

Heart Disease Prediction Using Machine Learning

Introduction

This project focuses on the development of a machine learning application that predicts heart disease in individuals based on a variety of health indicators. Utilizing the K-Nearest Neighbors (KNN) algorithm, this application aims to assist healthcare professionals and individuals in assessing the risk of heart disease efficiently and effectively.

Data Description

The dataset used in this project comprises cardiovascular measurements from 303 patients.

Key features include:

Age, sex, types of chest pain, resting blood pressure, cholesterol levels, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, ST depression induced by exercise, the slope of the peak exercise ST segment, number of major vessels colored by fluoroscopy, and thalassemia.

The target variable indicates the presence or absence of heart disease.

Methodology

I. Data Loading and Preprocessing

Dataset found on Kaggle is loaded to a public GitHub repository and then from there loaded into a Pandas dataframe. It is then preprocessed to handle missing values and standardize features. From the application perspective, Streamlit's caching mechanism is employed to optimize data loading operations within the application.

LOADING THE DATASET FROM GITHUB

+ Code

+ Text

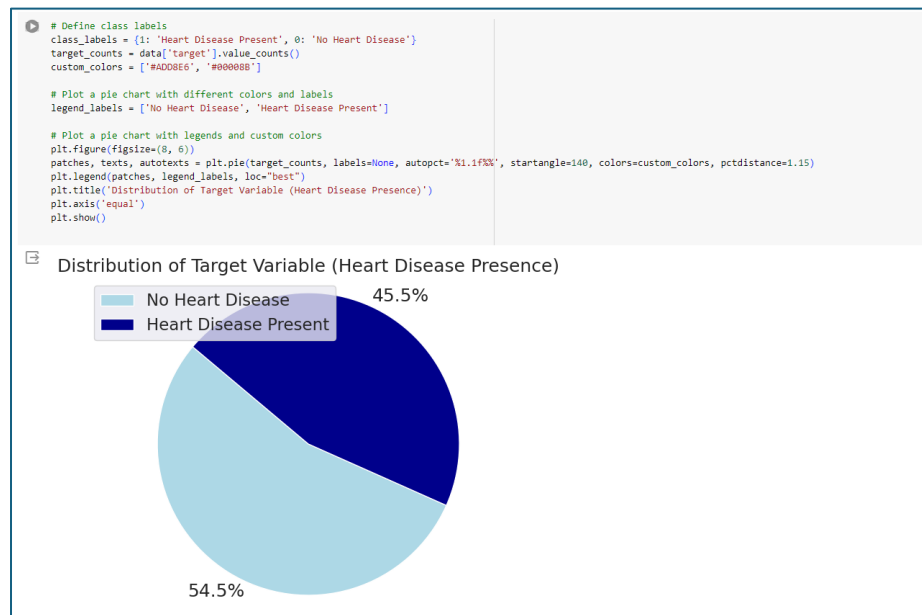
```
# Load and prepare the data
def load_data():
    # URL for the Kaggle heart disease dataset with the appropriate column names
    url = "https://raw.githubusercontent.com/vaishveerkumar/Data-Science/main/CAPSTONE_PROJECT/heart.csv"
    data = pd.read_csv(url)
    data = data.replace('?', np.nan) # Replace missing value placeholders
    data.dropna(inplace=True) # Drop rows with missing values
    data['target'] = (data['target'] > 0).astype(int)

    return data

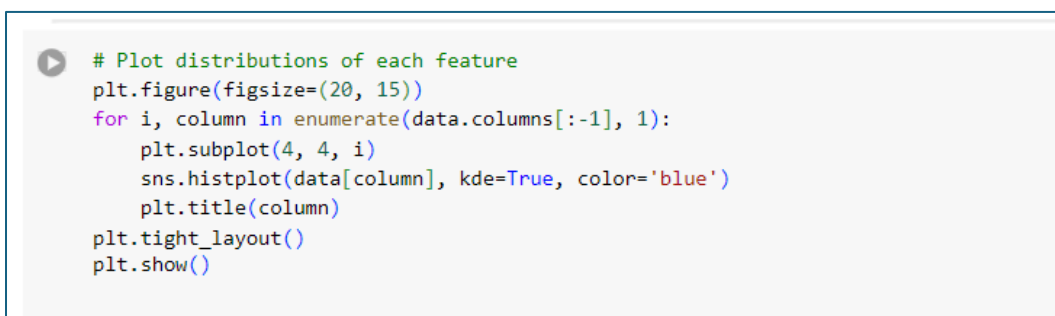
# Load the data
data = load_data()
```

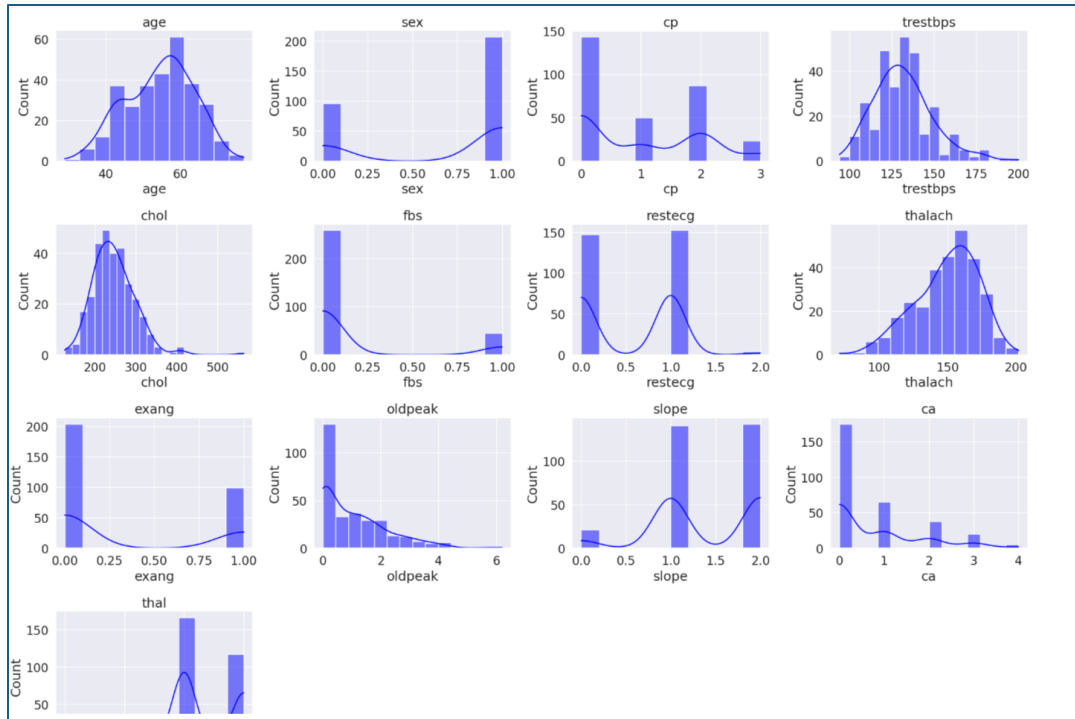
II. Exploratory Data Analysis (EDA)

In the exploratory data analysis (EDA) phase of the project, I meticulously examined the heart disease dataset to understand the distributions and relationships of the various physiological and clinical features. Initial analysis involved generating descriptive statistics to capture trends and outliers in the dataset, including measures like mean, median, and standard deviation for continuous variables such as age, cholesterol levels, and resting blood pressure.



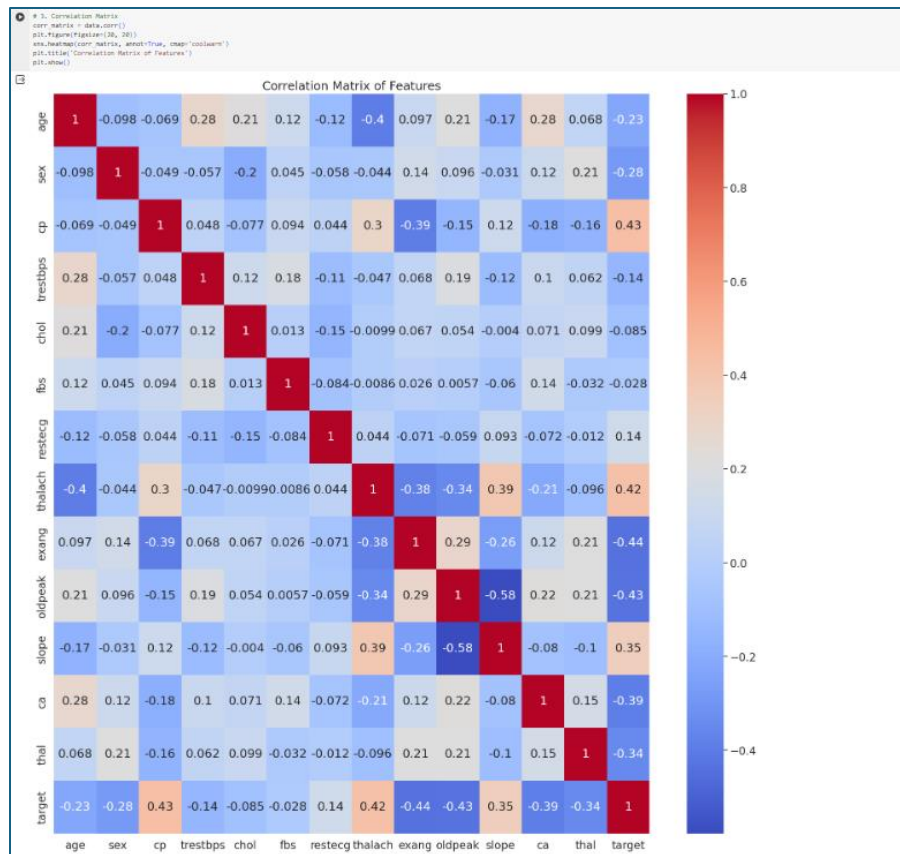
Histograms were employed to visualize the distribution of these features, revealing skewness and the presence of outliers that could impact model performance.





Correlation matrices were generated to investigate the relationships between features and the target variable, highlighting key predictors of heart disease such as the type of chest pain, maximum heart rate achieved, and ST depression induced by exercise.

```
# 3. Correlation Matrix
corr_matrix = data.corr()
plt.figure(figsize=(20, 20))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Features')
plt.show()
```



This analysis not only facilitated a deeper understanding of the data but also guided the feature selection process, ensuring that the most impactful variables were included in the model training phase. The insights from the EDA were instrumental in refining the preprocessing steps, such as scaling and encoding, to optimize the data for the machine learning models employed in the project.

****Detailed explanation/inferences from the correlation matrix and feature distributions has been included in the .ipynb notebook****

III. Model Selection and Training

The project evaluated two models: K-Nearest Neighbors (KNN) and Random Forest. Both models were applied to the dataset, with the following outcomes:

- a. **K-Nearest Neighbors (KNN):** Achieved a high accuracy of approximately 91.80% on the test data, demonstrating effective generalization from the training data. The model was configured with varying numbers of neighbors to optimize performance, ultimately selecting the best configuration based on validation accuracy.

```
[ ] knn = KNeighborsClassifier()

# Fit and score the model
def fit_and_score(knn, X_train, X_test, y_train, y_test):
    # Fit the model to data
    knn.fit(X_train, y_train)
    # Evaluate the model
    score = knn.score(X_test, y_test)
    return score
```

```
[ ] model_score = fit_and_score(knn,
                                X_train,
                                X_test,
                                y_train,
                                y_test)

print("Model Score:", model_score)
```

Model Score: 0.9016393442622951

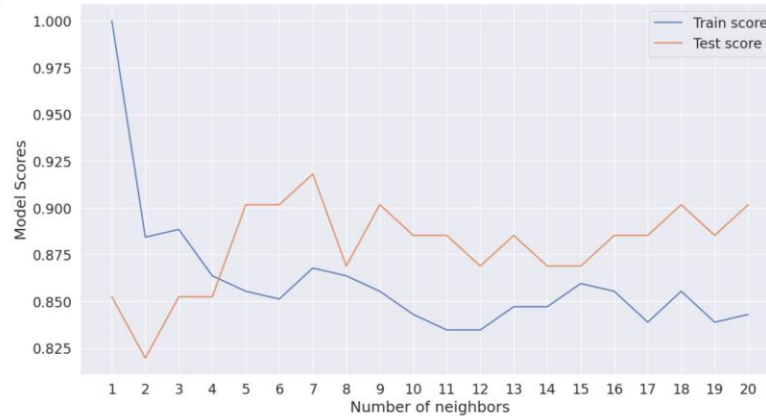
The model achieves an accuracy score of approximately 90.16% on the test data. This means that the model correctly predicts the target variable (whether a patient has heart disease or not) for about 90.16% of the samples in the test dataset.

VISUALIZING THE TEST AND TRAIN SCORES FROM THE KNN MODEL

```
plt.figure(figsize=(15, 8))
plt.plot(neighbors, train_scores, label="Train score")
plt.plot(neighbors, test_scores, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model Scores")
plt.legend()

print(f"Maximum KNN score on the test data: {max(test_scores)*100:.2f}%")
```

Maximum KNN score on the test data: 91.88%



From the graph the following can be inferred:

- At a low number of neighbors (from 1 to 5), the model performs better on the training data than on the test data, which is indicative of overfitting. The model is too complex and memorizes the training data, leading to poor generalization on unseen data.
- As the number of neighbors increases, the difference between training and test scores generally decreases, suggesting that the model is generalizing better.

Confusion Matrix for KNN Model



The following can be inferred from the confusion matrix and classification report:

- **Confusion Matrix:** 27 True Negatives, 29 True Positives, 2 False Positives, and 3 False Negatives, depicting the model's accuracy in predicting both negative and positive classes.
 - **Classification Report:** Precision for class 0 is 0.90, recall is 0.93, and F1-score is 0.92, while for class 1, precision is 0.94, recall is 0.91, and F1-score is 0.92, showcasing good model performance for both classes.
 - **Overall accuracy:** The model achieves an accuracy of 0.92 on the test set, indicating its capability to make correct predictions 92% of the time.
 - **Balanced Performance:** The model demonstrates similar metrics for precision, recall, and F1-score across both classes, suggesting balanced predictive performance.
 - **Slight Imbalance:** Although there is a small difference in false positives and false negatives, the model's overall performance remains robust, indicating a minimal bias towards either class.
- b. **Random Forest Classifier with Cross Validation:** This model provided a robust alternative with cross-validation accuracy averaging around 80.16%. Despite its comprehensive data handling capability, it was more complex and computationally intensive compared to KNN.

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import cross_val_score

    # Initialize the Random Forest classifier
    rf_classifier = RandomForestClassifier()

    # Perform cross-validation with 5 folds
    cv_scores = cross_val_score(rf_classifier, X_train, y_train, cv=5)

    # Print the cross-validation scores
    print("Cross-validation scores:", cv_scores)
    print("Mean CV score:", cv_scores.mean())

    # Train the Random Forest classifier on the entire training dataset
    rf_classifier.fit(X_train, y_train)

    # Make predictions using the trained model
    y_pred_rf = rf_classifier.predict(X_test)

    Cross-validation scores: [0.81632653 0.79591837 0.8125      0.79166667 0.79166667]
    Mean CV score: 0.8016156462585033
```

The following can be inferred from Cross Validation Scores of Random Forest Model:

- The line graph shows some variability in accuracy scores across five folds of cross validation, with scores ranging from approximately 79.5% to 81.5%.
- The variation suggests that the model's performance may depend on the particular subset of data used in each fold. The highest accuracy is observed in the third fold.

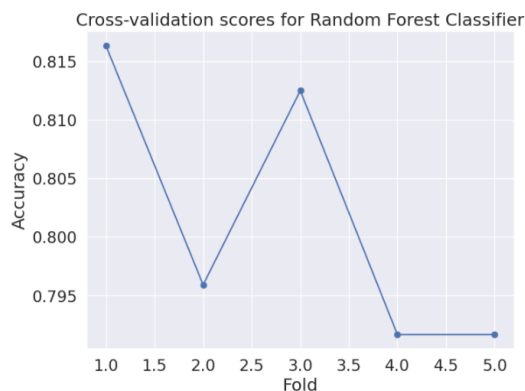
VISUALIZATION OF THE CROSS VALIDATION SCORES AND CONFUSION MATRIX

```
[ ] from sklearn.metrics import confusion_matrix

    # Plot the cross-validation scores
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, 6), cv_scores, marker='o', linestyle='-', color='b')
    plt.title('Cross-validation scores for Random Forest Classifier')
    plt.xlabel('Fold')
    plt.ylabel('Accuracy')
    plt.grid(True)
    plt.show()

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred_rf)

    # Plot the confusion matrix using seaborn
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix for Random Forest Classifier')
    plt.show()
```



The following can be inferred from the confusion matrix of Random Forest Model:

- Confusion Matrix reveals 24 true negatives and 28 true positives, indicating strong predictive performance. However, 5 false positives and 4 false negatives suggest some misclassifications.
- Precision and Recall: The classifier exhibits good precision and recall for both classes, with slightly better performance observed for class 1.

****Detailed inference for both models are included in the .ipynb notebook****

IV. Model Selection Justification:

The KNN model was chosen for the application due to its simplicity, efficiency, and the interpretability of its results, which are crucial for health-related predictions. KNN's ability to operate effectively with a smaller dataset and fewer hyperparameters to tune made it more suitable for this application, especially given the real-time nature of the Streamlit interface.

The K-Nearest Neighbors (KNN) algorithm was selected for its simplicity and effectiveness in classification tasks. The model was trained using a subset of the data, with features standardized to improve performance. Model evaluation was conducted using a split of training and testing data to ensure unbiased assessment.

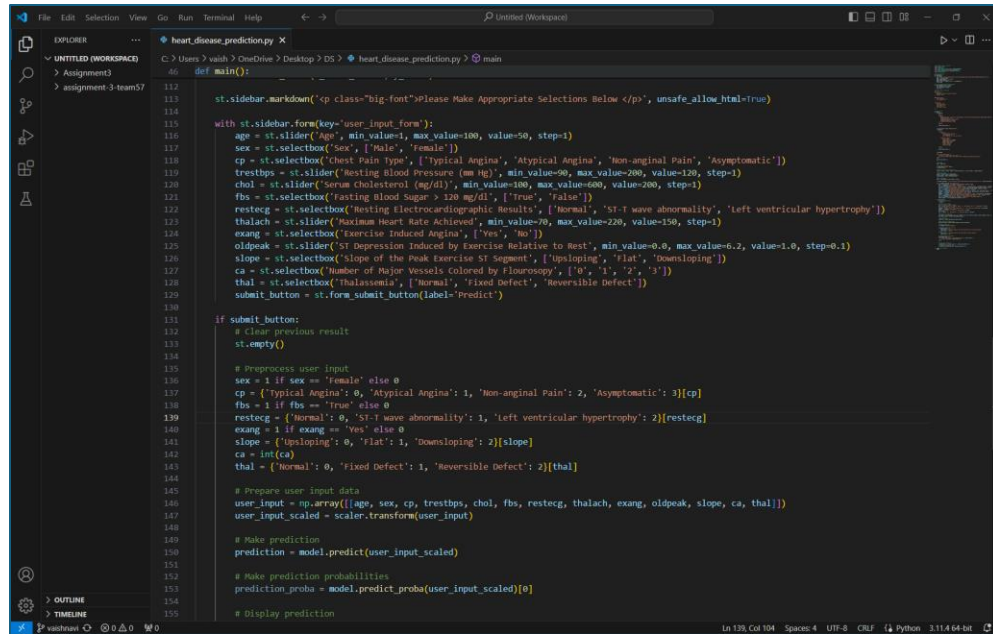
V. Application Design

The Streamlit application is structured to interactively collect user input through widgets and display predictions made by the KNN model. Key components of the code include:

- Data Loading: Utilizes Streamlit's caching to efficiently load and preprocess data.
- User Input Handling : Sliders and dropdowns allow users to specify their health metrics.
- Model Prediction : Once inputs are collected, they are processed by the KNN model to predict the likelihood of heart disease.
- Result Presentation: The prediction is displayed dynamically on the app interface.

The application's code leverages Python for backend operations (data handling, model training, and predictions) and Streamlit for the frontend interface, providing a seamless user experience.

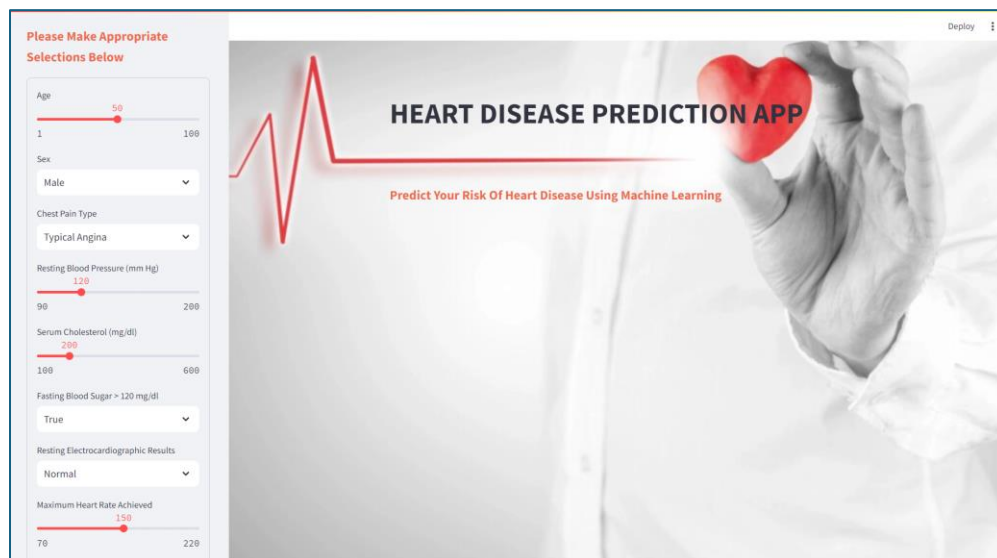
The application is designed using Streamlit, providing a user-friendly interface that includes input sliders and selection boxes for user-provided data and displays predictions dynamically. CSS is used to enhance the visual aspects of the application.



```

112 st.sidebar.markdown('Please Make Appropriate Selections Below </p>', unsafe_allow_html=True)
113
114 with st.sidebar.form(key='user_input_form'):
115     age = st.slider('Age', min_value=1, max_value=100, value=50, step=1)
116     sex = st.selectbox('Sex', ['Male', 'Female'])
117     cp = st.selectbox('Chest Pain Type', ['Typical Angina', 'Atypical Angina', 'Non-anginal Pain', 'Asymptomatic'])
118     trestbps = st.slider('Resting blood Pressure (mm Hg)', min_value=90, max_value=200, value=120, step=1)
119     chol = st.slider('Serum Cholesterol (mg/dl)', min_value=100, max_value=600, value=200, step=1)
120     fbs = st.selectbox('Fasting blood Sugar > 120 mg/dl', ['True', 'False'])
121     restecg = st.selectbox('Resting Electrocardiographic Results', ['Normal', 'ST-T wave abnormality', 'Left ventricular hypertrophy'])
122     thalach = st.slider('Maximum Heart Rate Achieved', min_value=70, max_value=220, value=150, step=1)
123     exang = st.selectbox('Exercise Induced Angina', ['Yes', 'No'])
124     oldpeak = st.slider('ST Depression Induced by Exercise Relative to Rest', min_value=0.0, max_value=0.2, value=0.0, step=0.1)
125     slope = st.selectbox('Slope of the Peak Exercise ST Segment', ['Upsloping', 'Flat', 'Downsloping'])
126     ca = st.selectbox('Number of Major Vessels colored by Fluoroscopy', ['0', '1', '2', '3'])
127     thal = st.selectbox('Thalassemia', ['Normal', 'Fixed Defect', 'Reversible Defect'])
128     submit_button = st.form_submit_button(label='Predict')
129
130 if submit_button:
131     # Clear previous result
132     st.empty()
133
134     # Preprocess user input
135     sex = 1 if sex == 'Female' else 0
136     cp = {'Typical Angina': 0, 'Atypical Angina': 1, 'Non-anginal Pain': 2, 'Asymptomatic': 3}[cp]
137     fbs = 1 if fbs == 'True' else 0
138     restecg = {'Normal': 0, 'ST-T wave abnormality': 1, 'Left ventricular hypertrophy': 2}[restecg]
139     exang = 1 if exang == 'Yes' else 0
140     slope = {'Upsloping': 0, 'Flat': 1, 'Downsloping': 2}[slope]
141     ca = int(ca)
142     thal = {'Normal': 0, 'Fixed Defect': 1, 'Reversible Defect': 2}[thal]
143
144     # Prepare user input data
145     user_input = np.array([age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal])
146     user_input_scaled = scaler.transform(user_input)
147
148     # Make prediction
149     prediction = model.predict(user_input_scaled)
150
151     # Make prediction probabilities
152     prediction_proba = model.predict_proba(user_input_scaled)[0]
153
154     # Display prediction

```



****The code(.py file) has been included in the zip file****

Results

The KNN model achieved a validation accuracy of approximately 90%, demonstrating its capability to effectively predict heart disease. The application allows users to input their health metrics and receive real-time predictions based on the trained model.

Use Case Snippets

The screenshot shows a web application titled "HEART DISEASE PREDICTION APP". On the left, there is a form titled "Please Make Appropriate Selections Below" with various input fields and sliders. The main area displays the prediction result: "The model predicts that the patient has heart disease!".

Input Fields:

- Age: 50
- Sex: Male
- Chest Pain Type: Typical Angina
- Resting Blood Pressure (mm Hg): 200
- Serum Cholesterol (mg/dl): 600
- Fasting Blood Sugar > 120 mg/dl: True
- Resting Electrocardiographic Results: Normal
- Maximum Heart Rate Achieved: 150
- Exercise Induced Angina: (checkbox)

Prediction Result: The model predicts that the patient has heart disease!

The screenshot shows the same web application as above, but with different input values. The prediction result is now: "The model predicts that the patient does not have heart disease!".

Input Fields:

- Age: 50
- Sex: Male
- Chest Pain Type: Typical Angina
- Resting Blood Pressure (mm Hg): 115
- Serum Cholesterol (mg/dl): 154
- Fasting Blood Sugar > 120 mg/dl: True
- Resting Electrocardiographic Results: Normal
- Maximum Heart Rate Achieved: (checkbox)

Prediction Result: The model predicts that the patient does not have heart disease!

Conclusion

The project successfully demonstrates the use of machine learning in predicting heart disease, with potential for further improvements and expansions. Future work could explore the integration of more complex models or additional features to enhance predictive accuracy.