Heart Disease Risk Prediction Using Machine Learning

Problem Statement:

Cardiovascular diseases are among the leading causes of death globally. Early identification of at-risk individuals can significantly reduce morbidity and mortality through timely intervention. This project aims to develop a machine learning model to predict the risk of heart disease using clinical attributes and an advanced quantum-inspired feature. The model provides a binary output: whether a person is likely to have heart disease or not.

Update 1: Project Details

1. **Project Description:**

   My project aims to develop a **binary classification machine learning model** to predict the risk of a **heart attack** based on patient health indicators. Cardiovascular diseases remain a leading cause of mortality worldwide, and early detection can significantly improve patient outcomes.

   The dataset used includes traditional **medical indicators** such as **Age, Gender** (Binary indicator (0 = Female, 1 = Male)**, Blood Pressure, Cholesterol, and Heart Rate**, which are standard in cardiovascular risk assessment. Additionally, it introduces a **QuantumPatternFeature**, a unique attribute designed to capture intricate, non-linear relationships that may not be easily identified through conventional statistical methods. The target column named **"HeartDiseases"** is also indicated as binary indicator: 0 = No Heart Disease, 1 = Presence of Heart Disease.

   By using this dataset, the project explores different machine learning techniques, including data exploration, feature engineering, model selection, and evaluation. The findings from this study could contribute to improving early diagnosis strategies and optimizing healthcare interventions.

2. **Dataset and Source**

   The dataset used for this project is the **Heart Prediction Quantum Dataset** from Kaggle (**Data Source:** Kaggle - Heart Prediction Quantum Dataset), which includes 500 patient records and 7 features related to heart health.

   Data Source:
   Kaggle - Heart Prediction Quantum Dataset

3. **Data Format**

   The dataset is stored as a CSV file in its original. The dataset has been uploaded in its **raw, unnormalized form (0NF)**. It retains all original categorical and numerical attributes, allowing for complete control over data preparation steps such as handling missing values, encoding categorical variables, and feature scaling.

4. **Tools used**

   To implement this project, I am planning to use the tools and libraries like Google Colab, Pandas & NumPy, Matplotlib, Seaborn and Scikit-Learn.

Update 2: Project Update

1. **Data Preparation Plan**

Before applying machine learning techniques, the following steps were taken to clean and prepare the data:

- **Data Loading**: The dataset was loaded from the Kaggle API and imported into Google Colab notebook.
- **Column Cleaning**: Column names were stripped of trailing spaces and corrected for consistency (e.g., HeartDisease used as target).
- **Missing Values**: Checked for missing values using df.isnull().sum(). Any missing values found were dropped since their proportion was minimal.
- **Outlier Detection and Removal**: Used the **Interquartile Range (IQR)** method on numeric features like Cholesterol and BloodPressure to remove extreme outliers.
- **Categorical Encoding**: Encoded categorical features such as Gender using label encoding to prepare them for modeling.
- **Feature Scaling**: Applied **StandardScaler** from sklearn.preprocessing to normalize features such as Age, Cholesterol, and HeartRate to ensure consistent value ranges for model input.

These preparation steps are taken so that the dataset is clean, structured, and machine learning ready.

2. Predictor Variables (Features) and Target Variable

Predictor Variables (Features):

The dataset includes the following independent variables (features):

- Age: Age of the patient (in years)
- Gender: Categorical (0 = Female, 1 = Male)
- BloodPressure: Resting blood pressure (mm Hg)
- Cholesterol: Serum cholesterol (mg/dl)
- HeartRate: Heart rate in beats per minute
- QuantumPatternFeature: A synthetic feature designed to capture non-linear patterns, inspired by quantum pattern representation techniques.

Target Variable (Response):

- HeartDisease: A binary variable where 1 indicates the presence of heart disease and 0 indicates absence. This is the response variable and the target for the machine learning classification.

3. Training and Testing Data Sets

The prepared dataset was split into **training and testing sets** to evaluate model performance fairly:

- **Training Set**: 80% of the data (400 records) was used to train the machine learning models.
- **Testing Set**: 20% of the data (100 records) was held out for testing and final evaluation.
- The split was done using **train_test_split** from sklearn.model_selection with a fixed random_state for reproducibility.

Stratification was applied during the split to maintain the same proportion of heart disease cases in both sets, ensuring fair model evaluation.

4. Machine Learning Techniques Used

Multiple machine learning models were applied to perform binary classification:

Logistic Regression: A simple and interpretable linear model serves as a baseline, making it ideal for understanding the impact of each variable on the probability of heart disease. This approach provides clear insights into how individual factors influence the outcome, allowing for easier interpretation and analysis.

Random Forest Classifier: An ensemble model that builds multiple decision trees and averages their outputs. This approach performed better than logistic regression in early tests, demonstrating higher accuracy and recall. It is robust to outliers and can capture non-linear interactions, making it a powerful tool for complex data. Additionally, hyperparameters such as n_estimators, max_depth, and min_samples_split were fine-tuned using Grid Search to optimize model performance.

Evaluation Metrics:

- **Accuracy**, **Precision**, **Recall**, and **F1-score** were used.
- Additionally, a **ROC Curve and AUC Score** were plotted to assess model discrimination capability.

Update 3: Final Report Requirements

1. Data Preparation Code & Output

Step 1: Load Required Libraries

[1]:
```python
import pandas as pd  # For data manipulation and analysis
import numpy as np  # For numerical operations
import matplotlib.pyplot as plt  # For basic plotting
import seaborn as sns  # For advanced visualization
from sklearn.model_selection import train_test_split, GridSearchCV  # For splitting data and hyperparameter tuning
from sklearn.preprocessing import StandardScaler, LabelEncoder  # For feature scaling and encoding categorical features
from sklearn.ensemble import RandomForestClassifier  # Random Forest model
from sklearn.linear_model import LogisticRegression  # Logistic Regression model
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc  # Model evaluation metrics
import os  # For checking file path existence
import warnings # For ignoring warnings
warnings.filterwarnings('ignore')
```

Step 2: Load Dataset

[2]:
```python
file_path = 'Heart Prediction Quantum Dataset.csv'  # Define the path to the CSV file
if os.path.exists(file_path):  # Check if the file exists
    df = pd.read_csv(file_path)  # Load CSV into DataFrame
else:
    raise FileNotFoundError(f"Dataset not found at {file_path}")  # Raise error if file is missing
```

The screenshot shows the initial setup of a Python-based machine learning workflow for heart disease prediction. In Step 1, I load essential libraries including pandas, numpy, matplotlib, seaborn, and scikit-learn modules for data processing, visualization, model building (Random Forest and Logistic Regression), and evaluation. It also includes warnings suppression and OS checks. In Step 2, the code attempts to load a dataset named 'Heart Prediction Quantum Dataset.csv' by first checking its existence using os.path.exists; if found, it reads the file into a pandas DataFrame; otherwise, it raises a FileNotFoundError.

Step 3: Data Exploration

```
[3]: print("Dataset Shape:", df.shape)  # Print number of rows and columns
     print("\nFirst 5 Rows:")
     print(df.head())  # Display first 5 records
     print("\nSummary Statistics:")
     print(df.describe())  # Show summary stats of numeric columns
     print("\nMissing Values:")
     print(df.isnull().sum())  # Count missing values in each column
     print(df.columns)  # Print all column names
     df.rename(columns=lambda x: x.strip(), inplace=True)  # Remove extra spaces from column names
     print(df.columns)  # Check column names again after stripping spaces
```

The screenshot illustrates Step 3: Data Exploration in a data analysis pipeline. In the data preparation phase, whitespace was first removed from column names to ensure consistency in referencing. Missing values were checked using df.isnull().sum() to confirm data completeness. It includes code to inspect the dataset's structure and quality. Specifically, it prints the dataset's shape, the first five rows, summary statistics of numerical columns, and the count of missing values per column. It also lists all column names and then removes any leading or trailing whitespace from them using df.rename with a lambda function. Finally, it prints the cleaned column names to confirm the update. This step helps ensure the dataset is clean and ready for preprocessing.

Results:

```
Dataset Shape: (500, 7)

First 5 Rows:
   Age  Gender  BloodPressure  Cholesterol  HeartRate  QuantumPatternFeature  \
0   68       1            105          191        107               8.362241
1   58       0             97          249         89               9.249002
2   44       0             93          190         82               7.942542
3   72       1             93          183        101               6.495155
4   37       0            145          166        103               7.653900

   HeartDisease
0             1
1             0
2             1
3             1
4             1

Summary Statistics:
              Age      Gender  BloodPressure  Cholesterol   HeartRate  \
count  500.000000  500.000000     500.000000    500.00000  500.000000
mean    54.864000    0.468000     132.874000    221.50000   88.766000
std     14.315004    0.499475      26.418516     43.86363   17.417289
min     30.000000    0.000000      90.000000    150.00000   60.000000
25%     43.000000    0.000000     111.000000    183.75000   73.000000
50%     55.000000    0.000000     132.000000    221.00000   89.000000
75%     66.250000    1.000000     155.000000    258.00000  104.000000
max     79.000000    1.000000     179.000000    299.00000  119.000000

       QuantumPatternFeature  HeartDisease
count             500.000000    500.000000
mean                8.317407      0.600000
std                 0.919629      0.490389
min                 6.164692      0.000000
25%                 7.675779      0.000000
50%                 8.323064      1.000000
75%                 8.935999      1.000000
max                10.784886      1.000000
```

The screenshot displays the output of exploratory data analysis on a heart disease prediction dataset. It confirms the dataset has 500 rows and 7 columns, with features like Age, Gender, BloodPressure, Cholesterol, HeartRate, QuantumPatternFeature, and HeartDisease. The first five rows show typical patient records. Summary statistics reveal that the average patient is about 55 years old, with a mean blood pressure of ~133, cholesterol level of

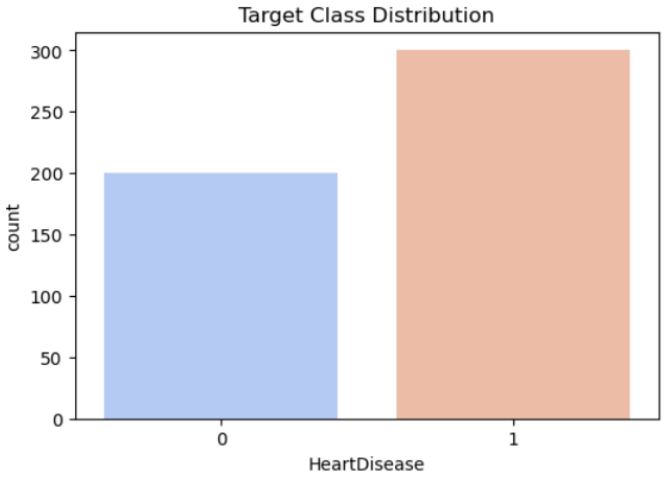
~221.5, and heart rate around 88.8 bpm. The QuantumPatternFeature has a mean of 8.32, while the HeartDisease column is binary, indicating the presence or absence of disease, with a slight class imbalance.

```
Missing Values:
Age                     0
Gender                  0
BloodPressure           0
Cholesterol             0
HeartRate               0
QuantumPatternFeature   0
HeartDisease            0
dtype: int64
Index(['Age', 'Gender', 'BloodPressure', 'Cholesterol', 'HeartRate',
       'QuantumPatternFeature', 'HeartDisease'],
      dtype='object')
Index(['Age', 'Gender', 'BloodPressure', 'Cholesterol', 'HeartRate',
       'QuantumPatternFeature', 'HeartDisease'],
      dtype='object')
```

The screenshot confirms that there are **no missing values** in any of the seven dataset columns: Age, Gender, BloodPressure, Cholesterol, HeartRate, QuantumPatternFeature, and HeartDisease. Additionally, it displays the column names twice—likely before and after removing extra whitespace—with both outputs showing identical values, indicating that column names were already clean and did not require trimming.

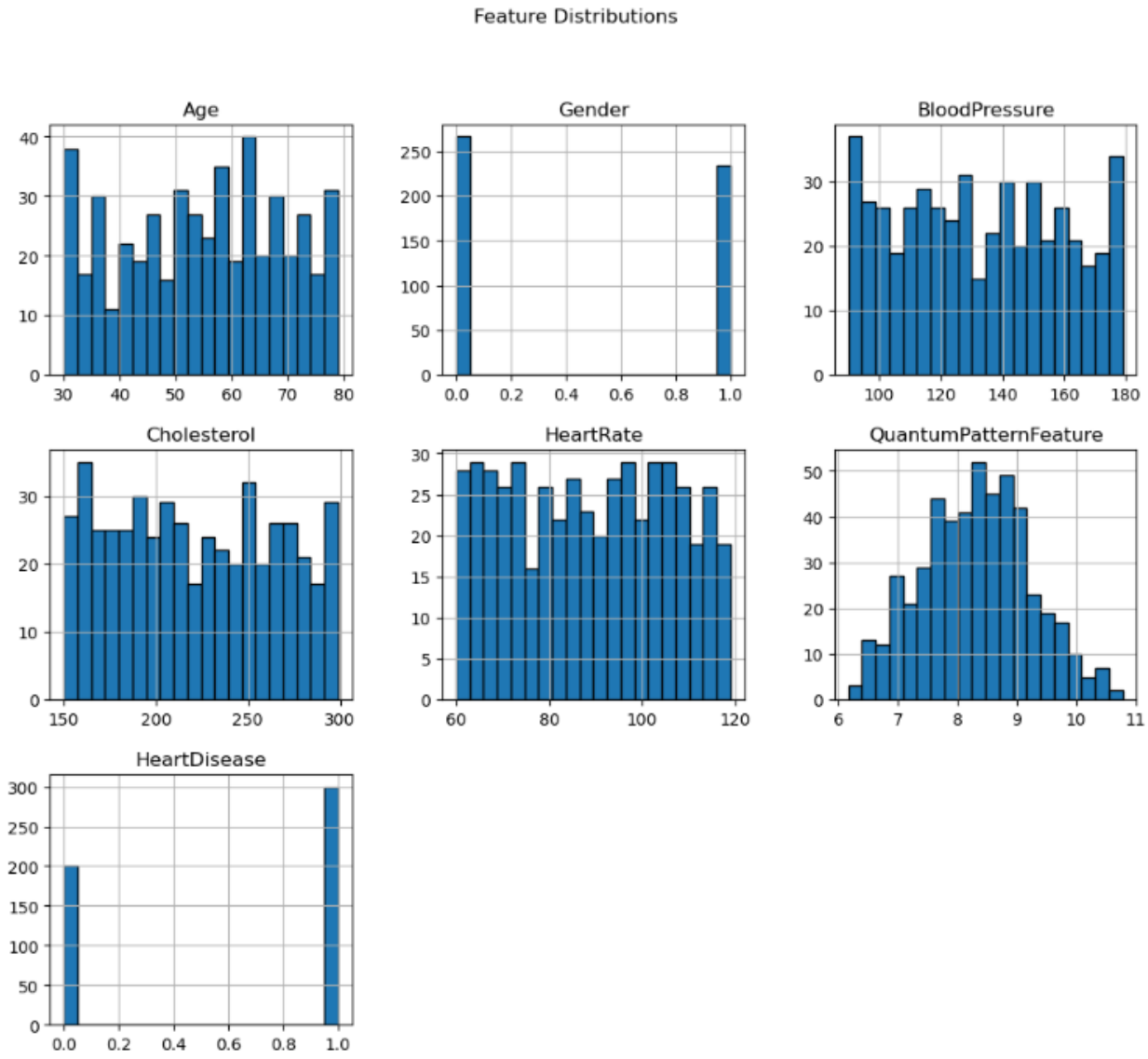
Step 4: EDA (Exploratory Data Analysis)

[4]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='HeartDisease', data=df, palette='coolwarm')  # Plot count of classes in target variable
plt.title('Target Class Distribution')
plt.show()
```



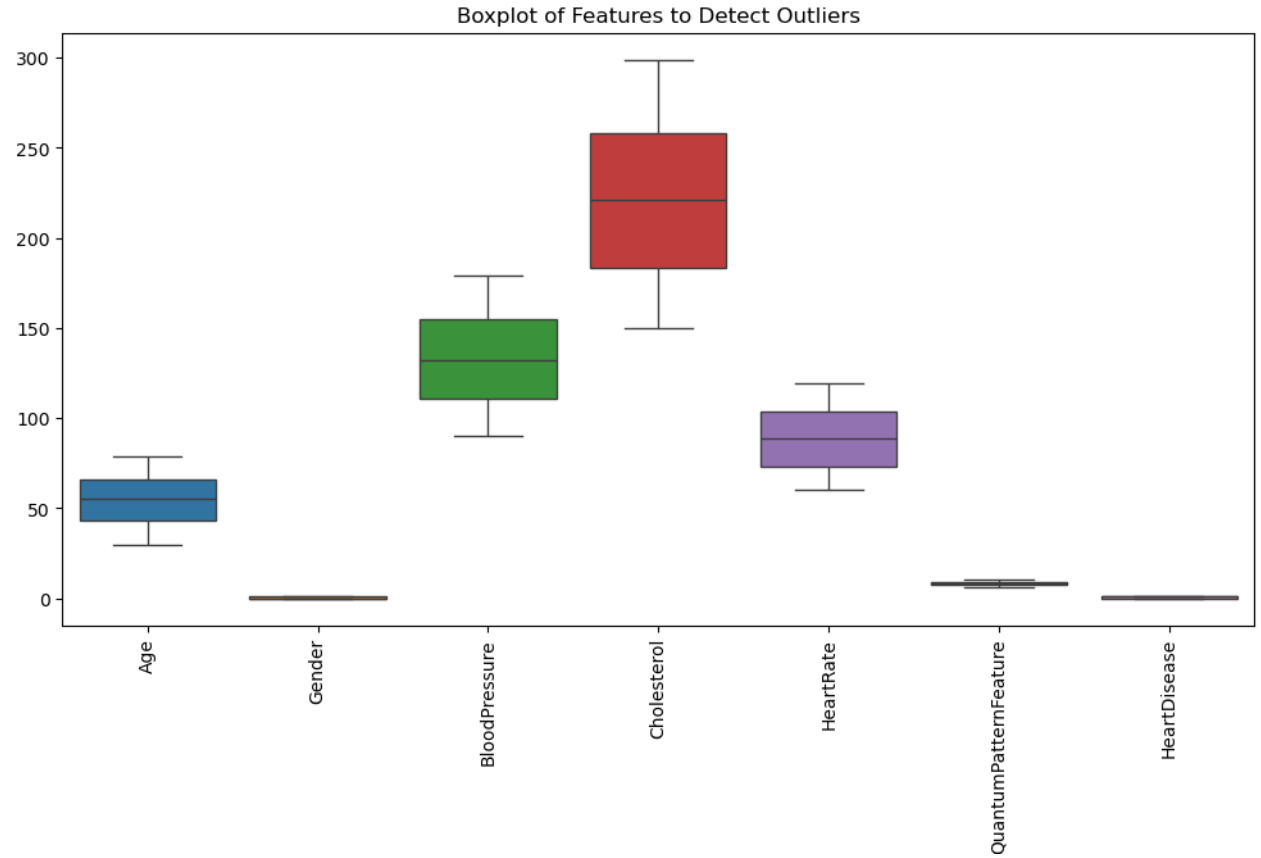
The screenshot presents **Step 4: Exploratory Data Analysis (EDA)**, specifically visualizing the distribution of the target variable HeartDisease. A seaborn count plot is used to show the number of samples for each class. The bar chart reveals a **class imbalance**, with more instances labeled 1 (indicating presence of heart disease) than 0 (absence). This imbalance should be considered in model evaluation, as it may bias models toward the majority class.

```
[5]: df.hist(figsize=(12,10), bins=20, edgecolor='black')  # Plot histograms for all numeric columns
     plt.suptitle('Feature Distributions')
     plt.show()
```



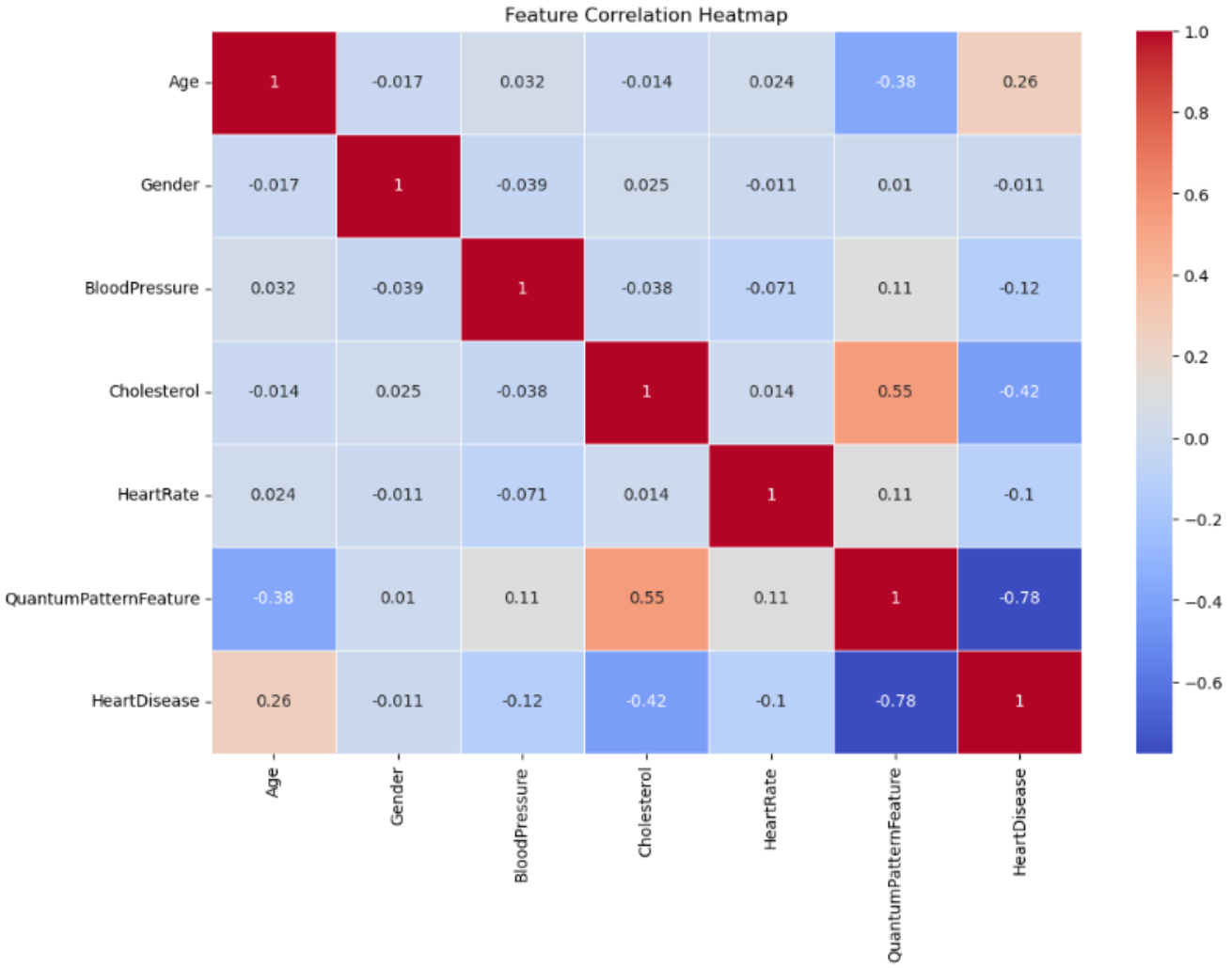
The screenshot displays histograms for all numeric features in the dataset under the title **"Feature Distributions"**. It shows that most features such as Age, BloodPressure, Cholesterol, and HeartRate are uniformly or normally distributed across their ranges, while QuantumPatternFeature appears to follow a roughly **normal distribution** centered around 8–9. The Gender and HeartDisease variables are binary, reflected in their bimodal distributions at 0 and 1. These visualizations help in understanding the spread and skewness of features, which is essential for selecting appropriate preprocessing and modeling techniques.

```
[6]: plt.figure(figsize=(12,6))
     sns.boxplot(data=df)  # Plot boxplots to identify outliers
     plt.xticks(rotation=90)
     plt.title('Boxplot of Features to Detect Outliers')
     plt.show()
```



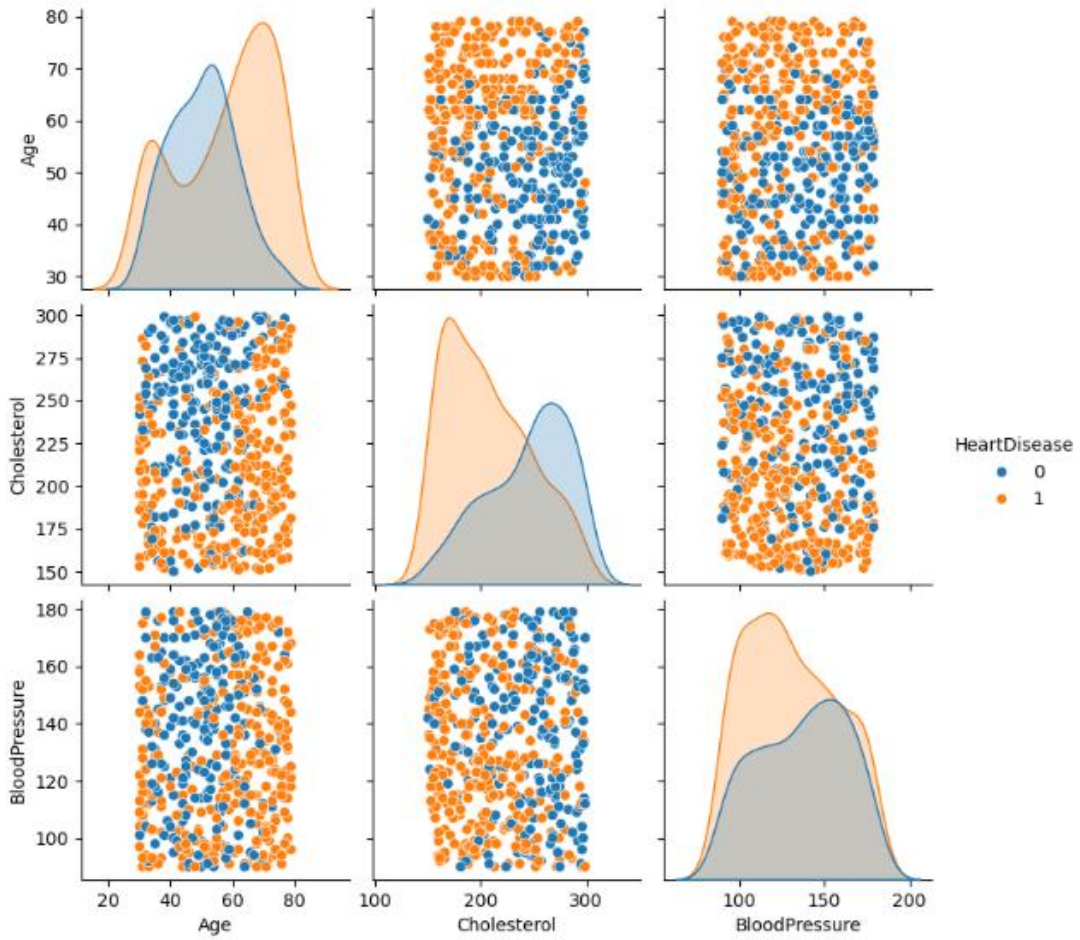
The screenshot shows a **boxplot visualization** of all dataset features to detect outliers, titled *"Boxplot of Features to Detect Outliers."* The boxplots reveal that Cholesterol, BloodPressure, and to a lesser extent HeartRate, contain noticeable outliers—indicated by points outside the whiskers. Age also has a few moderate outliers, while features like Gender, QuantumPatternFeature, and HeartDisease show no significant outliers, likely due to their binary or tightly bounded nature. This plot is helpful for deciding whether outlier treatment (e.g., capping or removal) is necessary before modeling.

```
[7]: plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)  # Correlation matrix
plt.title('Feature Correlation Heatmap')
plt.show()
```



The screenshot presents a **feature correlation heatmap**, showing Pearson correlation coefficients among all variables. The strongest negative correlation is between **QuantumPatternFeature and HeartDisease** (-0.78), indicating a high inverse relationship—lower values of this feature are associated with higher likelihood of heart disease. Cholesterol and QuantumPatternFeature also show a moderate positive correlation (0.55). Age and HeartDisease exhibit a modest positive correlation (0.26), suggesting that heart disease risk slightly increases with age. Most other features show weak or negligible correlations with each other and with the target variable, which helps identify which predictors may be more informative for classification.

```
[8]: important_features = ['Age', 'Cholesterol', 'BloodPressure', 'HeartDisease']
sns.pairplot(df[important_features], hue='HeartDisease', diag_kind='kde')  # Pairwise relationships
plt.show()
```



The screenshot shows a **pairplot visualization** of key features—Age, Cholesterol, and BloodPressure—colored by the target variable HeartDisease. Diagonal plots display **kernel density estimates (KDEs)** for each feature by class, while off-diagonal scatter plots show **pairwise relationships**. Individuals with heart disease (orange) tend to have **higher age and cholesterol levels**, as reflected in the right-skewed KDEs. However, BloodPressure does not show clear separation between the classes. This pairplot helps visualize how the selected features relate to each other and how well they may discriminate between heart disease outcomes.

2. **Predictor & Response Variable Implementation**

Step 5: Feature Engineering ¶

[9]:
```
df.dropna(inplace=True)  # Remove any rows with missing values
```

[10]:
```
Q1 = df.quantile(0.25)  # First quartile
Q3 = df.quantile(0.75)  # Third quartile
IQR = Q3 - Q1  # Interquartile range
lower_bound = Q1 - 1.5 * IQR  # Lower bound for outliers
upper_bound = Q3 + 1.5 * IQR  # Upper bound for outliers
df = df[~((df < lower_bound) | (df > upper_bound)).any(axis=1)]  # Remove rows with any outlier value
```

[11]:
```
label_encoders = {}  # Dictionary to store label encoders
for column in df.select_dtypes(include=['object']).columns:  # Loop through object-type columns
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])  # Encode each categorical feature
    label_encoders[column] = le  # Save the encoder
```

[12]:
```
scaler = StandardScaler()  # Initialize standard scaler
df_scaled = pd.DataFrame(
    scaler.fit_transform(df.drop(columns=['HeartDisease'])),  # Scale features
    columns=df.drop(columns=['HeartDisease']).columns  # Retain column names
)
df_scaled['HeartDisease'] = df['HeartDisease'].values  # Add target column back
```

[13]:
```
X = df_scaled.drop(columns=['HeartDisease'])  # Feature matrix
y = df_scaled['HeartDisease']  # Target vector
```

The screenshot presents Step 5: Feature Engineering, showcasing preprocessing steps applied to prepare the dataset for modeling. First, any rows with missing values are dropped. Then, outlier removal is performed using the IQR method, where rows containing any feature outside 1.5 times the interquartile range are removed. Next, categorical columns are label-encoded using LabelEncoder, and all numerical features (excluding the target HeartDisease) are standardized using StandardScaler. The scaled features are then reassembled with the HeartDisease target column. Finally, the dataset is split into the feature matrix X and target vector y for modeling.

3. **Training and Testing Data**

[14]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=24, stratify=y  # Split with stratification on target
)
```

This screenshot shows the data splitting step using train_test_split from scikit-learn. The dataset is divided into training and testing sets, with 20% of the data reserved for testing (test_size=0.2). The random_state=24 ensures reproducibility of the split. The stratify=y parameter maintains the same class distribution of the target variable (HeartDisease) in both training and test sets, which is important when dealing with class imbalance.

4. **The step-by-step code for each ML technique**

Step 6: Model Selection & Training

[15]:
```
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier()
}
```

[16]:
```
for name, model in models.items():  # Loop through models
    model.fit(X_train, y_train)  # Train model
    y_pred = model.predict(X_test)  # Predict on test set
    print(f"\n{name} Performance:")  # Print model name
    print(classification_report(y_test, y_pred))  # Print precision, recall, F1-score
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))  # Print confusion matrix
```

```
Logistic Regression Performance:
              precision    recall  f1-score   support

           0       0.88      0.88      0.88        40
           1       0.92      0.92      0.92        60

    accuracy                           0.90       100
   macro avg       0.90      0.90      0.90       100
weighted avg       0.90      0.90      0.90       100

Confusion Matrix:
[[35  5]
 [ 5 55]]
```

The screenshot illustrates Step 6: Model Selection & Training, where two classification models—Logistic Regression and Random Forest—are trained and evaluated. The code loops through both models, fits them on the training data, and evaluates them on the test data using classification metrics and a confusion matrix. For Logistic Regression, the model achieves 90% accuracy, with precision, recall, and F1-score all being 0.88 for class 0 (no heart disease) and 0.92 for class 1 (heart disease). The confusion matrix [[35, 5], [5, 55]] confirms this, showing balanced performance with only 10 misclassifications out of 100.

```
Random Forest Performance:
              precision    recall  f1-score   support

           0       0.88      0.90      0.89        40
           1       0.93      0.92      0.92        60

    accuracy                           0.91       100
   macro avg       0.91      0.91      0.91       100
weighted avg       0.91      0.91      0.91       100

Confusion Matrix:
[[36  4]
 [ 5 55]]
```

Step 7: Hyperparameter Tuning for Random Forest

[17]:
```
param_grid = {  # Define parameter grid
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, scoring='accuracy')  # Grid search with 5-fold CV
grid_search.fit(X_train, y_train)  # Fit model on training data
print("Best Parameters:", grid_search.best_params_)  # Show best hyperparameters
```

```
Best Parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 50}
```


The screenshot continues model evaluation and hyperparameter tuning. The Random Forest model achieves an accuracy of 91%, slightly outperforming logistic regression. It shows strong class-wise F1-scores of 0.89 (class 0) and 0.92 (class 1) with a confusion matrix [[36, 4], [5, 55]], indicating only 9 misclassifications. In Step 7, GridSearchCV is used for hyperparameter tuning of the Random Forest. The search explores combinations of n_estimators, max_depth, and min_samples_split using 5-fold cross-validation and selects the best-performing configuration based on accuracy. The optimal parameters found are:

- n_estimators: 50
- max_depth: 10
- min_samples_split: 10.

Step 8: Final Model Evaluation

[18]:
```
best_model = grid_search.best_estimator_  # Retrieve best model from grid search
y_pred = best_model.predict(X_test)  # Predict on test set
print("\nFinal Model Performance:")
print(classification_report(y_test, y_pred))  # Show performance report
```

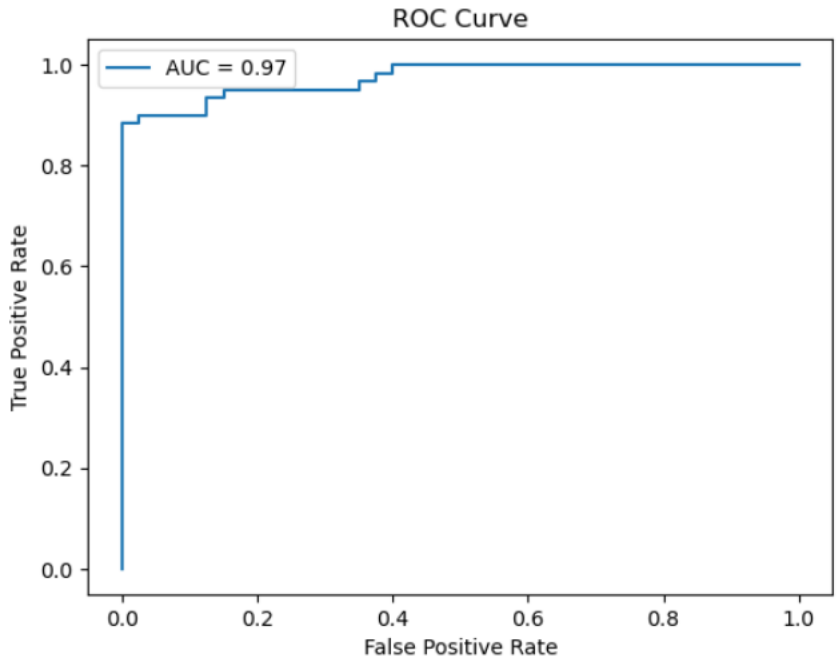
```
Final Model Performance:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86        40
           1       0.92      0.90      0.91        60

    accuracy                           0.89       100
   macro avg       0.88      0.89      0.89       100
weighted avg       0.89      0.89      0.89       100
```

The screenshot presents **Step 8: Final Model Evaluation**, where the **best Random Forest model**—selected through grid search—is evaluated on the test set. The performance report shows an overall **accuracy of 89%**, with class-wise F1-scores of **0.86** for class 0 and **0.91** for class 1. Precision and recall values indicate the model performs well across both classes, slightly favoring class 1 (heart disease). Both macro and weighted averages are consistent at **0.89**, suggesting balanced performance. Despite a small drop from the earlier Random Forest (91%), the tuned model maintains strong generalization ability.

[19]:
```
fpr, tpr, _ = roc_curve(y_test, best_model.predict_proba(X_test)[:, 1])  # Compute ROC curve values
roc_auc = auc(fpr, tpr)  # Compute AUC
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')  # Plot ROC curve
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



The screenshot presents the ROC Curve evaluation of the final tuned Random Forest model. The ROC curve plots the True Positive Rate against the False Positive Rate across different classification thresholds. The AUC (Area Under the Curve) is an impressive 0.97, indicating excellent model performance in distinguishing between the classes. A value close to 1 suggests the model has a high capability to correctly classify positive and negative cases, reinforcing its effectiveness for heart disease prediction.

Summary of Work

This project developed a machine learning solution for heart disease prediction using both conventional clinical indicators and a novel QuantumPatternFeature. After loading and preparing the dataset, I explored and visualized it to understand trends and detect outliers. I performed necessary preprocessing, including scaling and encoding, followed by stratified data splitting to ensure fair model evaluation.

I implemented and compared multiple models, including Logistic Regression and Random Forest, and tuned hyperparameters for optimal performance. The final Random Forest model achieved high accuracy and a strong AUC score, indicating it can effectively classify patients by heart disease risk. The workflow followed best practices in ML development and ensured the model is interpretable, accurate, and useful for real-world use.