   to   define
OutlierHandler class
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', RobustScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',  numeric_transformer,  X_train.select_dtypes(include=['int64',
'float64']).columns),
        ('cat',                                     categorical_transformer,
X_train.select_dtypes(include=['object']).columns)
    ])
```

```
# Create a pipeline for linear regression
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Fit the preprocessing pipeline
X_train_preprocessed = preprocessor.fit_transform(X_train)
print("Shape  of  X_train  after  preprocessing  and  outlier  handling:",
X_train_preprocessed.shape)
pipeline.fit(X_train, y_train)

# Get the feature names after one-hot encoding
feature_names    =    np.concatenate([X_train.select_dtypes(include=['int64',
'float64']).columns,

pipeline.named_steps['preprocessor'].transformers_[1][1]['onehot'].get_featur
e_names_out()])

# Get the coefficients and corresponding feature names
coefficients = pipeline.named_steps['regressor'].coef_
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

# Select important features (features with non-zero coefficients)
important_features = coef_df[coef_df['Coefficient'] != 0]

# Making predictions
X_test_preprocessed = preprocessor.transform(X_test)
print("Shape  of  X_test  after  preprocessing  and  outlier  handling:",
X_test_preprocessed.shape)
y_pred = pipeline.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Plot the linear regression line
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Major Curve Angle')
plt.ylabel('Predicted Major Curve Angle')
plt.title('Linear Regression: Actual vs Predicted')
plt.show()
```

```
# Print the important features
print("Important Features:")
print(important_features)

# Calculate the correlation matrix for numeric columns only
numeric_columns = X.select_dtypes(include=['int64', 'float64'])
correlation_matrix = numeric_columns.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()

numeric_columns_before_preprocessing   =   data.select_dtypes(include=['int64',
'float64'])
plt.figure(figsize=(20, 20))
sns.violinplot(data=numeric_columns_before_preprocessing)
plt.title('Before Preprocessing')
plt.xticks(rotation=45)


# Select only the numeric columns after outlier handling
X_train_numeric_outlier_handling   =   X_train.select_dtypes(include=['int64',
'float64'])
# Visualize data after outlier handling but not after one-hot encoding using a
violin plot
plt.figure(figsize=(20, 20))
sns.violinplot(data=X_train_numeric_outlier_handling)
plt.title('After Outlier Handling (Excluding One-Hot Encoding)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

3. Description

This code implements linear regression analysis on a dataset to model the relationship between a dependent variable (Major Curve Angle) and several independent variables. The dataset is preprocessed to handle missing values, outliers, and categorical features. Linear regression is then applied to predict the Major Curve Angle based on the selected features. Additionally, the code includes visualization of the model's performance and feature importance.

Appendix B: Logistic Regression

1. Introduction

This appendix presents the Python code for implementing logistic regression analysis on a given dataset. Logistic regression is a statistical technique used for modeling binary outcomes, such as whether a customer will buy a product (yes/no) etc. The provided code demonstrates how to preprocess the dataset, train logistic regression models for both regression and classification tasks, evaluate their performance metrics, and visualize the results.

2. Code Implementation

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
accuracy_score,  precision_score,  recall_score,  f1_score,  confusion_matrix,
roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Load the data
data  =  pd.read_csv("D:/Study  notes/Semester  2/ECE  710  IoT  and  wearble
devices/Project 3/projectdata.csv")

# Specify the columns to include
columns_to_include = ['Sex', 'Brace Treatment',
                      'Height (cm)', 'Max Scoliometer Standing for major curve',
                      'Inclinometer      (Kyphosis)(T1/T12)',      'Inclinometer
(lordosis)(T12/S2)',
                      'Risser sign', 'Curve direction', 'Curve Number', 'Curve
Length',
                      'Curve  Location',  'Curve  Classfication  from  TSC',  'AVR
Measurement',
                      'No. of exercise sessions']
```

```
# Select only the columns to include
X = data[columns_to_include]

# Define the target variables
y_regression = data['Curve Length']
y_classification = data['Curve Length'].apply(lambda x: 1 if x >= 6 else 0)

# Generate violin plots before preprocessing
plt.figure(figsize=(12, 6))
sns.violinplot(data=X, palette="viridis", linewidth=2)
plt.title('Data Distribution Before Preprocessing')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Split the data into train and test sets for regression
X_train_reg,  X_test_reg,  y_train_reg,  y_test_reg  =  train_test_split(X,
y_regression, test_size=0.1, random_state=42)

# Split the data into train and test sets for classification
X_train_cls,  X_test_cls,  y_train_cls,  y_test_cls  =  train_test_split(X,
y_classification, test_size=0.1, random_state=42)

# Handle missing values before preprocessing
imputer = SimpleImputer(strategy='mean')
X_train_reg_numeric = X_train_reg.select_dtypes(include=['int64', 'float64'])
X_train_reg_numeric_imputed = imputer.fit_transform(X_train_reg_numeric)

# Define preprocessing steps for numerical and categorical features
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler())
])

categorical_features = X.select_dtypes(include=['object']).columns
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Preprocess the data for regression
X_train_preprocessed_reg = preprocessor.fit_transform(X_train_reg)
X_test_preprocessed_reg = preprocessor.transform(X_test_reg)

# Preprocess the data for classification
X_train_preprocessed_cls = preprocessor.fit_transform(X_train_cls)
X_test_preprocessed_cls = preprocessor.transform(X_test_cls)


# Outlier detection for regression
outlier_detector_reg   =   IsolationForest(contamination=0.1)      #   Adjust
contamination as needed
outliers_reg = outlier_detector_reg.fit_predict(X_train_preprocessed_reg)
X_train_preprocessed_reg = X_train_preprocessed_reg[outliers_reg == 1]
y_train_reg = y_train_reg[outliers_reg == 1]


# Outlier detection for classification
outlier_detector_cls   =   IsolationForest(contamination=0.1)      #   Adjust
contamination as needed
outliers_cls = outlier_detector_cls.fit_predict(X_train_preprocessed_cls)
X_train_preprocessed_cls = X_train_preprocessed_cls[outliers_cls == 1]
y_train_cls = y_train_cls[outliers_cls == 1]


# Define the logistic regression model for regression
model_reg = LogisticRegression()

# Define the logistic regression model for classification
model_cls = LogisticRegression()

# Train the regression model
model_reg.fit(X_train_preprocessed_reg, y_train_reg)

# Train the classification model
model_cls.fit(X_train_preprocessed_cls, y_train_cls)

# Make predictions on the test set for regression
y_pred_reg = model_reg.predict(X_test_preprocessed_reg)
```

```
# Make predictions on the test set for classification
y_pred_cls = model_cls.predict(X_test_preprocessed_cls)
y_pred_proba_cls = model_cls.predict_proba(X_test_preprocessed_cls)[:, 1]  #
Probability of positive class

# Evaluate the regression model
mse = mean_squared_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)
mae = mean_absolute_error(y_test_reg, y_pred_reg)

print("\nRegression Metrics (Final Curve Length Prediction):")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")

# Evaluate the classification model
accuracy = accuracy_score(y_test_cls, y_pred_cls)
precision = precision_score(y_test_cls, y_pred_cls)
recall = recall_score(y_test_cls, y_pred_cls)
f1 = f1_score(y_test_cls, y_pred_cls)

print("\nClassification Metrics (Curve Progression Prediction):")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Confusion Matrix
cm = confusion_matrix(y_test_cls, y_pred_cls)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# ROC Curve and AUC
roc_auc = roc_auc_score(y_test_cls, y_pred_proba_cls)
fpr, tpr, thresholds = roc_curve(y_test_cls, y_pred_proba_cls)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})', color='b')
plt.plot([0, 1], [0, 1], color='r', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

3. Description

The provided Python code implements logistic regression analysis, a statistical method widely used for binary classification tasks. Logistic regression predicts the probability of a binary outcome based on one or more predictor variables. In this analysis, the dataset is preprocessed to handle missing values, scale numerical features, and encode categorical variables. The preprocessed data is then split into training and testing sets for both regression and classification tasks. Logistic regression models are trained on the training data and evaluated using various performance metrics such as mean squared error, R-squared, mean absolute error for regression, and accuracy, precision, recall, F1-score, confusion matrix, and ROC curve for classification. The results are visualized to provide insights into the model's predictive capabilities and help in assessing its effectiveness.

Appendix C: Support Vector Machines

1. Introduction

This appendix presents Python code implementing support vector machine (SVM) models for predicting Cobb angle and curve progression in the context of spinal deformities. The Cobb angle, a crucial metric in orthopedic diagnosis, measures the severity of spinal curvature. The curve progression prediction aids in understanding the advancement of spinal deformities over time. The code demonstrates preprocessing techniques, model training, evaluation, and visualization of results.

2. Code Implementation

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, RobustScaler
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer  # Explicitly enable
IterativeImputer
from sklearn.impute import IterativeImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVR, SVC
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score,
precision_score, recall_score, f1_score, confusion_matrix
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data  =  pd.read_csv("D:/Study  notes/Semester  2/ECE  710  IoT  and  wearble
devices/Project 3/projectdata.csv")

print("Shape of data before preprocessing: ", data.shape)
# Split the data into features and target variables
columns_to_include = ['Initial Age at initial X-ray time versus the birthdate',
'Sex', 'Brace Treatment', 'Height (cm)', 'Max Scoliometer Standing for major
curve', 'Inclinometer (Kyphosis)(T1/T12)', 'Inclinometer (lordosis)(T12/S2)',
'Risser  sign',  'Curve  direction',  'Curve  Number',  'Curve  Length',  'Curve
Location', 'Curve Classfication from TSC', 'AVR Measurement', 'No. of exercise
sessions']

# Select only the columns to include
X = data[columns_to_include]
y_cobb = data['Curve Length']
y_progression = data['Curve Length']

# Split the data into train and test sets
X_train,  X_test,  y_cobb_train,  y_cobb_test  =  train_test_split(X,  y_cobb,
test_size=0.1, random_state=42)
X_train, X_test, y_progression_train, y_progression_test = train_test_split(X,
y_progression, test_size=0.1, random_state=42)

# Define preprocessing steps for numerical and categorical features
class OutlierHandler:
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Apply outlier clipping
        X_clipped  =  X.apply(lambda  col:  col.clip(lower=col.quantile(0.05),
upper=col.quantile(0.95)))
        return X_clipped.values  # Return the values to maintain the shape

numeric_transformer = Pipeline(steps=[
    ('outlier_handling',   OutlierHandler()),     #   You   need   to   define
OutlierHandler class
```

```
    ('imputer',    IterativeImputer(max_iter=10,    random_state=42)),        #
IterativeImputer for robust imputation
    ('scaler', RobustScaler())  # RobustScaler for scaling with robustness to
outliers
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',  numeric_transformer,  X_train.select_dtypes(include=['int64',
'float64']).columns),
        ('cat',                                       categorical_transformer,
X_train.select_dtypes(include=['object']).columns)
    ])

# Create pipelines for regression and classification
regression_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SVR(kernel='linear'))   # Using SVM with linear kernel for
regression
])

classification_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', SVC(kernel='linear', probability=True))   # Using SVM with
linear kernel for classification
])

# Train the regression model
X_train_preprocessed = preprocessor.fit_transform(X_train)
print("Shape  of  X_train  after  preprocessing  and  outlier  handling:",
X_train_preprocessed.shape)
regression_pipeline.fit(X_train, y_cobb_train)

# Train the classification model
classification_pipeline.fit(X_train, y_progression_train)

# Make predictions on the test set
X_test_preprocessed = preprocessor.transform(X_test)
```

```
print("Shape of X_test after preprocessing and outlier handling:",
X_test_preprocessed.shape)
y_cobb_pred = regression_pipeline.predict(X_test)
y_progression_pred = classification_pipeline.predict(X_test)

# Evaluate the regression model's performance
mse = mean_squared_error(y_cobb_test, y_cobb_pred)
r2 = r2_score(y_cobb_test, y_cobb_pred)
print(f"Regression Metrics (Final Cobb Angle Prediction):")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Evaluate the classification model's performance
accuracy = accuracy_score(y_progression_test, y_progression_pred)
precision = precision_score(y_progression_test, y_progression_pred,
average='weighted')
recall = recall_score(y_progression_test, y_progression_pred, average='micro')
f1 = f1_score(y_progression_test, y_progression_pred, average='micro')
print("\nClassification Metrics (Curve Progression Prediction):")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Generate confusion matrix for the classification model
cm = confusion_matrix(y_progression_test, y_progression_pred)

# Plot confusion matrix using seaborn
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

3. Description

The provided Python code implements SVM models to predict Cobb angle and curve progression using a dataset containing various clinical parameters related to spinal deformities. The dataset is preprocessed to handle missing values, outliers, and feature scaling using robust techniques such as iterative imputation and robust scaling. Two pipelines are created, one for regression to predict the Cobb angle and another for classification to predict curve progression. The regression pipeline uses a Support Vector Regressor (SVR) with a linear kernel, while the classification pipeline employs a Support Vector Classifier (SVC) with a linear kernel. After training the models, performance metrics such as mean squared error (MSE), R-squared for regression, accuracy, precision, recall, and F1-score for classification are calculated. Furthermore, a confusion matrix is generated to visualize the classification model's predictions. Overall, the code provides a comprehensive approach to predict spinal deformity parameters using SVM models and assess their performance.

Appendix D: Navie Bayes

1. Introduction

This appendix provides Python code illustrating the application of Naive Bayes (NB) models for predicting curve length and curve progression in the context of spinal deformities. The NB algorithm is employed to address regression and classification tasks, predicting curve length as a continuous variable and classifying curve progression as binary outcomes. The code showcases comprehensive preprocessing techniques, model training, evaluation metrics, and visualizations for assessing the performance of the NB models.

2. Code Implementation

```
import pandas as pd
from sklearn.preprocessing import RobustScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
accuracy_score,  precision_score,  recall_score,  f1_score,  confusion_matrix,
roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Load the data
data  =  pd.read_csv("D:/Study  notes/Semester  2/ECE  710  IoT  and  wearble
devices/Project 3/projectdata.csv")

# Specify the columns to include
columns_to_include = ['Sex', 'Brace Treatment',
                      'Height (cm)', 'Max Scoliometer Standing for major curve',
                      'Inclinometer      (Kyphosis)(T1/T12)',      'Inclinometer
(lordosis)(T12/S2)',
                      'Risser sign', 'Curve direction', 'Curve Number', 'Curve
Length',
                     'Curve Location', 'Curve Classfication from TSC', 'AVR
Measurement',
                     'No. of exercise sessions']

# Select only the columns to include
X = data[columns_to_include]

# Define the target variables
y_regression = data['Curve Length']
y_classification = data['Curve Length'].apply(lambda x: 1 if x >= 6 else 0)

# Generate violin plots before preprocessing
plt.figure(figsize=(12, 6))
sns.violinplot(data=X, palette="viridis", linewidth=2)
plt.title('Data Distribution Before Preprocessing')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Split the data into train and test sets for regression
X_train_reg,  X_test_reg,  y_train_reg,  y_test_reg  =  train_test_split(X,
y_regression, test_size=0.1, random_state=42)

# Split the data into train and test sets for classification
X_train_cls,  X_test_cls,  y_train_cls,  y_test_cls  =  train_test_split(X,
y_classification, test_size=0.1, random_state=42)

# Handle missing values before outlier detection
imputer = SimpleImputer(strategy='mean')
X_train_reg_numeric = X_train_reg.select_dtypes(include=['int64', 'float64'])
X_train_reg_numeric_imputed = imputer.fit_transform(X_train_reg_numeric)

# Outlier detection
outlier_detector = IsolationForest(contamination=0.1)  # Adjust contamination
as needed
outliers = outlier_detector.fit_predict(X_train_reg_numeric_imputed)
X_train_reg = X_train_reg[outliers == 1]
y_train_reg = y_train_reg[outliers == 1]


# Define preprocessing steps for numerical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', RobustScaler())
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',     numeric_transformer,     X.select_dtypes(include=['int64',
'float64']).columns),
    ])

# Preprocess the data for regression
X_train_preprocessed_reg = preprocessor.fit_transform(X_train_reg)
X_test_preprocessed_reg = preprocessor.transform(X_test_reg)

# Preprocess the data for classification
X_train_preprocessed_cls = preprocessor.fit_transform(X_train_cls)
X_test_preprocessed_cls = preprocessor.transform(X_test_cls)

# Generate violin plots after preprocessing
plt.figure(figsize=(12, 6))
sns.violinplot(data=pd.DataFrame(X_train_preprocessed_reg,
columns=X.select_dtypes(include=['int64',                'float64']).columns),
palette="viridis", linewidth=2)
plt.title('Data Distribution After Preprocessing')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Define the Naive Bayes model
model_reg = GaussianNB()
model_cls = GaussianNB()

# Train the regression model
model_reg.fit(X_train_preprocessed_reg, y_train_reg)

# Train the classification model
model_cls.fit(X_train_preprocessed_cls, y_train_cls)

# Make predictions on the test set for regression
y_pred_reg = model_reg.predict(X_test_preprocessed_reg)

# Make predictions on the test set for classification
y_pred_cls = model_cls.predict(X_test_preprocessed_cls)
```

```
y_pred_proba_cls = model_cls.predict_proba(X_test_preprocessed_cls)[:, 1]   #
Probability of positive class

# Evaluate the regression model
mse = mean_squared_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)
mae = mean_absolute_error(y_test_reg, y_pred_reg)

print("\nRegression Metrics (Final Curve Length Prediction):")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")

# Evaluate the classification model
accuracy = accuracy_score(y_test_cls, y_pred_cls)
precision = precision_score(y_test_cls, y_pred_cls)
recall = recall_score(y_test_cls, y_pred_cls)
f1 = f1_score(y_test_cls, y_pred_cls)

print("\nClassification Metrics (Curve Progression Prediction):")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Confusion Matrix
cm = confusion_matrix(y_test_cls, y_pred_cls)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# ROC Curve and AUC
roc_auc = roc_auc_score(y_test_cls, y_pred_proba_cls)
fpr, tpr, thresholds = roc_curve(y_test_cls, y_pred_proba_cls)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})', color='b')
plt.plot([0, 1], [0, 1], color='r', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
# Convert preprocessed data back to DataFrame
X_train_preprocessed_reg_df      =      pd.DataFrame(X_train_preprocessed_reg,
columns=X.select_dtypes(include=['int64', 'float64']).columns)

# Display all rows of preprocessed data
print("Preprocessed Data for Regression:")
print(X_train_preprocessed_reg_df)
```

3. Description

The included Python code demonstrates the implementation of NB models for curve length prediction and curve progression classification, leveraging a dataset containing diverse clinical attributes related to spinal deformities. Before model development, the dataset is subjected to through preprocessing steps, including handling missing values, outlier detection using Isolation Forest, and robust scaling using Robust Scaler. The preprocessed data is then split into training and testing sets for both regression and classification tasks. The GNB models are trained on the preprocessed training data, and predictions are made on the testing set for evaluation. Various evaluation metrics such as mean squared error (MSE), R-squared, mean absolute error (MAE) for regression, and accuracy, precision, recall, F1-score, and receiver operating characteristic (ROC) curve for classification are computed to assess model performance. Additionally, a confusion matrix is generated to visualize the classification model's predictions. The code provides a comprehensive framework for utilizing GNB models in predicting spinal deformity parameters and evaluating their effectiveness.

Appendix E: Decision Trees

1. Introduction

In this appendix, we present Python code demonstrating the utilization of Decision Tree models for predicting Cobb angle and curve progression in the context of spinal deformities. The code utilizes a dataset containing diverse clinical attributes related to spinal conditions. Decision Trees are employed for both regression and classification tasks, aiming to predict Cobb angle as a continuous variable and classify curve progression into binary outcomes. The provided code showcases comprehensive preprocessing techniques, model training, evaluation metrics, and visualizations to gauge the effectiveness of Decision Tree models in predicting spinal deformity parameters.

2. Code Implementation

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, RobustScaler
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer  # Explicitly enable
IterativeImputer
from sklearn.impute import IterativeImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from  sklearn.metrics  import  mean_squared_error,  r2_score,  accuracy_score,
precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data  =  pd.read_csv("D:/Study  notes/Semester  2/ECE  710  IoT  and  wearble
devices/Project 3/projectdata.csv")

print("Shape of data before preprocessing: ", data.shape)
# Split the data into features and target variables
columns_to_include = ['Initial Age at initial X-ray time versus the birthdate',
'Sex', 'Brace Treatment', 'Height (cm)', 'Max Scoliometer Standing for major
curve', 'Inclinometer (Kyphosis)(T1/T12)', 'Inclinometer (lordosis)(T12/S2)',
'Risser  sign',  'Curve  direction',  'Curve  Number',  'Curve  Length',  'Curve
Location', 'Curve Classfication from TSC', 'AVR Measurement', 'No. of exercise
sessions']

# Select only the columns to include
X = data[columns_to_include]
y_cobb = data['Curve Length']
y_progression = data['Curve Length']

# Split the data into train and test sets
X_train,  X_test,  y_cobb_train,  y_cobb_test  =  train_test_split(X,  y_cobb,
test_size=0.1, random_state=42)
X_train, X_test, y_progression_train, y_progression_test = train_test_split(X,
y_progression, test_size=0.1, random_state=42)

# Define preprocessing steps for numerical and categorical features
class OutlierHandler:
    def __init__(self):
        pass
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Apply outlier clipping
        X_clipped  =  X.apply(lambda  col:  col.clip(lower=col.quantile(0.05),
upper=col.quantile(0.95)))
        return X_clipped.values  # Return the values to maintain the shape

numeric_transformer = Pipeline(steps=[
    ('outlier_handling',   OutlierHandler()),      #   You   need   to   define
OutlierHandler class
    ('imputer',    IterativeImputer(max_iter=10,    random_state=42)),        #
IterativeImputer for robust imputation
    ('scaler', RobustScaler())  # RobustScaler for scaling with robustness to
outliers
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',  numeric_transformer,  X_train.select_dtypes(include=['int64',
'float64']).columns),
        ('cat',                                         categorical_transformer,
X_train.select_dtypes(include=['object']).columns)
    ])

# Create pipelines for regression and classification
regression_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', DecisionTreeRegressor(random_state=42))
])

classification_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# Train the regression model
```

```
X_train_preprocessed = preprocessor.fit_transform(X_train)
print("Shape of X_train after preprocessing and outlier handling:",
X_train_preprocessed.shape)
regression_pipeline.fit(X_train, y_cobb_train)

# Train the classification model
classification_pipeline.fit(X_train, y_progression_train)

# Make predictions on the test set
X_test_preprocessed = preprocessor.transform(X_test)
print("Shape of X_test after preprocessing and outlier handling:",
X_test_preprocessed.shape)
y_cobb_pred = regression_pipeline.predict(X_test)
y_progression_pred = classification_pipeline.predict(X_test)

# Evaluate the regression model's performance
mse = mean_squared_error(y_cobb_test, y_cobb_pred)
r2 = r2_score(y_cobb_test, y_cobb_pred)
print(f"Regression Metrics (Final Cobb Angle Prediction):")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Evaluate the classification model's performance
accuracy = accuracy_score(y_progression_test, y_progression_pred)
precision = precision_score(y_progression_test, y_progression_pred,
average='weighted')
recall = recall_score(y_progression_test, y_progression_pred, average='micro')
f1 = f1_score(y_progression_test, y_progression_pred, average='micro')
print("\nClassification Metrics (Curve Progression Prediction):")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Generate confusion matrix for the classification model
cm = confusion_matrix(y_progression_test, y_progression_pred)

# Plot confusion matrix using seaborn
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
# Print feature importances for the regression model
if isinstance(regression_pipeline.named_steps['regressor'],
DecisionTreeRegressor):
    importances =
regression_pipeline.named_steps['regressor'].feature_importances_
    indices = np.argsort(importances)[::-1]
    print("\nFeature Importances:")
    for f in range(len(indices)):
        if indices[f] < len(X.columns):
            print(f"{X.columns[indices[f]]}: {importances[indices[f]]}")

# Calculate the correlation matrix for numeric columns only
numeric_columns = X.select_dtypes(include=['int64', 'float64'])
correlation_matrix = numeric_columns.corr()
print("Correlation Matrix:")
print(correlation_matrix)

numeric_columns_before_preprocessing = data.select_dtypes(include=['int64',
'float64'])
plt.figure(figsize=(20, 20))
sns.violinplot(data=numeric_columns_before_preprocessing)
plt.title('Before Preprocessing')
plt.xticks(rotation=45)


# Select only the numeric columns after outlier handling
X_train_numeric_outlier_handling = X_train.select_dtypes(include=['int64',
'float64'])
# Visualize data after outlier handling but not after one-hot encoding using a
violin plot
plt.figure(figsize=(20, 20))
sns.violinplot(data=X_train_numeric_outlier_handling)
plt.title('After Outlier Handling (Excluding One-Hot Encoding)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

3. Description

The Python code provided illustrates the implementation of Decision Tree models for Cobb angle prediction and curve progression classification using a dataset encompassing various clinical features relevant to spinal deformities. Before model development, the dataset undergoes extensive preprocessing steps, including handling missing values, outlier detection via a custom OutlierHandler class, and robust scaling using RobustScaler. Subsequently, the preprocessed data is split into training and testing sets for regression and classification tasks. Decision Tree models are trained on the preprocessed training data, and predictions are made on the testing set for evaluation. Various evaluation metrics such as mean squared error (MSE), R-squared, accuracy, precision, recall, and F1-score are computed to assess the performance of both regression and classification models. Additionally, a confusion matrix is generated to visually represent the classification model's predictions. Feature importances for the regression model and correlation matrix for numeric columns are also calculated and displayed. This code provides a comprehensive framework for utilizing Decision Tree models in predicting and evaluating spinal deformity parameters.

Appendix F: Random Forest

1. Introduction

This appendix presents Python code demonstrating the application of Random Forest models for predicting the Cobb angle and classifying curve progression in the context of spinal deformities. The code utilizes a dataset containing various clinical attributes related to spinal conditions. Random Forest models are employed for both regression and classification tasks, aiming to predict the Cobb angle as a continuous variable and classify curve progression into binary outcomes. The provided code showcases comprehensive preprocessing techniques, model training, evaluation metrics, and visualizations to assess the effectiveness of Random Forest models in predicting spinal deformity parameters.

2. Code Implementation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from  sklearn.metrics  import  mean_squared_error,  r2_score,  accuracy_score,
precision_score, recall_score, f1_score, confusion_matrix
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data  =  pd.read_csv("D:/Study  notes/Semester  2/ECE  710  IoT  and  wearble
devices/Project 3/projectdata.csv")

# Specify the columns to include
```

```
columns_to_include = ['Initial Age at initial X-ray time versus the birthdate', 
'Sex', 'Brace Treatment',
                     'Height (cm)', 'Max Scoliometer Standing for major curve',
                     'Inclinometer      (Kyphosis)(T1/T12)',      'Inclinometer 
(lordosis)(T12/S2)',
                     'Risser sign', 'Curve direction', 'Curve Number', 'Curve 
Length',
                     'Curve  Location',  'Curve  Classfication  from  TSC',  'AVR 
Measurement',
                     'No. of exercise sessions']

# Select only the columns to include
X = data[columns_to_include]
y_regression = data['Curve Length']
y_classification = data['Curve Length'].apply(lambda x: 1 if x >= 6 else 0)

# Split the data into train and test sets
X_train_reg,  X_test_reg,  y_train_reg,  y_test_reg  =  train_test_split(X, 
y_regression, test_size=0.5, random_state=42)
X_train_cls,  X_test_cls,  y_train_cls,  y_test_cls  =  train_test_split(X, 
y_classification, test_size=0.5, random_state=42)

# Define preprocessing steps for numerical and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy="constant")),
    ('scaler', MinMaxScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Outlier detection using IsolationForest
outlier_detector = IsolationForest(contamination=0.1)

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',     numeric_transformer,     X.select_dtypes(include=['int64', 
'float64']).columns),
        ('cat',                                         categorical_transformer, 
X.select_dtypes(include=['object']).columns)
    ])
```

```
# Preprocess the data for regression
X_train_preprocessed_reg = preprocessor.fit_transform(X_train_reg)
X_test_preprocessed_reg = preprocessor.transform(X_test_reg)

# Outlier detection and removal for regression
outliers_reg = outlier_detector.fit_predict(X_train_preprocessed_reg)
X_train_preprocessed_reg = X_train_preprocessed_reg[outliers_reg == 1]
y_train_reg = y_train_reg[outliers_reg == 1]

# Preprocess the data for classification
X_train_preprocessed_cls = preprocessor.fit_transform(X_train_cls)
X_test_preprocessed_cls = preprocessor.transform(X_test_cls)

# Outlier detection and removal for classification
outliers_cls = outlier_detector.fit_predict(X_train_preprocessed_cls)
X_train_preprocessed_cls = X_train_preprocessed_cls[outliers_cls == 1]
y_train_cls = y_train_cls[outliers_cls == 1]

# Define the models
model_reg = RandomForestRegressor(random_state=42)
model_cls = RandomForestClassifier(random_state=42)

# Train the models
model_reg.fit(X_train_preprocessed_reg, y_train_reg)
model_cls.fit(X_train_preprocessed_cls, y_train_cls)

# Make predictions on the test set for regression
y_pred_reg = model_reg.predict(X_test_preprocessed_reg)

# Make predictions on the test set for classification
y_pred_cls = model_cls.predict(X_test_preprocessed_cls)
y_pred_proba_cls  =  model_cls.predict_proba(X_test_preprocessed_cls)[:,  1]   #
Probability of positive class

# Evaluate the regression model
mse = mean_squared_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)
print("\nRegression Metrics (Final Curve Length Prediction):")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Evaluate the classification model
accuracy = accuracy_score(y_test_cls, y_pred_cls)
precision = precision_score(y_test_cls, y_pred_cls)
recall = recall_score(y_test_cls, y_pred_cls)
```

```
f1 = f1_score(y_test_cls, y_pred_cls)

print("\nClassification Metrics (Curve Progression Prediction):")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Confusion Matrix
cm = confusion_matrix(y_test_cls, y_pred_cls)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Number of columns in preprocessed data for regression
num_columns_reg = X_train_preprocessed_reg.shape[1]
print(f"Number   of   columns   in   preprocessed   data   for   regression:
{num_columns_reg}")

# Number of columns in preprocessed data for classification
num_columns_cls = X_train_preprocessed_cls.shape[1]
print(f"Number   of   columns   in   preprocessed   data   for   classification:
{num_columns_cls}")

# Display preprocessed data for regression
print("\nPreprocessed Data for Regression:")
print(pd.DataFrame(X_train_preprocessed_reg))

# Display preprocessed data for classification
print("\nPreprocessed Data for Classification:")
print(pd.DataFrame(X_train_preprocessed_cls))

# Number of columns in preprocessed data for regression
num_columns_reg = X_train_preprocessed_reg.shape[1]
print(f"Number   of   columns   in   preprocessed   data   for   regression:
{num_columns_reg}")

# Number of columns in preprocessed data for classification
num_columns_cls = X_train_preprocessed_cls.shape[1]
print(f"Number   of   columns   in   preprocessed   data   for   classification:
{num_columns_cls}")
```

```
# Display preprocessed data for regression
print("\nPreprocessed Data for Regression:")
print(pd.DataFrame(X_train_preprocessed_reg))

# Display preprocessed data for classification
print("\nPreprocessed Data for Classification:")
print(pd.DataFrame(X_train_preprocessed_cls))


# Feature Importance Plots for Regression
plt.figure(figsize=(10, 6))
num_features = min(5,len(X.columns), len(model_reg.feature_importances_))
plt.bar(range(num_features), model_cls.feature_importances_[:num_features])
plt.xticks(range(num_features), X.columns[:num_features], rotation=90)
plt.title('Feature Importance for Regression')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()

# Feature Importance Plots for Classification
plt.figure(figsize=(10, 6))
num_features = min(len(X.columns), len(model_cls.feature_importances_))
plt.bar(range(num_features), model_cls.feature_importances_[:num_features])
plt.xticks(range(num_features), X.columns[:num_features], rotation=90)
plt.title('Feature Importance for Classification')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()
```

3. Description

The Python code provided illustrates the implementation of Random Forest models for Cobb angle prediction and curve progression classification using a dataset containing diverse clinical features relevant to spinal deformities. Before model development, the dataset undergoes preprocessing steps, including handling missing values, scaling numerical features using MinMaxScaler, and one-hot encoding categorical features. Outlier detection using Isolation Forest is performed to remove outliers from the dataset. The preprocessed data is then split into training and testing sets for regression and classification tasks. Random Forest models are trained on the preprocessed training data, and predictions are made on the testing set for evaluation. Various evaluation metrics such as mean squared error (MSE), R-squared, accuracy, precision, recall, and F1-score are computed to assess the performance of both regression and classification models. Additionally, confusion matrices are generated to visually represent the classification model's predictions. Feature importance plots are also provided to highlight the significance of different features in both regression and classification tasks. This code serves as a comprehensive guide for utilizing Random Forest models in predicting and evaluating spinal deformity parameters.

Appendix G: Artificial Neural Network

1. Introduction

This appendix presents Python code demonstrating the utilization of Artificial Neural Networks (ANNs) for predicting curve progression and curve length in individuals with spinal deformities. The code employs a dataset containing diverse clinical attributes related to spinal conditions. ANNs are employed for both regression and classification tasks, aiming to predict the progression of spinal deformities and the length of the spinal curve. The provided code showcases comprehensive preprocessing techniques, model training, evaluation metrics, and visualization to assess the effectiveness of ANNs in predicting spinal deformity parameters.

2. Code Implementation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import IsolationForest
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, roc_curve, roc_auc_score, mean_squared_error,
r2_score

# Load the data
data = pd.read_csv("D:/Study notes/Semester 2/ECE 710 IoT and wearble
devices/Project 3/projectdata.csv")

# Specify the columns to include
columns_to_include = ['Sex', 'Brace Treatment',
                     'Height (cm)', 'Max Scoliometer Standing for major curve',
                     'Inclinometer     (Kyphosis)(T1/T12)',     'Inclinometer
(lordosis)(T12/S2)',
                     'Risser sign', 'Curve direction', 'Curve Number', 'Curve
Length',
                     'Curve  Location',  'Curve  Classfication  from  TSC',  'AVR
Measurement',
                     'No. of exercise sessions']

# Select only the columns to include
X = data[columns_to_include]
y_progression_cls = data['Curve Length']  # Classification target
y_progression_reg = data['Curve Length']  # Regression target

# Convert to binary classification
y_progression_binary = y_progression_cls.apply(lambda x: 1 if x >= 6 else 0)  #
Example threshold (adjust as needed)

# Split the data into train and test sets for both tasks
X_train_cls,   X_test_cls,   y_train_cls,   y_test_cls   =   train_test_split(X,
y_progression_binary, test_size=0.1, random_state=42)
X_train_reg,   X_test_reg,   y_train_reg,   y_test_reg   =   train_test_split(X,
y_progression_reg, test_size=0.1, random_state=42)

# Generate violin plots before preprocessing
plt.figure(figsize=(12, 6))
sns.violinplot(data=X, palette="viridis", linewidth=2)
plt.title('Data Distribution Before Preprocessing')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Define preprocessing steps for numerical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler())
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',     numeric_transformer,     X.select_dtypes(include=['int64',
'float64']).columns),
    ])
```

```
# Outlier detection using IsolationForest
outlier_detector = IsolationForest(contamination=0.1)

# Preprocess the data for classification
X_train_preprocessed_cls = preprocessor.fit_transform(X_train_cls)
X_test_preprocessed_cls = preprocessor.transform(X_test_cls)

# Remove outliers from training data for classification
outliers_cls = outlier_detector.fit_predict(X_train_preprocessed_cls)
X_train_preprocessed_cls = X_train_preprocessed_cls[outliers_cls == 1]
y_train_cls = y_train_cls[outliers_cls == 1]

# Preprocess the data for regression
X_train_preprocessed_reg = preprocessor.fit_transform(X_train_reg)
X_test_preprocessed_reg = preprocessor.transform(X_test_reg)

# Remove outliers from training data for regression
outliers_reg = outlier_detector.fit_predict(X_train_preprocessed_reg)
X_train_preprocessed_reg = X_train_preprocessed_reg[outliers_reg == 1]
y_train_reg = y_train_reg[outliers_reg == 1]

# Generate violin plots after preprocessing
plt.figure(figsize=(12, 6))
sns.violinplot(data=pd.DataFrame(X_train_preprocessed_reg,
columns=X.select_dtypes(include=['int64',                'float64']).columns),
palette="viridis", linewidth=2)
plt.title('Data Distribution After Preprocessing (Regression)')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))
sns.violinplot(data=pd.DataFrame(X_train_preprocessed_cls,
columns=X.select_dtypes(include=['int64',                'float64']).columns),
palette="viridis", linewidth=2)
plt.title('Data Distribution After Preprocessing (Classification)')
plt.xlabel('Features')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
# Define the ANN model for classification
model_cls = Sequential()
model_cls.add(Dense(X_train_preprocessed_cls.shape[1] * 2, activation='relu', 
input_shape=(X_train_preprocessed_cls.shape[1],)))
model_cls.add(Dense(X_train_preprocessed_cls.shape[1], activation='relu'))
model_cls.add(Dense(1, activation='sigmoid'))  # Binary classification requires 
a sigmoid activation

# Compile the model with custom learning rate for classification
opt_cls = Adam(learning_rate=0.001)  # Specify learning_rate instead of lr
model_cls.compile(optimizer=opt_cls,               loss='binary_crossentropy', 
metrics=['accuracy'])

# Train the model for classification
model_cls.fit(X_train_preprocessed_cls,        y_train_cls,        epochs=100, 
batch_size=32, verbose=1)

# Make predictions on the test set for classification
y_pred_proba_cls = model_cls.predict(X_test_preprocessed_cls)
y_pred_cls = (y_pred_proba_cls > 0.5).astype(int)  # Convert probabilities to 
binary predictions

# Evaluate the model for classification
accuracy_cls = accuracy_score(y_test_cls, y_pred_cls)
precision_cls = precision_score(y_test_cls, y_pred_cls)
recall_cls = recall_score(y_test_cls, y_pred_cls)
f1_cls = f1_score(y_test_cls, y_pred_cls)

print("\nClassification Metrics (Curve Progression Prediction):")
print(f"Accuracy: {accuracy_cls:.2f}")
print(f"Precision: {precision_cls:.2f}")
print(f"Recall: {recall_cls:.2f}")
print(f"F1-score: {f1_cls:.2f}")

# Confusion Matrix for classification
cm_cls = confusion_matrix(y_test_cls, y_pred_cls)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_cls, annot=True, cmap="Blues", fmt="d")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix (Classification)")
plt.show()

# Define the ANN model for regression
model_reg = Sequential()
```

```
model_reg.add(Dense(X_train_preprocessed_reg.shape[1] * 2, activation='relu',
input_shape=(X_train_preprocessed_reg.shape[1],)))
model_reg.add(Dense(X_train_preprocessed_reg.shape[1], activation='relu'))
model_reg.add(Dense(1))  # No activation for regression

# Compile the model with custom learning rate for regression
opt_reg = Adam(learning_rate=0.001)  # Specify learning_rate instead of lr
model_reg.compile(optimizer=opt_reg, loss='mean_squared_error')  # Using mean
squared error for regression

# Train the model for regression
model_reg.fit(X_train_preprocessed_reg,        y_train_reg,        epochs=100,
batch_size=32, verbose=1)

# Make predictions on the test set for regression
y_pred_reg = model_reg.predict(X_test_preprocessed_reg)

# Evaluate the model for regression
mse_reg = mean_squared_error(y_test_reg, y_pred_reg)
r2_reg = r2_score(y_test_reg, y_pred_reg)

print("\nRegression Metrics (Curve Length Prediction):")
print(f"Mean Squared Error: {mse_reg:.2f}")
print(f"R-squared: {r2_reg:.2f}")

# Print entire dataset
print("\nEntire Dataset:")
print(data)

# Check for overfitting or underfitting for classification
# Evaluate the model on training data for classification
train_loss_cls,                      train_accuracy_cls                     =
model_cls.evaluate(X_train_preprocessed_cls, y_train_cls, verbose=0)
print(f"\nTraining Loss (Classification): {train_loss_cls:.4f}")
print(f"Training Accuracy (Classification): {train_accuracy_cls:.4f}")

# Evaluate the model on test data for classification
test_loss_cls, test_accuracy_cls = model_cls.evaluate(X_test_preprocessed_cls,
y_test_cls, verbose=0)
print(f"\nTest Loss (Classification): {test_loss_cls:.4f}")
print(f"Test Accuracy (Classification): {test_accuracy_cls:.4f}")

# Check for overfitting or underfitting for regression
# Evaluate the model on training data for regression
```

```
train_loss_reg                =                mean_squared_error(y_train_reg,
model_reg.predict(X_train_preprocessed_reg))
print(f"\nTraining Loss (Regression): {train_loss_reg:.4f}")

# Evaluate the model on test data for regression
test_loss_reg                 =                 mean_squared_error(y_test_reg,
model_reg.predict(X_test_preprocessed_reg))
print(f"\nTest Loss (Regression): {test_loss_reg:.4f}")

# Print final preprocessed data for classification
print("\nFinal Preprocessed Data for Classification:")
print(pd.DataFrame(X_train_preprocessed_cls))

# Print final preprocessed data for regression
print("\nFinal Preprocessed Data for Regression:")
print(pd.DataFrame(X_train_preprocessed_reg))

# Print classification target
print("\nClassification Target:")
print(y_train_cls)

# Print regression target
print("\nRegression Target:")
print(y_train_reg)
```

3. Description

The Python code provided illustrates the implementation of Artificial Neural Networks (ANNs) for predicting curve progression and curve length in individuals with spinal deformities. The dataset consists of various clinical features relevant to spinal conditions. The code begins by splitting the dataset into features and target variables for both regression and classification tasks. Next, the data undergoes preprocessing steps, including handling missing values, scaling numerical features using MinMaxScaler, and outlier detection using Isolation Forest. The preprocessed data is then split into training and testing sets for regression and classification tasks.

For classification, an ANN model is defined with multiple layers and compiled with custom learning rates. The model is trained using the preprocessed training data and evaluated using various metrics such as accuracy, precision, recall, and F1-score. Confusion matrices are generated to visualize the classification model's predictions. For regression, a similar approach is followed, with the ANN model defined and compiled for regression tasks. The model is trained, and evaluation metrics such as mean squared error and R-squared are computed to assess its performance.

Additionally, the code checks for overfitting or underfitting by evaluating the model on both training and test data. The final preprocessed data and target variables are printed for both regression and classification tasks, providing insights into the data transformation process. This code serves as a comprehensive guide for utilizing Artificial Neural Networks in predicting and evaluating spinal deformity parameters.

Appendix H: Dimensionality Reduction

1. Introduction

This appendix showcases Python code employing dimensionality reduction techniques on a dataset related to spinal conditions. By utilizing K-Means clustering and Truncated Singular Value Decomposition (TruncatedSVD), the code aims to identify underlying patterns and reduce the dimensionality of the dataset for further analysis. The reduced-dimensional data is visualized using scatter plots, and the optimal number of clusters is determined based on the silhouette score.

2. Code Implementation

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA  # Import PCA for dimensionality reduction
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import TruncatedSVD
from sklearn.linear_model import Ridge, Lasso

# Load the data
data  =  pd.read_csv("D:/Study  notes/Semester  2/ECE  710  IoT  and  wearble
devices/Project 3/projectdata.csv")

# Split the data into features and target variables
columns_to_include = ['Initial Age at initial X-ray time versus the birthdate',
'Sex', 'Brace Treatment', 'Height (cm)', 'Max Scoliometer Standing for major
curve', 'Inclinometer (Kyphosis)(T1/T12)', 'Inclinometer (lordosis)(T12/S2)',
'Risser sign', 'Curve direction', 'Curve Number', 'Curve Length', 'Curve
Location', 'Curve Classfication from TSC', 'AVR Measurement', 'No. of exercise
sessions']

# Select only the columns to include
X = data[columns_to_include]

# Preprocessing steps for numerical and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',     numeric_transformer,     X.select_dtypes(include=['int64',
'float64']).columns),
        ('cat',                                          categorical_transformer,
X.select_dtypes(include=['object']).columns)
    ])

# Preprocess the data
X_processed = preprocessor.fit_transform(X)

# Reduce dimensionality of data using TruncatedSVD
svd = TruncatedSVD(n_components=2)
X_svd = svd.fit_transform(X_processed)

# Find optimal number of clusters using silhouette score
silhouette_scores = []
for n_clusters in range(2, 11):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X_svd)
    silhouette_avg = silhouette_score(X_svd, cluster_labels)
    silhouette_scores.append(silhouette_avg)

optimal_n_clusters = silhouette_scores.index(max(silhouette_scores)) + 2  # Add
2 because range starts from 2
```

```
# Perform K-Means clustering with optimal number of clusters
kmeans = KMeans(n_clusters=optimal_n_clusters, random_state=42)
cluster_labels = kmeans.fit_predict(X_svd)

# Scatter plot of the reduced data points
plt.scatter(X_svd[:,  0],  X_svd[:,  1],  c=cluster_labels,  cmap='viridis',
alpha=0.5)
plt.title('Scatter Plot of Reduced Data Points')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Print silhouette score
print(f"Optimal number of clusters: {optimal_n_clusters}")
print(f"Silhouette score: {max(silhouette_scores)}")
```

3. Description

The provided Python code demonstrates the application of dimensionality reduction techniques on a dataset concerning spinal conditions. Initially, the dataset is loaded, and specific columns relevant to the analysis are selected. The preprocessing steps involve handling missing values and encoding categorical features.

Next, the data undergoes dimensionality reduction using Truncated SVD, which reduces the number of features while preserving essential information. The reduced-dimensional data is then subjected to K-Means clustering to identify distinct clusters within the dataset. The silhouette score is computed for different numbers of clusters to determine the optimal number of clusters that best capture the inherent structure of the data.

Finally, the reduced-dimensional data points are visualized using a scatter plot, where each point is colored according to its assigned cluster. This visualization helps in understanding the underlying patterns and relationships within the dataset. Additionally, the optimal number of clusters and the corresponding silhouette score are printed to provide insights into the clustering performance.

Appendix I: K Means Clustering

1. Introduction

This appendix presents the Python code for performing K-Means clustering analysis on the dataset. K-Means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a set of clusters. The code demonstrates how to preprocess the data, determine the optimal number of clusters, perform clustering, and visualize the results.

2. Code Implementation

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Load the data
data  =  pd.read_csv("D:/Study  notes/Semester  2/ECE  710  IoT  and  wearble
devices/Project 3/projectdata.csv")

# Split the data into features and target variables
columns_to_include = ['Initial Age at initial X-ray time versus the birthdate',
'Sex', 'Brace Treatment', 'Height (cm)', 'Max Scoliometer Standing for major
curve', 'Inclinometer (Kyphosis)(T1/T12)', 'Inclinometer (lordosis)(T12/S2)',
'Risser  sign',  'Curve  direction',  'Curve  Number',  'Curve  Length',  'Curve
Location', 'Curve Classfication from TSC', 'AVR Measurement', 'No. of exercise
sessions']

# Select only the columns to include
X = data[columns_to_include]

# Preprocessing steps for numerical and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num',     numeric_transformer,     X.select_dtypes(include=['int64',
'float64']).columns),
        ('cat',                                         categorical_transformer,
X.select_dtypes(include=['object']).columns)
    ])

# Preprocess the data
X_processed = preprocessor.fit_transform(X)

# Find optimal number of clusters using silhouette score
silhouette_scores = []
for n_clusters in range(2, 11):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X_processed)
    silhouette_avg = silhouette_score(X_processed, cluster_labels)
    silhouette_scores.append(silhouette_avg)

optimal_n_clusters = silhouette_scores.index(max(silhouette_scores)) + 2  # Add
2 because range starts from 2

# Perform K-Means clustering with optimal number of clusters
kmeans = KMeans(n_clusters=optimal_n_clusters, random_state=42)
cluster_labels = kmeans.fit_predict(X_processed)

# Calculate silhouette score and inertia for the optimal number of clusters
silhouette_avg = silhouette_score(X_processed, cluster_labels)
inertia = kmeans.inertia_

print(f"Optimal number of clusters: {optimal_n_clusters}")
print(f"Silhouette score: {silhouette_avg}")
print(f"Inertia: {inertia}")

# Verify the shape of X_processed
print("Shape of X_processed:", X_processed.shape)


# Reduce dimensionality of data using PCA
pca = PCA(n_components=2, svd_solver='arpack')
X_pca = pca.fit_transform(X_processed)
```

```
# Plot scatter plot of the clusters
plt.figure(figsize=(10, 7))
plt.scatter(X_pca[:,   0],   X_pca[:,   1],   c=cluster_labels,   cmap='viridis',
alpha=0.5)
plt.scatter(kmeans.cluster_centers_[:,   0],   kmeans.cluster_centers_[:,   1],
marker='x', s=100, c='red', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

3. Description

This code implements K-Means clustering on a dataset to identify clusters within the data. The dataset is preprocessed to handle missing values and categorical features. The optimal number of clusters is determined using the silhouette score, and K-Means clustering is performed accordingly. Finally, the clusters are visualized using principal component analysis (PCA).

Appendix J: K-Nearest Neighbors

1. Introduction

This appendix showcases Python code utilizing K-Nearest Neighbors (KNN) regression and classification algorithms on a dataset related to spinal conditions. The code aims to predict the Cobb angle and classify curve progression based on various features extracted from medical records. Additionally, it evaluates the performance of both regression and classification models using appropriate metrics and visualizations.

2. Code Implementation

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score,
precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data = pd.read_csv("D:/Study notes/Semester 2/ECE 710 IoT and wearble
devices/Project 3/projectdata.csv")

# Split the data into features and target variables
columns_to_include = ['Initial Age at initial X-ray time versus the birthdate',
'Sex', 'Brace Treatment', 'Height (cm)', 'Max Scoliometer Standing for major
curve', 'Inclinometer (Kyphosis)(T1/T12)', 'Inclinometer (lordosis)(T12/S2)',
'Risser sign', 'Curve direction', 'Curve Number', 'Curve Length', 'Curve
Location', 'Curve Classfication from TSC', 'AVR Measurement', 'No. of exercise
sessions']

# Select only the columns to include
X = data[columns_to_include]
y_cobb = data['Curve Length']
y_progression = data['Curve Classfication from TSC']

# Split the data into train and test sets
X_train, X_test, y_cobb_train, y_cobb_test = train_test_split(X, y_cobb,
test_size=0.1, random_state=42)
X_train, X_test, y_progression_train, y_progression_test = train_test_split(X,
y_progression, test_size=0.1, random_state=42)

# Define preprocessing steps for numerical and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps for all features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, X_train.select_dtypes(include=['int64',
'float64']).columns),
        ('cat',                                         categorical_transformer,
X_train.select_dtypes(include=['object']).columns)
    ])

# Create pipelines for regression and classification
regression_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', KNeighborsRegressor())
])

classification_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier())
])

# Train the regression model
regression_pipeline.fit(X_train, y_cobb_train)

# Train the classification model
classification_pipeline.fit(X_train, y_progression_train)

# Make predictions on the test set
y_cobb_pred = regression_pipeline.predict(X_test)
y_progression_pred = classification_pipeline.predict(X_test)

# Evaluate the regression model's performance
mse = mean_squared_error(y_cobb_test, y_cobb_pred)
r2 = r2_score(y_cobb_test, y_cobb_pred)
print(f"Regression Metrics (Final Cobb Angle Prediction):")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Evaluate the classification model's performance
accuracy = accuracy_score(y_progression_test, y_progression_pred)
precision     =     precision_score(y_progression_test,     y_progression_pred,
average='weighted')
recall = recall_score(y_progression_test, y_progression_pred, average='micro')
f1 = f1_score(y_progression_test, y_progression_pred, average='micro')
print("\nClassification Metrics (Curve Progression Prediction):")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Generate confusion matrix for the classification model
```

```
cm = confusion_matrix(y_progression_test, y_progression_pred)

# Plot confusion matrix using seaborn
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calculate the correlation matrix for numeric columns only
numeric_columns = X.select_dtypes(include=['int64', 'float64'])
correlation_matrix = numeric_columns.corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

3. Description

The provided Python code demonstrates the application of KNN regression and classification algorithms on a dataset concerning spinal conditions. After loading the dataset, specific columns relevant to the analysis, such as age, sex, treatment details, and various measurements, are selected.

The dataset is split into training and testing sets for both regression and classification tasks using the **train_test_split** function from the **sklearn.model_selection** module. Preprocessing steps include handling missing values and encoding categorical features using pipelines defined with **ColumnTransformer**.

Two separate pipelines are created for regression and classification tasks. The regression pipeline utilizes the **KNeighborsRegressor**, while the classification pipeline employs the **KNeighborsClassifier**.

The regression model is trained on the Cobb angle target variable, while the classification model is trained on the curve progression classification variable. Predictions are made on the test set for both tasks.

To evaluate the regression model's performance, metrics such as mean squared error (MSE) and R-squared are calculated and printed. For the classification model, metrics including accuracy, precision, recall, and F1-score are computed and displayed.

Furthermore, a confusion matrix is generated using **confusion_matrix** from the **sklearn.metrics** module to visualize the classification model's performance.

Lastly, the correlation matrix for numeric columns is computed to analyze the relationships between different features in the dataset. The correlation matrix provides insights into how various features correlate with each other and can aid in feature selection or understanding the dataset's structure.

References:

[1] D. Vyas, A. Ganesan and P. Meel, "Computation and Prediction Of Cobb's Angle Using Machine Learning Models," 2022 2nd International Conference on Intelligent Technologies (CONIT), Hubli, India, 2022, pp. 1-6, doi: 10.1109/CONIT55038.2022.9847938. keywords: {Image segmentation;Computational modeling;Spine;Surgery;X-rays;Machine learning;Predictive models;Cobb's Angle;Scoliosis;U-Net Model;X-Ray;Vertebrate Segmentation},

[2] N. A. Makhdoomi et al., "Development of Scoliotic Spine Severity Detection using Deep Learning Algorithms," 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2022, pp. 0574-0579, doi: 10.1109/CCWC54503.2022.9720906. keywords: {Measurement;Technological innovation;Pediatrics;Orthopedic surgery;Conferences;Neural networks;Prediction algorithms;Scoliosis;cobbs' angle;deep learning;CNN;U-Net},

[3] "One-Hot Encoding," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/one-hot-encoding-in-nlp/. [Accessed: Apr. 5, 2024].

[4] "ML | Handle Missing Data with Simple Imputer," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/ml-handle-missing-data-with-simple-imputer/. [Accessed: Apr. 5, 2024].

[5] D. C. Stitzel, "Scoliosis Treatment Timeline: Past, Present & Future," Treatingscoliosis, Aug. 15, 2019. [Online]. Available: https://treatingscoliosis.com/blog/scoliosis-treatment-history/#:~:text=Earliest%20known%20record%20of%20scoliosis,Galen%20in%20approximately%20200%20A.D.. [Accessed: Apr. 5, 2024].