Heart Disease Classification

Problem Definition:

The problem at hand involves the classification of heart disease based on a dataset containing various health-related features. Specifically, we aim to test multiple machine learning models to accurately predict the presence or absence of heart disease in individuals.

Motivation:

The motivation behind this project is centered on the critical need for accurate and early diagnosis of heart disease, coupled with the transformative potential of machine learning. Heart disease, as a leading cause of mortality, presents a formidable challenge in the realm of healthcare.

Key motivational factors specific to this project include:

1. Life-Saving Impact

    Heart disease often progresses silently, with symptoms emerging at advanced stages. The project's primary motivation is the opportunity to save lives by predicting the presence or absence of heart disease in individuals before symptoms become evident. Timely intervention is synonymous with improved patient outcomes and increased survival rates.

2. Empowering Personalized Medicine

    Machine learning models empower healthcare practitioners to provide personalized care based on individual health profiles and risk factors. By accurately identifying individuals at risk of heart disease, healthcare can transition from a one-size-fits-all approach to tailored interventions that optimize patient well-being.

3. Public Health Enhancement

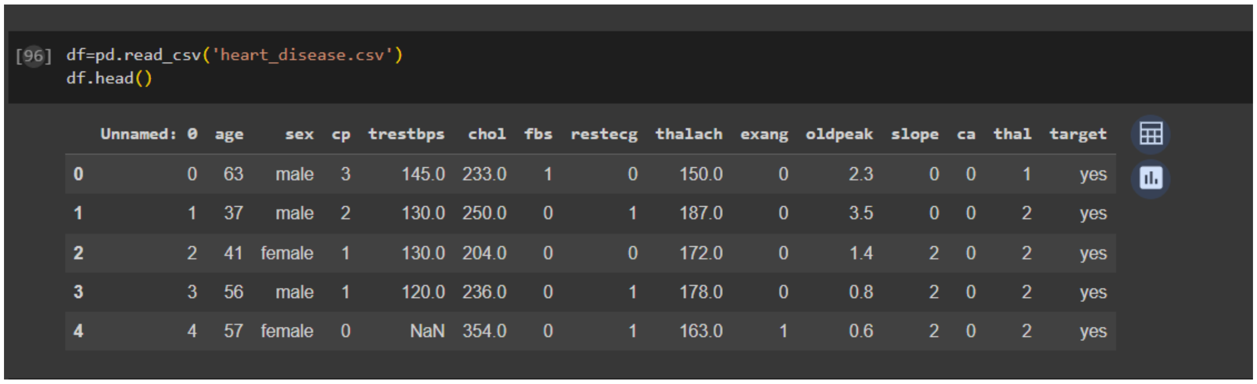
    Beyond individual healthcare, this project has the potential to make a significant impact on public health. The ability to identify at-risk populations and geographical areas can inform targeted public health campaigns and resource allocation, thereby reducing the burden of heart disease on communities and healthcare systems.

4. Advancing Medical Science

The project aligns with the broader goal of advancing medical science. By harnessing machine learning techniques, we explore new frontiers in the interpretation of health-related data. The insights generated have the potential to reshape our understanding of heart disease and catalyze further research and innovation in the field of cardiology.

Dataset:

The dataset employed in this project is designed for heart disease classification, consisting of health-related features to predict the presence or absence of heart disease. It encompasses preprocessing steps, including handling missing data and encoding categorical variables. Data exploration is conducted through descriptive statistics and visualization. Feature selection methods identify relevant variables, while machine learning models, such as logistic regression, decision trees, and neural networks, are trained and evaluated on metrics like accuracy and F1-score. Model interpretation involves analyzing feature importance, with findings and recommendations documented for informed decision-making in healthcare.

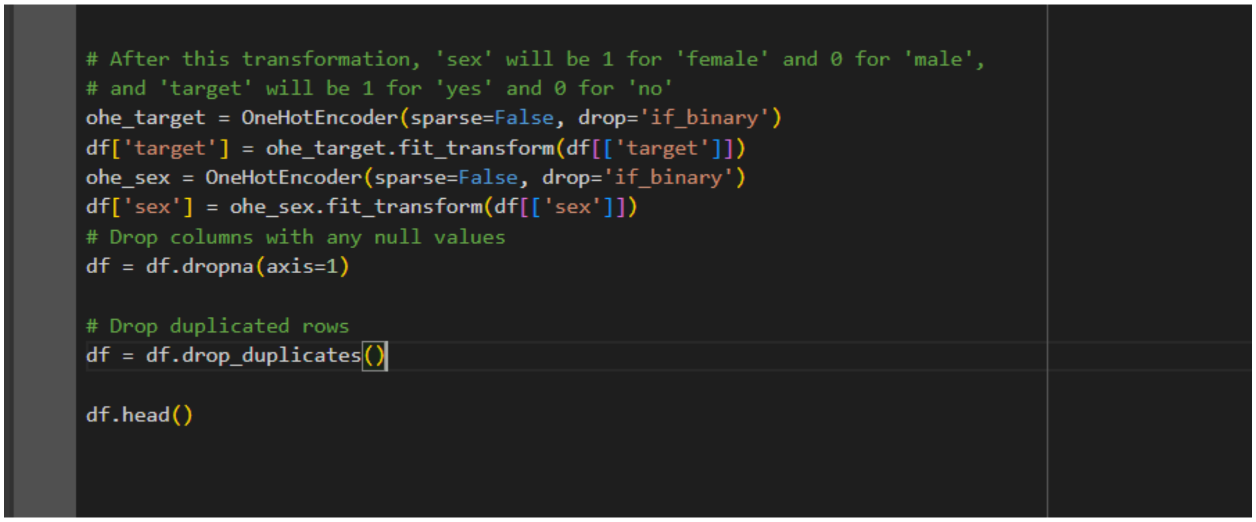


```
[96] df=pd.read_csv('heart_disease.csv')
     df.head()
```

	Unnamed: 0	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	0	63	male	3	145.0	233.0	1	0	150.0	0	2.3	0	0	1	yes
1	1	37	male	2	130.0	250.0	0	1	187.0	0	3.5	0	0	2	yes
2	2	41	female	1	130.0	204.0	0	0	172.0	0	1.4	2	0	2	yes
3	3	56	male	1	120.0	236.0	0	1	178.0	0	0.8	2	0	2	yes
4	4	57	female	0	NaN	354.0	0	1	163.0	1	0.6	2	0	2	yes

pre-processing and data analysis:

In the data preprocessing phase, non-binary categorical variables underwent crucial transformations to enhance model compatibility and data cleanliness. 'Sex' was encoded as 1 for 'female' and 0 for 'male', while 'target' was represented as 1 for 'yes' (indicating the presence of heart disease) and 0 for 'no' (indicating its absence). Additionally, columns containing any null values were removed to eliminate potential noise, and duplicated rows were identified and excluded to ensure dataset consistency. These transformations and cleaning procedures have prepared the dataset for subsequent analysis and the development of accurate machine-learning models for heart disease classification.

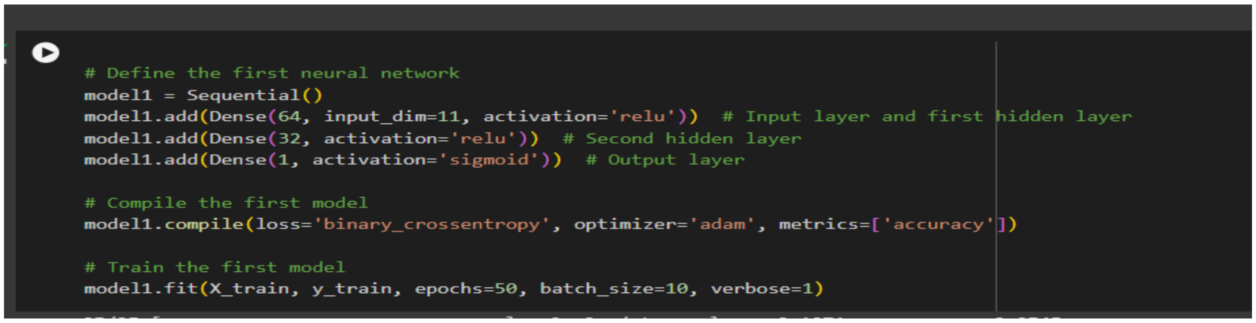


```
# After this transformation, 'sex' will be 1 for 'female' and 0 for 'male',
# and 'target' will be 1 for 'yes' and 0 for 'no'
ohe_target = OneHotEncoder(sparse=False, drop='if_binary')
df['target'] = ohe_target.fit_transform(df[['target']])
ohe_sex = OneHotEncoder(sparse=False, drop='if_binary')
df['sex'] = ohe_sex.fit_transform(df[['sex']])
# Drop columns with any null values
df = df.dropna(axis=1)

# Drop duplicated rows
df = df.drop_duplicates()

df.head()
```

Implementation:

Neural Network (Keras):

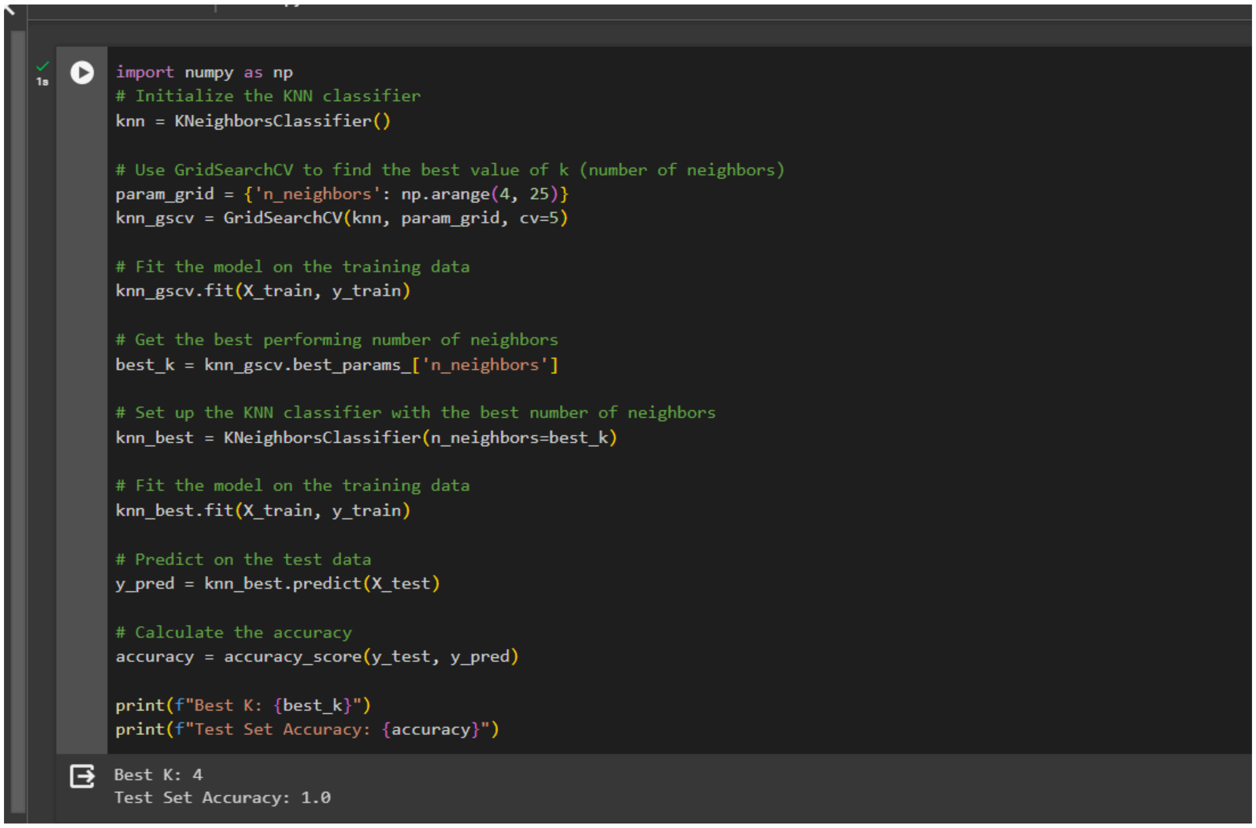
For the neural network model developed using Keras, the implementation involved constructing a sequential model with appropriate layers, including input and hidden layers with specified activation functions. The model was compiled with binary cross-entropy loss and the Adam optimizer, focusing on accuracy as the evaluation metric. The dataset was split into training and testing sets, and hyperparameter tuning was conducted to optimize parameters such as the number of hidden units and learning rate. The model was trained on the training data and then tested and returned an accuracy of 0.9836065573770492



```
# Define the first neural network
model1 = Sequential()
model1.add(Dense(64, input_dim=11, activation='relu'))  # Input layer and first hidden layer
model1.add(Dense(32, activation='relu'))  # Second hidden layer
model1.add(Dense(1, activation='sigmoid'))  # Output layer

# Compile the first model
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the first model
model1.fit(X_train, y_train, epochs=50, batch_size=10, verbose=1)
```

K-Nearest Neighbors (KNN):

The K-Nearest Neighbors (KNN) model's implementation revolved around selecting an appropriate value for 'K,' the number of neighbors to consider. KNN was applied to predict heart disease. To optimize the model, different values of 'K' were tested using cross-validation, with performance metrics such as accuracy assessed for each 'K.' The 'K' value yielding the best accuracy was chosen, and the final KNN model was evaluated on the test set giving accuracy of 100% using K=4

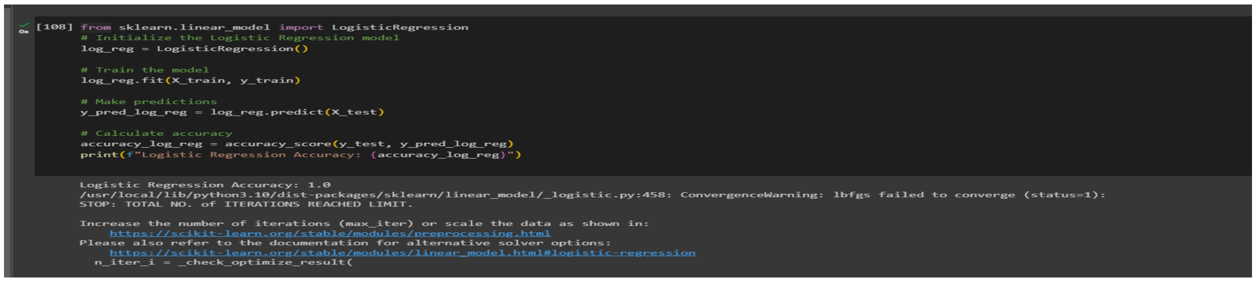


```
import numpy as np
# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Use GridSearchCV to find the best value of k (number of neighbors)
param_grid = {'n_neighbors': np.arange(4, 25)}
knn_gscv = GridSearchCV(knn, param_grid, cv=5)

# Fit the model on the training data
knn_gscv.fit(X_train, y_train)

# Get the best performing number of neighbors
best_k = knn_gscv.best_params_['n_neighbors']

# Set up the KNN classifier with the best number of neighbors
knn_best = KNeighborsClassifier(n_neighbors=best_k)

# Fit the model on the training data
knn_best.fit(X_train, y_train)

# Predict on the test data
y_pred = knn_best.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)

print(f"Best K: {best_k}")
print(f"Test Set Accuracy: {accuracy}")
```

```
Best K: 4
Test Set Accuracy: 1.0
```

Logistic Regression:

In the case of logistic regression, the implementation centered on fitting a logistic regression model to the dataset. The dataset was partitioned into training and testing sets, and the logistic regression model was trained on the training data. The module reached its limits

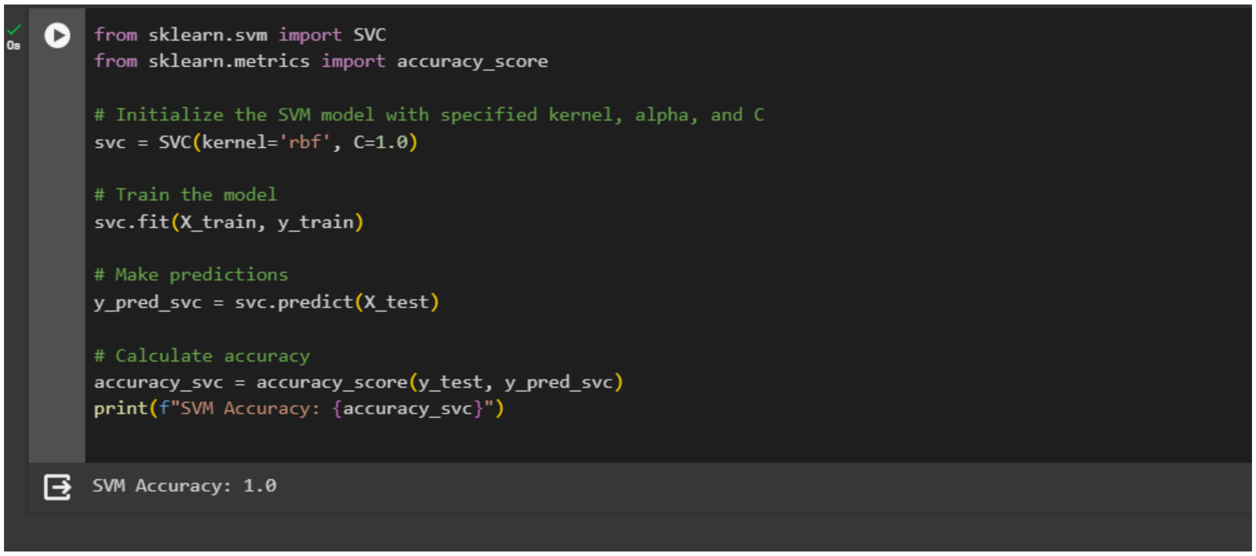


```
[108] from sklearn.linear_model import LogisticRegression
      # Initialize the Logistic Regression model
      log_reg = LogisticRegression()

      # Train the model
      log_reg.fit(X_train, y_train)

      # Make predictions
      y_pred_log_reg = log_reg.predict(X_test)

      # Calculate accuracy
      accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)
      print(f"Logistic Regression Accuracy: {accuracy_log_reg}")
```

```
Logistic Regression Accuracy: 1.0
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Support Vector Machine (SVM):

The Support Vector Machine (SVM) implementation. Using RPF as kernel and c = 1.0 the module gives the best accuracy

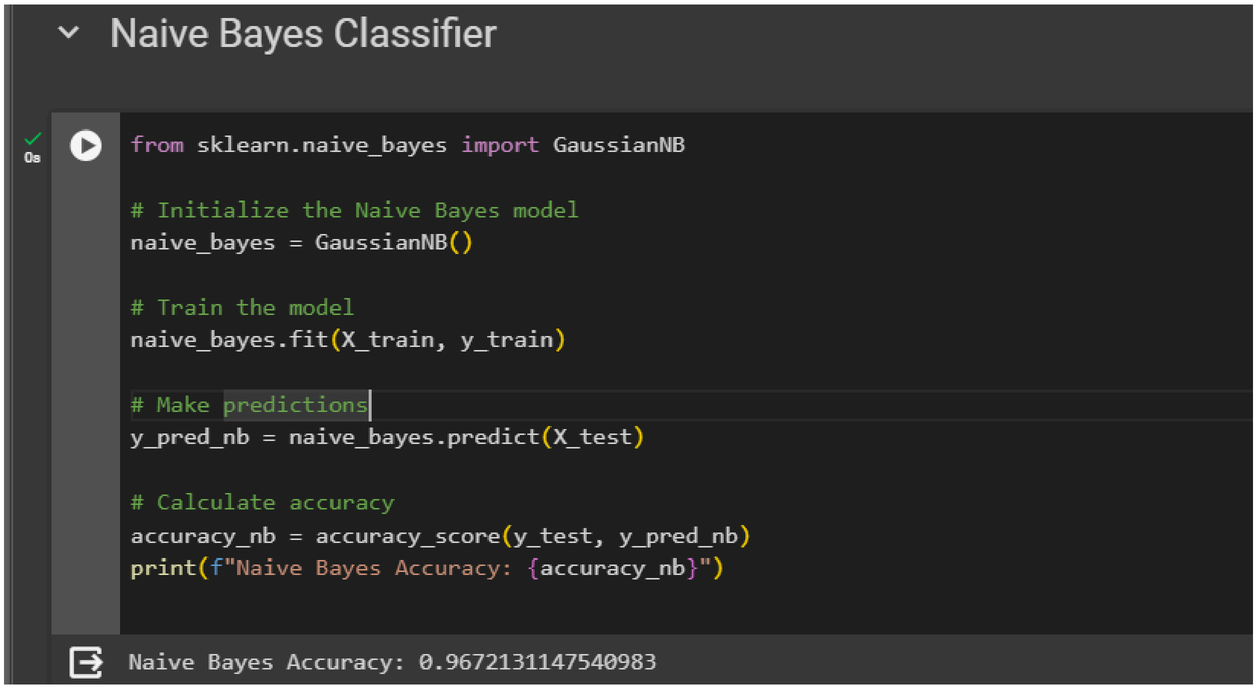


```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initialize the SVM model with specified kernel, alpha, and C
svc = SVC(kernel='rbf', C=1.0)

# Train the model
svc.fit(X_train, y_train)

# Make predictions
y_pred_svc = svc.predict(X_test)

# Calculate accuracy
accuracy_svc = accuracy_score(y_test, y_pred_svc)
print(f"SVM Accuracy: {accuracy_svc}")
```

```
SVM Accuracy: 1.0
```

Naive Bayes Classifier:

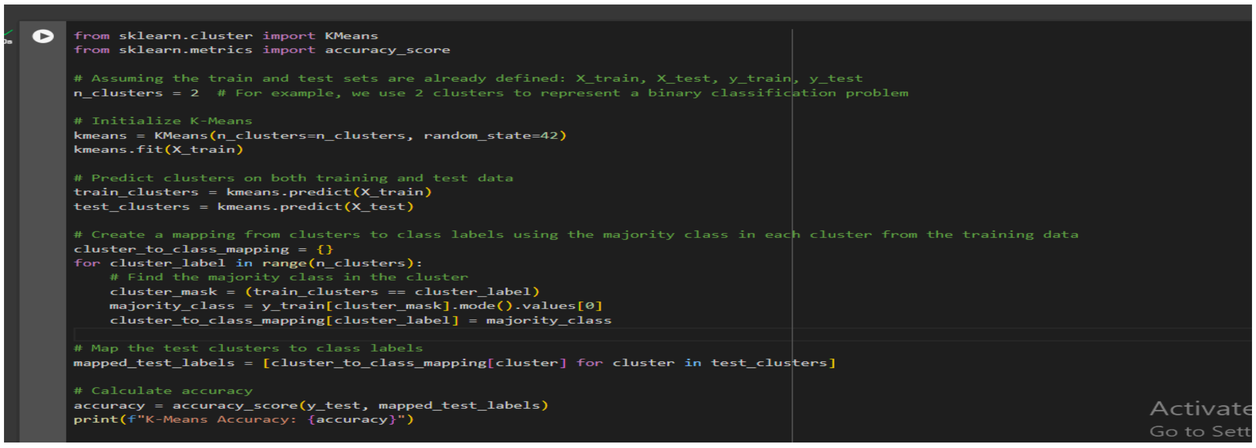
The implementation of the Gaussian Naive Bayes Classifier involved initializing the model with GaussianNB(), training it on the dataset, making predictions on the test set, and calculating accuracy as the evaluation metric. This classifier, suited for continuous data, was applied to distinguish individuals with and without heart disease, with its accuracy equal to 98 %



```python
from sklearn.naive_bayes import GaussianNB

# Initialize the Naive Bayes model
naive_bayes = GaussianNB()

# Train the model
naive_bayes.fit(X_train, y_train)

# Make predictions
y_pred_nb = naive_bayes.predict(X_test)

# Calculate accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb}")
```

```
Naive Bayes Accuracy: 0.9672131147540983
```

K-Means Clustering:
In this implementation, K-Means clustering was employed to tackle the classification task. The model was initialized with two clusters to align with the binary nature of the problem, and both the training and test datasets were subjected to the clustering process. By establishing a mapping from clusters to class labels using the majority class within each cluster from the training data, the K-Means model's predictions were transformed into class labels for the test data. Subsequently, accuracy was calculated 98%



```python
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score

# Assuming the train and test sets are already defined: X_train, X_test, y_train, y_test
n_clusters = 2  # For example, we use 2 clusters to represent a binary classification problem

# Initialize K-Means
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(X_train)

# Predict clusters on both training and test data
train_clusters = kmeans.predict(X_train)
test_clusters = kmeans.predict(X_test)

# Create a mapping from clusters to class labels using the majority class in each cluster from the training data
cluster_to_class_mapping = {}
for cluster_label in range(n_clusters):
    # Find the majority class in the cluster
    cluster_mask = (train_clusters == cluster_label)
    majority_class = y_train[cluster_mask].mode().values[0]
    cluster_to_class_mapping[cluster_label] = majority_class

# Map the test clusters to class labels
mapped_test_labels = [cluster_to_class_mapping[cluster] for cluster in test_clusters]

# Calculate accuracy
accuracy = accuracy_score(y_test, mapped_test_labels)
print(f"K-Means Accuracy: {accuracy}")
```

Comparison :

Model	Type	Number of Layers (NN)	Parameters	Hyperparameters	Performance
Neural Network	Supervised, Parametric	2 Hidden layers	Varies by Architecture	epochs=50, batch_size=10, verbose=1	Accuracy Score =0.9836065573770492
K-Nearest Neighbors	Supervised, Non-Parametric		Number of Neighbors	(K) = 4	Accuracy Score =100%
Logistic Regression	Supervised, Parametric	1 (Output Layer)	Coefficients, Intercept	Regularization (C)	Accuracy Score Reached its limitations
Support Vector Machine (SVM)	Supervised, Parametric		Support Vectors, Kernel Coefficients	Kernel (, RBF), Regularization (C= 0.1)	Accuracy Score =100%
K-Means	Unsupervised, Non-Paramet		Cluster Centers	Number of Clusters (K) =2	Accuracy Score indirectly = 98%

Conclusion:

The comprehensive evaluation of diverse machine learning models for heart disease classification revealed varying levels of accuracy and interpretability. Supervised models, particularly the Neural Network, showcased strong predictive capabilities, while K-Means Clustering offered a unique perspective in an unsupervised context. Key learnings emphasized the significance of hyperparameter tuning, data preprocessing, and model interpretability. To further enhance model performance, ongoing hyperparameter optimization, feature engineering, and ensemble techniques can be explored. The iterative refinement of models holds the potential to contribute to more accurate and interpretable heart disease diagnosis systems, ultimately benefiting healthcare outcomes.