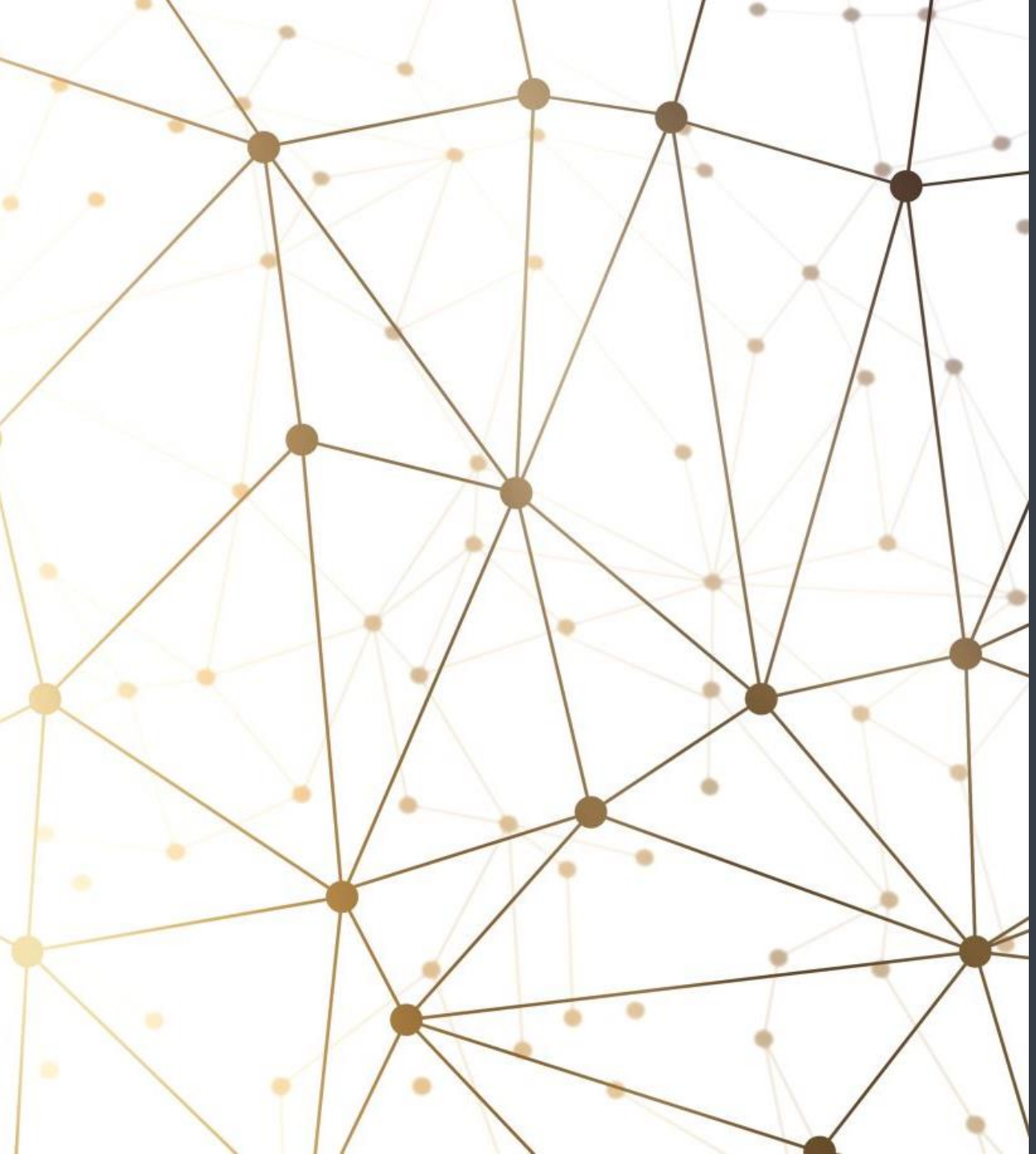
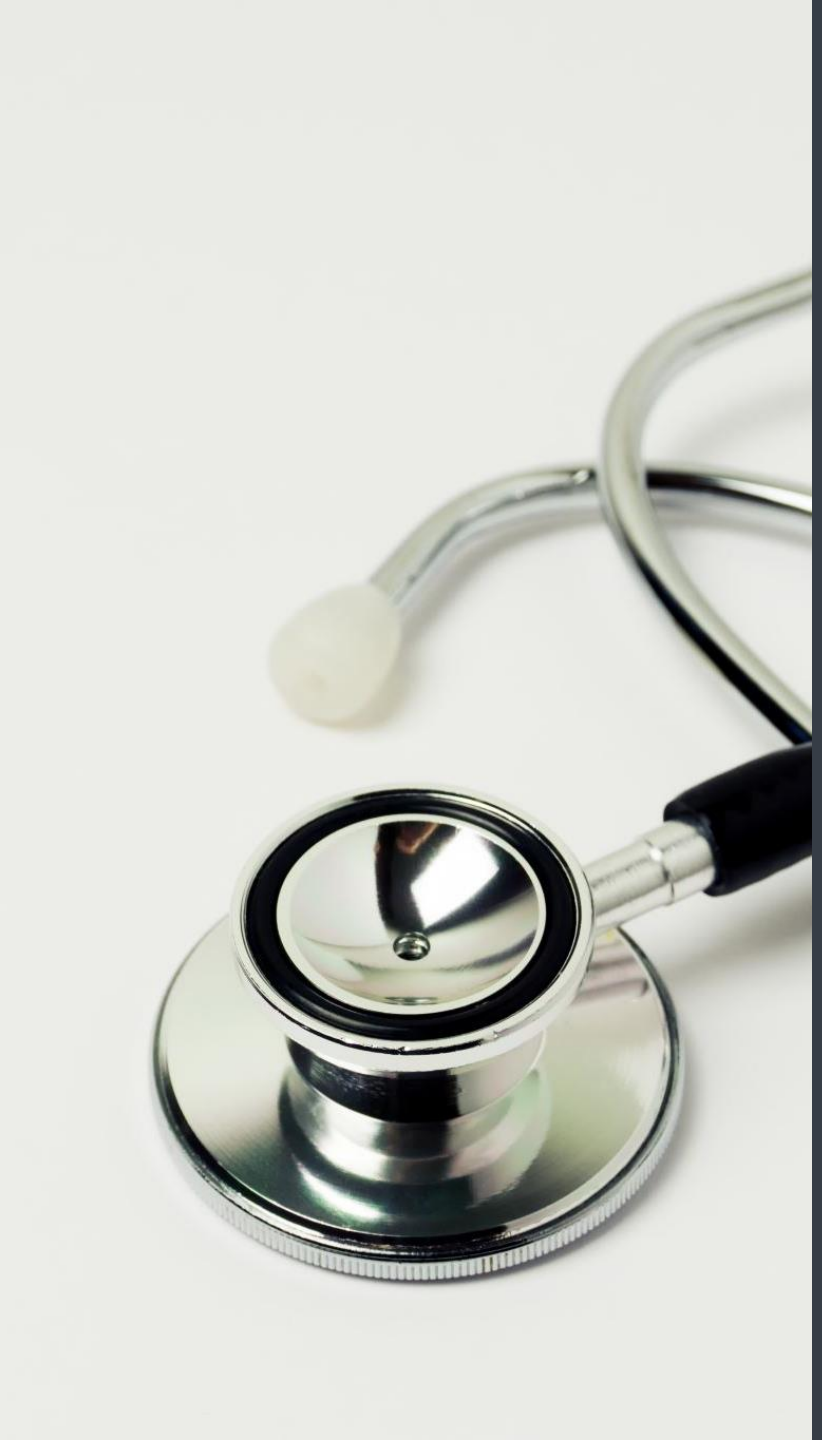


HEART FAILURE DETECTION

NAMRA PATEL



INTRODUCTION

Heart failure is a serious condition that affects millions of people worldwide. It is a leading cause of death and hospitalization. Early detection of heart failure is important for improving patient outcomes. The Heart Failure Prediction Dataset is a valuable resource for researchers who are developing new methods for detecting and managing heart failure.

OVERVIEW OF **CRISP-DM**

- CRISP-DM PROVIDES A STRUCTURED APPROACH TO THE ANALYSIS OF HEART FAILURE PREDICTION DATASETS:
- 1. BUSINESS UNDERSTANDING PHASE:- ANTICIPATE HEART FAILURE RISK, DETECT IT EARLY, AND REDUCE COSTS.
- 2. DATA UNDERSTANDING PHASE:- RELATIONSHIPS BETWEEN PATIENT INFORMATION, TARGET VARIABLES, MISSING VALUES, AND FEATURES.
- 3. DATA PREPARATION PHASE:- HANDLE MISSING VALUES, TRANSFORM FEATURES, AND CREATE NEW FEATURES AS NEEDED.
- 4. MODELING PHASE:- CHOOSE A SUITABLE ALGORITHM, TRAIN THE TRAINING DATA, EVALUATE THE VALIDATION DATA AND SELECT THE BEST MODEL.
- 5. EVALUATION PHASE:- INTEGRATE IT INTO CLINICAL APPLICATIONS TO EXPLAIN PREDICTIONS TO HEALTHCARE PROFESSIONALS.
- 6. DEPLOYMENT PHASE:- TRACK MODEL PERFORMANCE AND SHARE RESULTS WITH STAKEHOLDERS. TAKING THESE STEPS CAN HELP YOU EFFICIENTLY ANALYZE DATA SETS AND BUILD RELIABLE MODELS FOR PREDICTING HEART FAILURE RISK.

BUSINESS UNDERSTANDING

Who uses it? Doctor, patient or insurance company? What are their priorities?

What are your business goals? Cost reduction, early detection, targeting high-risk patients?

How is success measured? Lower costs, better results or wider market acceptance?

From these data, what factors are important in predicting heart disease? Age, lifestyle or genetics?

How does the model integrate into medical workflows and how does it ensure patient confidentiality? What are the ethical considerations when using this data to make predictions?

DATA UNDERSTANDING

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0

Statistics:
               Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.396514   198.799564    0.233115  136.809368
std      9.432617   18.514154   109.384145    0.423046   25.460334
min     28.000000    0.000000     0.000000    0.000000   60.000000
25%     47.000000  120.000000   173.250000    0.000000  120.000000
50%     54.000000  130.000000   223.000000    0.000000  138.000000
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000
...
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

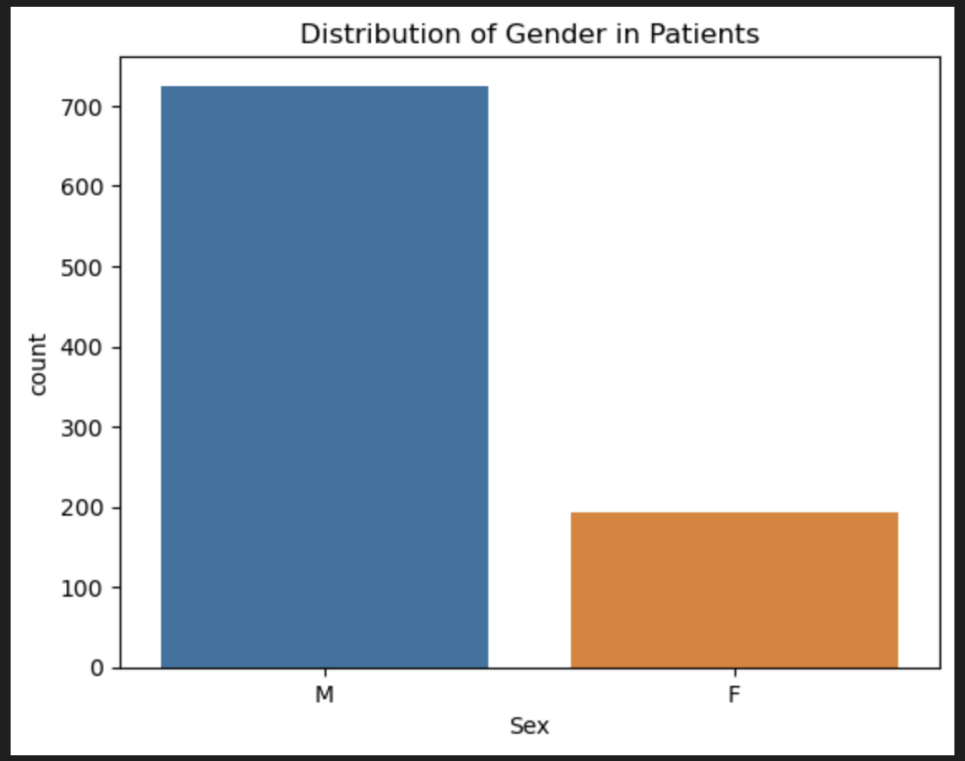
This heart failure prediction dataset from Kaggle combines data from five sources to provide 919 data points with 11 features to predict the presence or absence of heart failure. Characteristics range from demographics to medical history, lifestyle habits and blood tests. It is well refined and frequently used in heart failure prediction studies using machine learning.

Data Preparation and Analysis

- Handle missing values (e.g., impute with mean/median).
- Encode categorical features (e.g., one-hot encoding for "sex").
- Standardize numerical features (e.g., z-score scaling).

EXPLORATORY DATA ANALYSIS

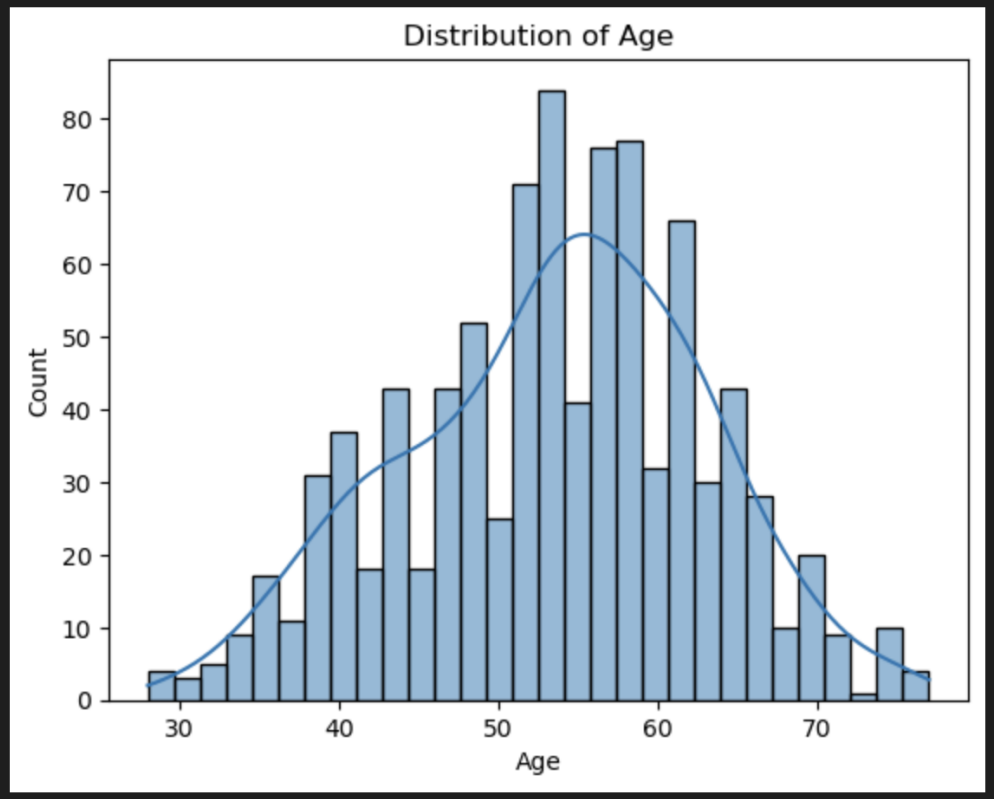
```
# Data Exploration
# Gender Distribution
sns.countplot(x='Sex', data=heart_data)
plt.title('Distribution of Gender in Patients')
plt.show()
```



Men are more likely to have heart diseases

The effect of age

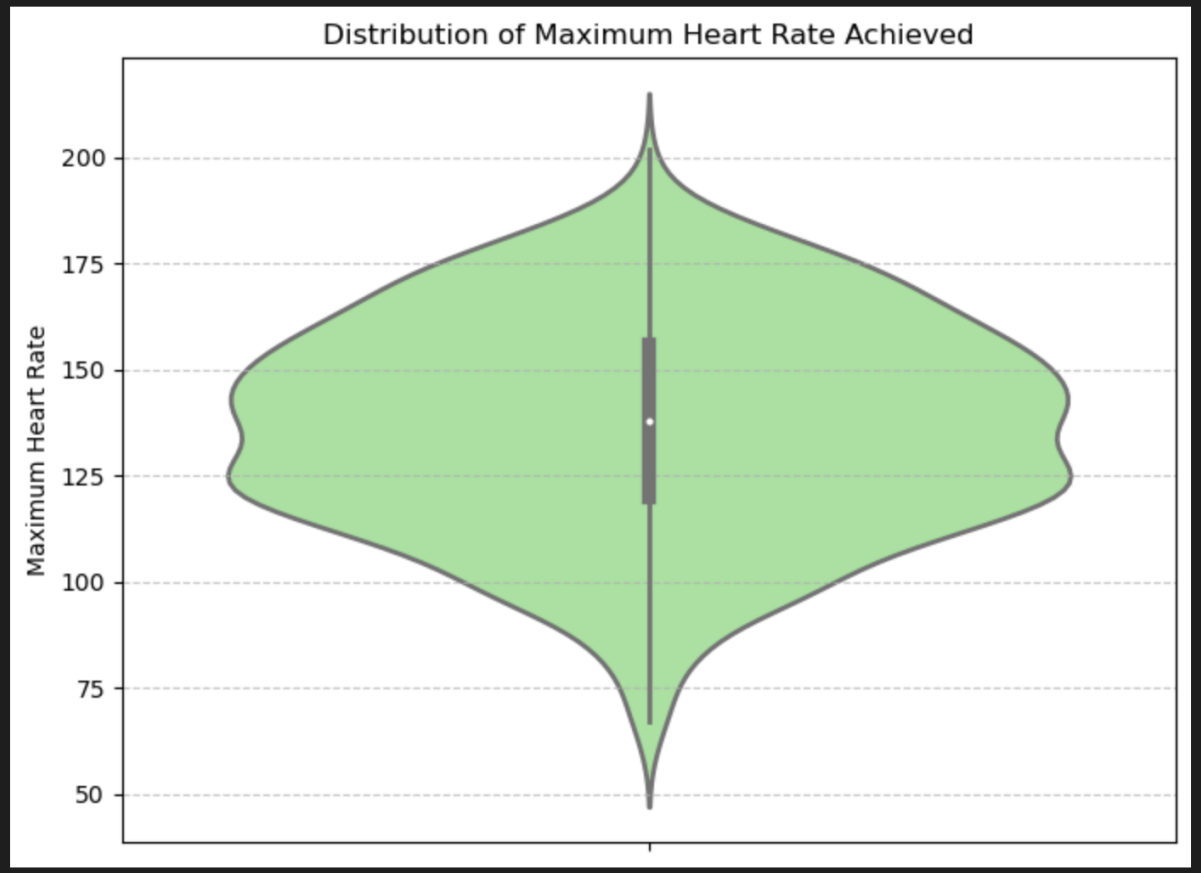
```
# Analyzing Age Effect
sns.histplot(heart_data['Age'], kde=True, bins=30)
plt.title('Distribution of Age')
plt.show()
```



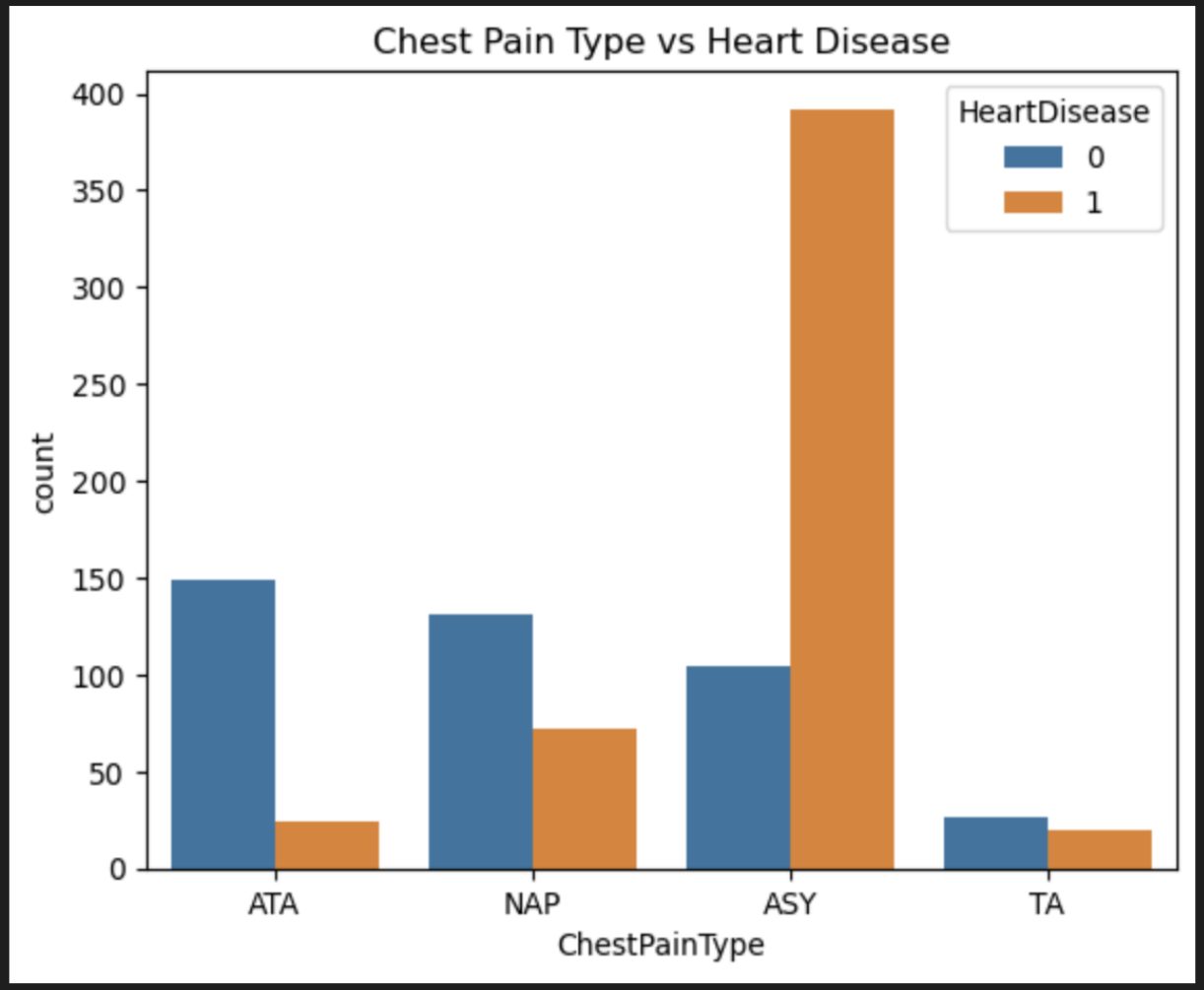
Adults age 50 and older are more likely to have heart disease

EXPLORATORY DATA ANALYSIS

```
plt.figure(figsize=(8, 6))
sns.violinplot(y=heart_data['MaxHR'], color='lightgreen', linewidth=2)
plt.title('Distribution of Maximum Heart Rate Achieved')
plt.ylabel('Maximum Heart Rate')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



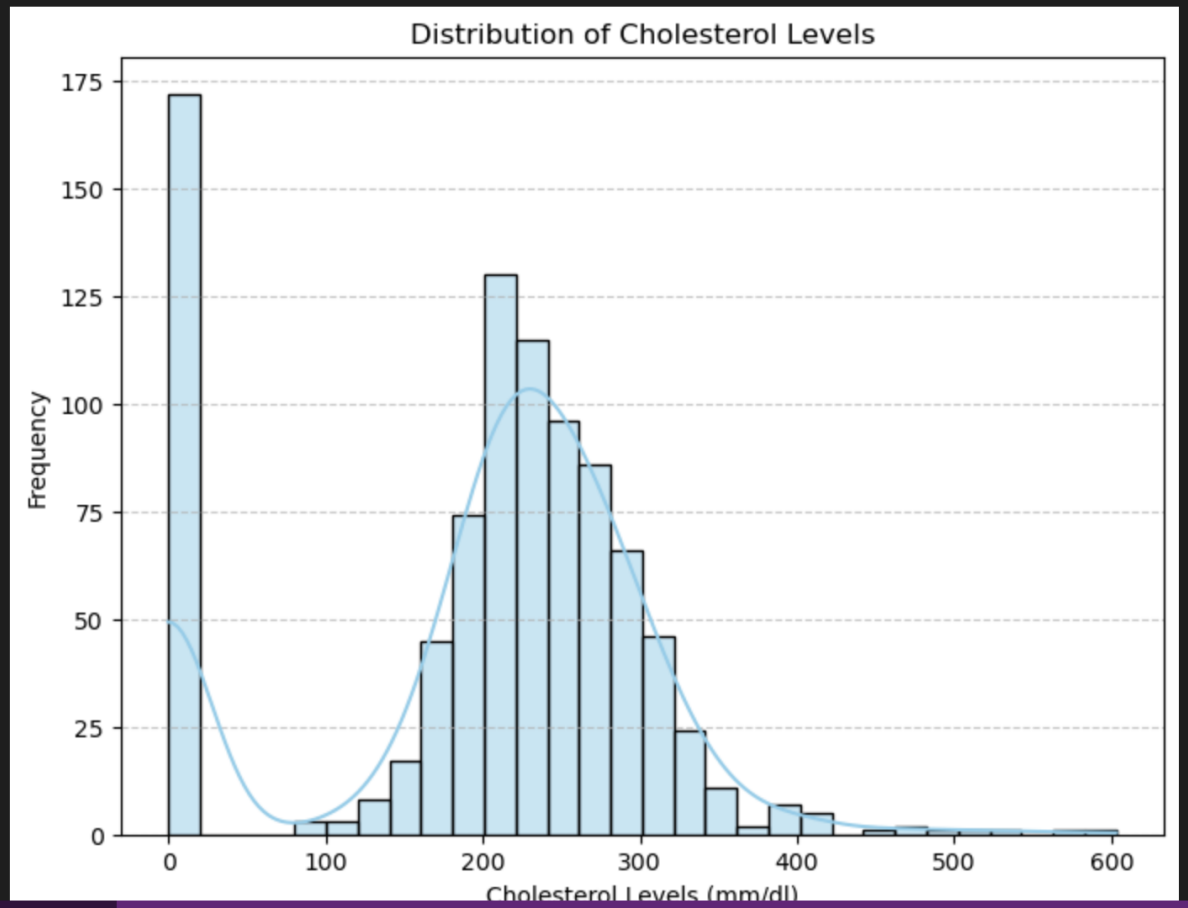

```
# Chest Pain Analysis
sns.countplot(x='ChestPainType', hue='HeartDisease', data=heart_data)
plt.title('Chest Pain Type vs Heart Disease')
plt.show()
```



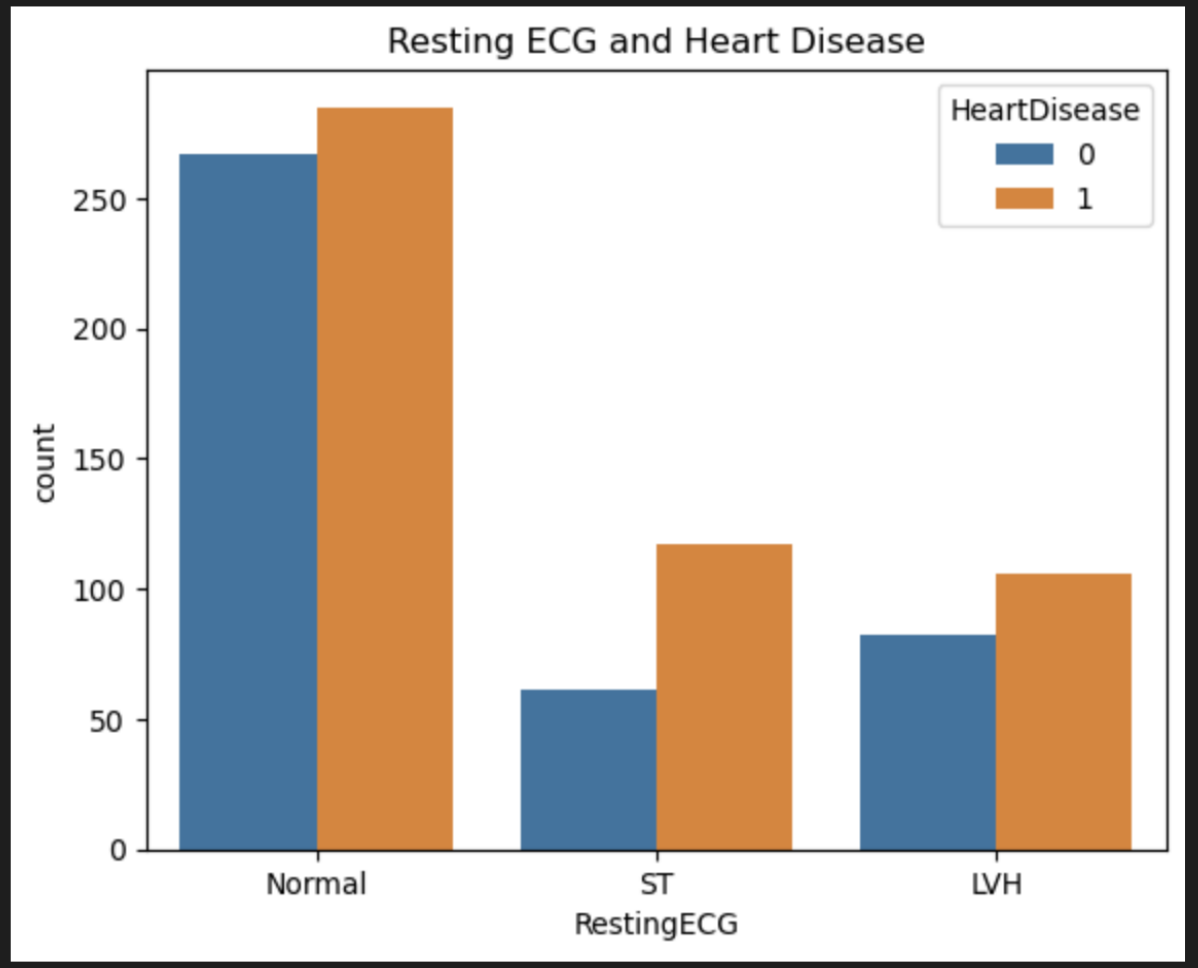
EXPLORATORY DATA ANALYSIS

```
plt.figure(figsize=(8, 6))
sns.histplot(heart_data['Cholesterol'], kde=True, bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Cholesterol Levels')
plt.xlabel('Cholesterol Levels (mm/dl)')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Loading...




```
# Resting ECG Analysis
sns.countplot(x='RestingECG', hue='HeartDisease', data=heart_data)
plt.title('Resting ECG and Heart Disease')
plt.show()
```



Data Normalization

```
# Normalizing the Data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model Evaluation Funct.

```
# Model Evaluation Function
def evaluate(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
    n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return np.mean(n_scores)
```

```
# Encoding categorical variables
categorical_vars = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
ct = ColumnTransformer([("encoder", OneHotEncoder(), categorical_vars)], remainder='passthrough')
heart_data_encoded = pd.DataFrame(ct.fit_transform(heart_data))
```

```
# Assigning Input and Output
X = heart_data_encoded.drop(columns=[heart_data.columns.get_loc("HeartDisease")])
y = heart_data['HeartDisease']
```

Data Splitting

```
# Splitting Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Pipeline for PCA and scaling
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95))
```

PREPARATION OF THE DATA AND ANALYSIS

LOGISTIC REGRESSION MODEL

Logistic Regression

```
# Logistic Regression hyperparameter tuning
param_grid_lr = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
grid_lr = GridSearchCV(LogisticRegression(max_iter=1000), param_grid_lr, cv=5)
grid_lr.fit(X_train, y_train)

# Best Logistic Regression model
logistic_model = grid_lr.best_estimator_

# Generating predictions for Logistic Regression
lr_predictions = logistic_model.predict(X_test)

# Evaluation metrics for Logistic Regression
print("Logistic Regression Accuracy:", accuracy_score(y_test, lr_predictions))
print("Classification Report for Logistic Regression:\n", classification_report(y_test, lr_predictions))
```

```
Logistic Regression Accuracy: 0.9166666666666666
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.88      0.91      0.89       107
           1       0.94      0.92      0.93       169

    accuracy                           0.92       276
   macro avg       0.91      0.91      0.91       276
weighted avg       0.92      0.92      0.92       276
```

This code snippet uses "GridSearchCV" to perform hyperparameter tuning for a logistic regression model. After finding the optimal tuning parameter “C” by cross-validation, the performance of the model is evaluated on the test set. Logistic regression was chosen because it is well suited to binary classification tasks, provides interpretation through likelihood ratios, is computationally efficient, and follows certain assumptions about the data. Evaluation metrics help you evaluate the accuracy of your model and provide detailed information about the classification.

RANDOM FOREST CLASSIFIER

Random Forest Classifier

```
# Random Forest with hyperparameter tuning
param_grid_rf = {'n_estimators': [100, 200, 300], 'max_depth': [10, 20, None]}
grid_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)
random_forest_model = grid_rf.best_estimator_

# Generating predictions for Random Forest
rf_predictions = random_forest_model.predict(X_test)
print("Random Forest Classifier Accuracy:", accuracy_score(y_test, rf_predictions))
print("Classification Report for Random Forest Classifier:\n", classification_report(y_test, rf_predictions))
```

```
Random Forest Classifier Accuracy: 0.9239130434782609
Classification Report for Random Forest Classifier:
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       107
           1       0.95      0.93      0.94       169

    accuracy                           0.92       276
   macro avg       0.92      0.92      0.92       276
weighted avg       0.92      0.92      0.92       276
```

This code uses `GridSearchCV` to tune the hyperparameters of the Random Forest classifier. To improve model performance, we explore combinations of "n_estimators" (number of trees) and "max_length" (maximum tree depth). The selected random forest classifier is suitable for this dataset due to its ensemble nature, which prevents overfitting, handles complex relationships, and provides insight into feature importance. The evaluation method evaluates the accuracy of the model and provides a detailed classification report, making Random Forest a comprehensive and reliable choice for this classification task.

SVC (SUPPORT VECTOR CLASSIFIER

SVC (C-Support Vector)

```
# SVC with hyperparameter tuning
param_grid_svc = {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto']}
grid_svc = GridSearchCV(SVC(), param_grid_svc, cv=5)
grid_svc.fit(X_train, y_train)
svc_model = grid_svc.best_estimator_

# Generating predictions for SVC
svc_predictions = svc_model.predict(X_test)

# Calculating accuracy and classification report
svc_accuracy = accuracy_score(y_test, svc_predictions)
svc_classification_report = classification_report(y_test, svc_predictions)

print("SVC (Support Vector Classifier) Accuracy:", svc_accuracy)
print("Classification Report for SVC (Support Vector Classifier):\n", svc_classification_report)
```

```
SVC (Support Vector Classifier) Accuracy: 0.9347826086956522
Classification Report for SVC (Support Vector Classifier):
              precision    recall  f1-score   support

           0       0.93      0.90      0.91       107
           1       0.94      0.96      0.95       169

    accuracy                           0.93       276
   macro avg       0.93      0.93      0.93       276
weighted avg       0.93      0.93      0.93       276
```

This code uses GridSearchCV to tune hyperparameters to optimize the SVC model for a classification task.
We explore the C and Gamma hyperparameters to find the best combination to maximize performance. Based on the cross-validation results, the best model is selected and used to make predictions from unseen data.
Detailed accuracy and classification reports are calculated to comprehensively evaluate model performance. This approach automates hyperparameter tuning for SVC, creating a potentially optimal model for complex classification problems.

CATBOOST CLASSIFIER

CatBoost Classifier

```
    # CatBoost Classifier with hyperparameter tuning
    param_grid_cb = {'iterations': [100, 200], 'depth': [4, 6, 10]}
    grid_cb = GridSearchCV(CatBoostClassifier(verbose=False), param_grid_cb, cv=5)
    grid_cb.fit(X_train, y_train)
    catboost_model = grid_cb.best_estimator_

    #Generating predictions for CatBoost
    catboost_predictions = catboost_model.predict(X_test)
    print("CatBoost Classifier Accuracy:", accuracy_score(y_test, catboost_predictions))
    print("Classification Report for CatBoost Classifier:\n", classification_report(y_test, catboost_predictions))
```

```
CatBoost Classifier Accuracy: 0.9239130434782609
Classification Report for CatBoost Classifier:
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       107
           1       0.95      0.93      0.94       169

    accuracy                           0.92       276
   macro avg       0.92      0.92      0.92       276
weighted avg       0.92      0.92      0.92       276
```

This code uses GridSearchCV and optimizes iteration and depth to find the best model for classification by five-fold cross-validation.
Accurate and detailed reporting (precision, recall, etc.) is performed for unseen data. It is efficient with category functions, strong performance and minimal preprocessing.

LGBM (LIGHT GRADIENT BOOSTING MACHINE)

LGBM Classifier (Light Gradient Boosting Machine)

```
# LGBM Classifier with hyperparameter tuning
param_grid_lgbm = {
    'num_leaves': [31, 50, 100],
    'learning_rate': [0.01, 0.1, 0.3],
    'n_estimators': [50, 100, 200]
}
grid_lgbm = GridSearchCV(lgb.LGBMClassifier(), param_grid_lgbm, cv=5)
grid_lgbm.fit(X_train, y_train)
lgbm_model = grid_lgbm.best_estimator_

# Generating predictions for LGBM
lgbm_predictions = lgbm_model.predict(X_test)
print("LGBM Classifier Accuracy:", accuracy_score(y_test, lgbm_predictions))
print("Classification Report for LGBM Classifier:\n", classification_report(y_test, lgbm_predictions))
```

```
[LightGBM] [Info] Number of positive: 270, number of negative: 242
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000247 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2223
[LightGBM] [Info] Number of data points in the train set: 512, number of used features: 13
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.527344 -> initscore=0.109484
[LightGBM] [Info] Start training from score 0.109484
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

GridSearchCV optimizes leaf_counts, learn_program, and n_estimators to find the best LGBM model using five-fold cross-validation.
The best models predict precision and obtain detailed reports (precision, recall, etc.) on unseen data. LGBM is fast and efficient on large datasets, handles categorical features well, and performs well in all scenarios.

COMPARISON OF THE MODELS

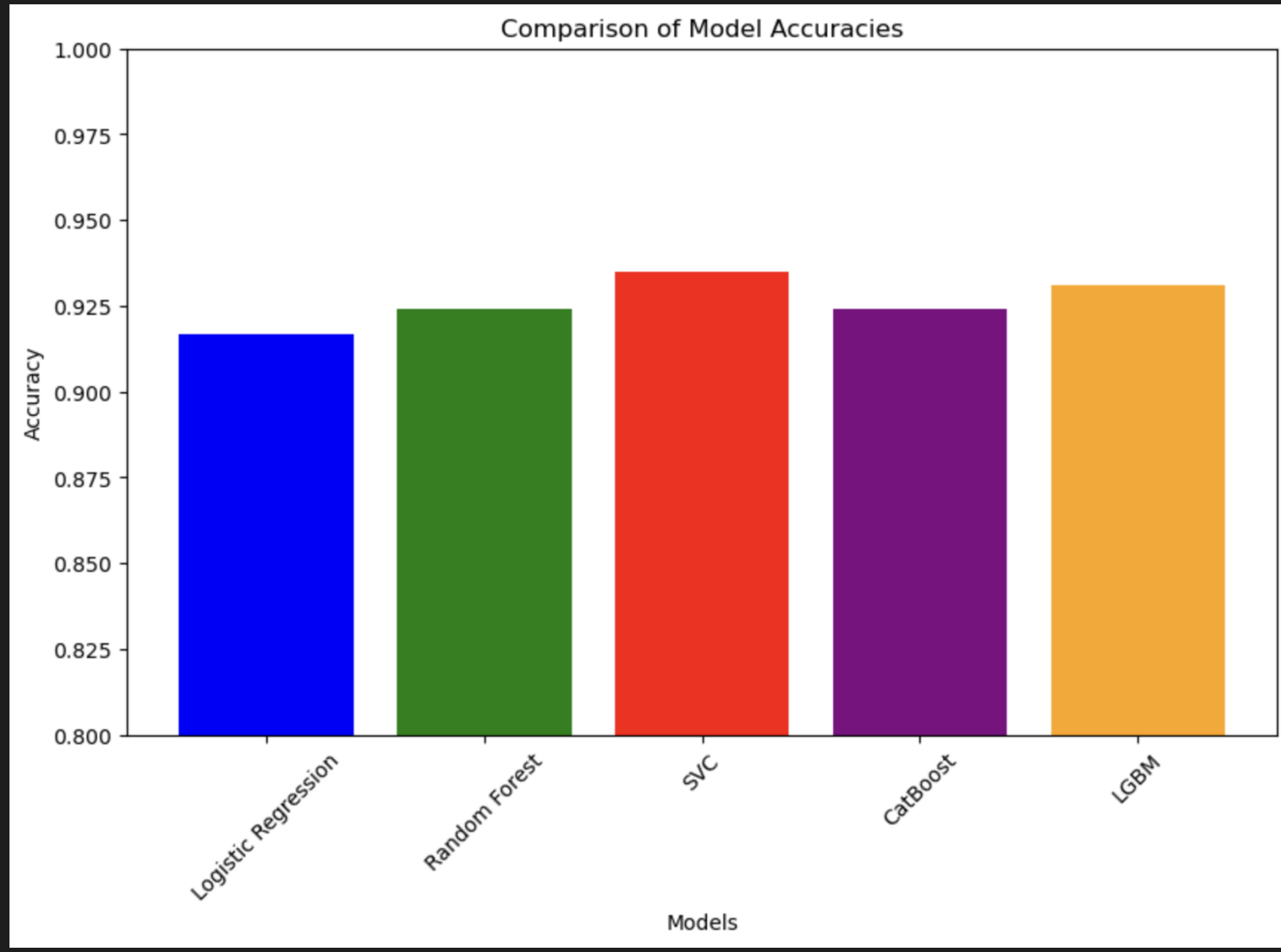
Comparision

Logistic Regression vs Random Forest vs SVC vs CatBoost vs LGBM

```
# Calculate accuracies for each model
accuracies = {
    "Logistic Regression": accuracy_score(y_test, lr_predictions),
    "Random Forest": accuracy_score(y_test, rf_predictions),
    "SVC": accuracy_score(y_test, svc_predictions),
    "CatBoost": accuracy_score(y_test, catboost_predictions),
    "LGBM": accuracy_score(y_test, lgbm_predictions)
}

# Creating a bar plot for model accuracies
plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(), accuracies.values(), color=['blue', 'green', 'red', 'purple', 'orange'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.ylim(0.8, 1.0)  # Adjust the y-axis limits as needed
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4],
 [Text(0, 0, 'Logistic Regression'),
  Text(1, 0, 'Random Forest'),
  Text(2, 0, 'SVC'),
  Text(3, 0, 'CatBoost'),
  Text(4, 0, 'LGBM')])
```



SVC Model got the highest accuracy of 93% among all the other models