QUESTIONS RELATED TO TASK 1 (HEART DISEASE PREDICTION)

Ques 1. Which Python libraries were most helpful in loading and exploring the dataset?

Ans.

Pandas (pd):

- Pandas are extremely helpful for loading, cleaning, and exploring datasets. It provides robust data structures like DataFrames and Series, making it easy to load and manipulate tabular data.
- Functions like read_csv(), read_excel(), and read_sql() in Pandas are used to load data from various sources.

NumPy (np):

- NumPy is essential for numerical and mathematical operations in Python. It provides a multi-dimensional array (ndarray) that is efficient for handling large datasets.
- While not primarily used for loading data, NumPy is often used alongside Pandas for numerical operations and data manipulation.

Matplotlib (plt):

- Matplotlib is a powerful library for creating static visualizations. It is particularly helpful for generating various types of plots and charts to explore data distributions and patterns.
- It's commonly used for creating bar plots, line plots, scatter plots, and more.

Seaborn (sns):

- Seaborn is a high-level visualization library that simplifies the creation of aesthetically pleasing statistical graphics.
- It complements Matplotlib and is especially useful for quickly generating complex visualizations such as heatmaps, pair plots, and violin plots.

Ques 2. What preprocessing steps did you find necessary to apply to the heart dataset?
Ans.

1. **‘data.shape**’: This line of code returns the shape of the DataFrame ‘data’, which consists of the number of rows and columns in the dataset. It helps you understand the dimensions of your dataset.

2. **‘data.info()’**: This line of code provides information about the DataFrame, including the data types of each column and the number of non-null values. It's useful for identifying missing values and understanding the data types of your features.

3. **data.isna().sum()**: This line of code calculates the number of missing values (NaN or None) in each column of the DataFrame. By summing the results, you get the count of missing values for each column.

Ques 3. What metrics were used to evaluate the Classification problems and why?
Ans.

Here are some commonly used metrics for evaluating classification problems:

1. **Accuracy**: Accuracy is a common metric that measures the ratio of correctly predicted instances to the total number of instances in the dataset. It's suitable for balanced datasets where classes are roughly equal in size. However, it can be misleading in imbalanced datasets.

2. **Precision**: Precision is the ratio of true positive predictions to the total number of positive predictions (true positives + false positives). It's useful when minimizing false positives is important (e.g., in medical diagnoses).

3. **Recall (Sensitivity or True Positive Rate)**: Recall measures the ratio of true positive predictions to the total number of actual positive instances (true positives + false negatives). It's important when minimizing false negatives is crucial (e.g., in fraud detection).

4. **F1-Score**: The F1-Score is the harmonic mean of precision and recall. It balances both precision and recall, making it suitable for imbalanced datasets.

5. **ROC-AUC**: Receiver Operating Characteristic Area Under the Curve (ROC-AUC) measures the ability of a model to distinguish between positive and negative classes across different probability thresholds. It's particularly useful when dealing with imbalanced datasets or when you want to assess the overall performance of a classifier.

6. **Confusion Matrix**: A confusion matrix provides a detailed breakdown of true positives, true negatives, false positives, and false negatives. It's a valuable tool for understanding the specific types of errors made by a classification model.

Ques 4. How did you detect overfitting in the model and what strategies did you use to mitigate it?
Ans.

To detect overfitting in the models and mitigate it, the following strategies are employed in the code:

1. **Cross-Validation**: K-fold cross-validation is used to evaluate the models. This technique helps in assessing model performance across multiple subsets of the data, which can reveal

whether a model is overfitting. It does so by providing performance scores for both training and testing subsets for each fold.

2. **Comparing Training and Testing Accuracy Scores**:
    - The code calculates and prints both training and testing accuracy scores for each fold and each model. This comparison is a key indicator of potential overfitting.
    - Overfitting is detected when the training accuracy is significantly higher than the testing accuracy. This suggests that the model is fitting the training data very well but does not generalize well to unseen data.

3. **Monitoring Average Accuracy Scores**:
    - The code calculates and prints the average training and testing accuracy scores across all folds for each model.
    - A significant drop in testing accuracy compared to training accuracy indicates overfitting. Lower testing accuracy suggests that the model is not performing well on new data.

4. **Model Complexity:**
    - The code uses different machine learning algorithms with varying degrees of complexity (K-Nearest Neighbors, Naive Bayes, and Decision Tree).
    - One strategy to mitigate overfitting is to use simpler models. Simplicity can be achieved by choosing models with fewer parameters, such as Naive Bayes, or by controlling the complexity of models, such as decision trees, through hyperparameters like maximum depth.

5. **Regularization (Not Explicitly Shown):**
    - The code doesn't explicitly implement regularization techniques, but regularization can be a strategy to mitigate overfitting in some models, such as linear models or neural networks.
    - Techniques like L1 and L2 regularization can be applied to penalize large parameter values and encourage simpler models.

Ques 5. How did you choose the number of neighbors for the KNN algorithm? Explain why?

Ans. Here's how the choice of the number of neighbors is made and the rationale behind it:

1. **Initial Calculation of k:**
    - An initial estimate for the value of k is calculated as the square root of the number of rows in the training data (X_train).
    - In this case, k = sqrt(len(X_train)) is calculated, but the result is not directly used.

2. **Manual Selection of K:**
    - Despite the initial calculation, a specific value of k (k = 29) is chosen manually for the KNN algorithm.
    - This choice appears to be based on prior knowledge or experimentation.

3. **Testing Multiple K Values:**
    - A range of k values from 1 to 29 is tested by training and evaluating KNN models with different values of k.
    - The accuracy of each model is calculated on a test dataset (X_test and Y_test), and the results are stored in the scores list.
4. **Plotting Accuracy vs. k:**
    - The code then plots the accuracy against the values of k using Matplotlib.

5. **Average Accuracy Calculation:**
    - An average accuracy value is calculated from a provided list of accuracy values for k ranging from 1 to 29. *
    - The list of provided accuracy values is used to calculate the average. Selection of Best k: The code chooses a specific value of k (k = 5) as the "best" k value based on some criteria.