Report Paper on UCI Heart Disease

Introduction:

This is a multivariate type of dataset which means providing or involving a variety of separate mathematical or statistical variables, multivariate numerical data analysis. It is composed of 14 attributes which are age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, oldpeak — ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels and Thalassemia. This database includes 76 attributes, but all published studies relate to the use of a subset of 14 of them. The Cleveland database is the only one used by ML researchers to date. One of the major tasks on this dataset is to predict based on the given attributes of a patient that whether that particular person has heart disease or not and other is the experimental task to diagnose and find out various insights from this dataset which could help in understanding the problem more.

Objective:

1. Understand the varieties of attributes related to heart disease.
2. Finding the most optimal solution in classifying and analyzing heart disease using statistical methods.
3. Gain insights into key factors contributing to the presence or absence of heart disease.

Data Source:

https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data

Dataset Columns:

1. id (Unique id for each patient)
2. age (Age of the patient in years)
3. origin (place of study)
4. sex (Male/Female)
5. cp chest pain type ([typical angina, atypical angina, non-anginal, asymptomatic])
6. trestbps resting blood pressure (resting blood pressure (in mm Hg on admission to the hospital))
7. chol (serum cholesterol in mg/dl)
8. fbs (if fasting blood sugar > 120 mg/dl)
9. restecg (resting electrocardiographic results)
10. Values: [normal, stt abnormality, lv hypertrophy]
11. thalch: maximum heart rate achieved
12. exang: exercise-induced angina (True/ False)
13. oldpeak: ST depression induced by exercise relative to rest
14. slope: the slope of the peak exercise ST segment
15. ca: number of major vessels (0-3) colored by fluoroscopy
16. thal: [normal; fixed defect; reversible defect]
17. num: the predicted attribute

Understanding Dataset:

Before doing any processing to the dataset, the data needs to be shuffled. This is a good practice for reasons:

1. Randomization of Data: To ensure that the order of samples does not introduce any biases into the learning process.
2. Avoiding Order Sensitivity: Some algorithms are sensitive to the order of data. The data can be biased in one direction.
3. Generalization: This helps in generalizing the model better. If the data is not shuffled, the model might memorize patterns related to the order of the data, which would not be desirable.
4. Reducing Variance in Mini-Batch Gradient Descent
5. Avoiding Overfitting: In cases where you are doing cross-validation, shuffling ensures that each fold gets a representative sample of the data.
6. Ensuring Fairness: Shuffling helps to ensure fairness in the learning process and prevents the model from learning spurious correlations.

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
178	179	43	Male	Cleveland	non-anginal	130.0	315.0	False	normal	162.0	False	1.9	upsloping	1.0	normal	0
585	586	47	Female	Hungary	asymptomatic	120.0	205.0	False	normal	98.0	True	2.0	flat	NaN	fixed defect	1
25	26	50	Female	Cleveland	non-anginal	120.0	219.0	False	normal	158.0	False	1.6	flat	0.0	normal	0
613	614	43	Male	Switzerland	asymptomatic	140.0	0.0	False	st-t abnormality	140.0	True	0.5	upsloping	NaN	reversable defect	2
876	877	64	Male	VA Long Beach	asymptomatic	150.0	193.0	False	st-t abnormality	135.0	True	0.5	flat	NaN	NaN	2

Check Missing Data:

	Missing Values	Percentage
id	0	0.000000
age	0	0.000000
sex	0	0.000000
dataset	0	0.000000
cp	0	0.000000
trestbps	59	6.413043
chol	30	3.260870
fbs	90	9.782609
restecg	2	0.217391
thalch	55	5.978261
exang	55	5.978261
oldpeak	62	6.739130
slope	309	33.586957
ca	611	66.413043
thal	486	52.826087
num	0	0.000000

Statistical Summary of Numerical Fields:

	id	age	trestbps	chol	thalch	oldpeak	ca	num
count	920.000000	920.000000	861.000000	890.000000	865.000000	858.000000	309.000000	920.000000
mean	460.500000	53.510870	132.132404	199.130337	137.545665	0.878788	0.676375	0.995652
std	265.725422	9.424685	19.066070	110.780810	25.926276	1.091226	0.935653	1.142693
min	1.000000	28.000000	0.000000	0.000000	60.000000	-2.600000	0.000000	0.000000
25%	230.750000	47.000000	120.000000	175.000000	120.000000	0.000000	0.000000	0.000000
50%	460.500000	54.000000	130.000000	223.000000	140.000000	0.500000	0.000000	1.000000
75%	690.250000	60.000000	140.000000	268.000000	157.000000	1.500000	1.000000	2.000000
max	920.000000	77.000000	200.000000	603.000000	202.000000	6.200000	3.000000	4.000000

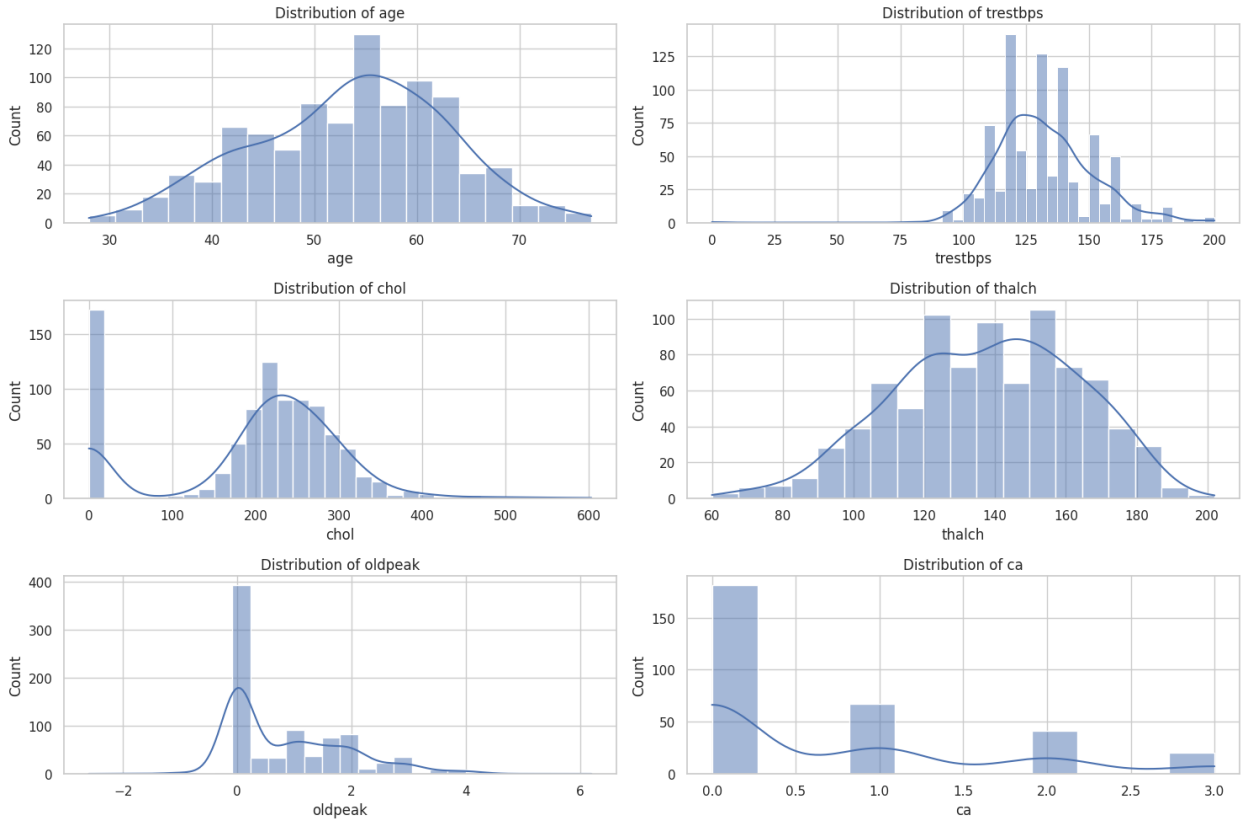
- 'age': The average age in the dataset is approximately 53.5 years with a standard deviation of 9.42 years, ranging from 28 to 77 years old.
- 'trestbps' (Resting Blood Pressure): The average resting blood pressure is around 132 mmHg. It's worth noting that there is a value of 0, which might indicate missing or incorrect data.
- 'chol' (Serum Cholesterol): The average cholesterol level is 199 mg/dl, with a very wide range, indicating there might be outliers or errors (a minimum value of 0 is not plausible for cholesterol levels).
- 'thalch' (Maximum Heart Rate Achieved): On average, the maximum heart rate recorded is around 138 bpm.
- 'oldpeak' (ST Depression): The average ST depression is about 0.88, but there are negative values, which might need further investigation since this is typically a non-negative measurement.
- 'ca' (Number of Major Vessels): On average, there is less than one major vessel observed per fluoroscopy.
- 'num' (Diagnosis of Heart Disease): This is the target variable, with values ranging from 0 (no presence of heart disease) to 4.

Distribution of Categorical Variables:

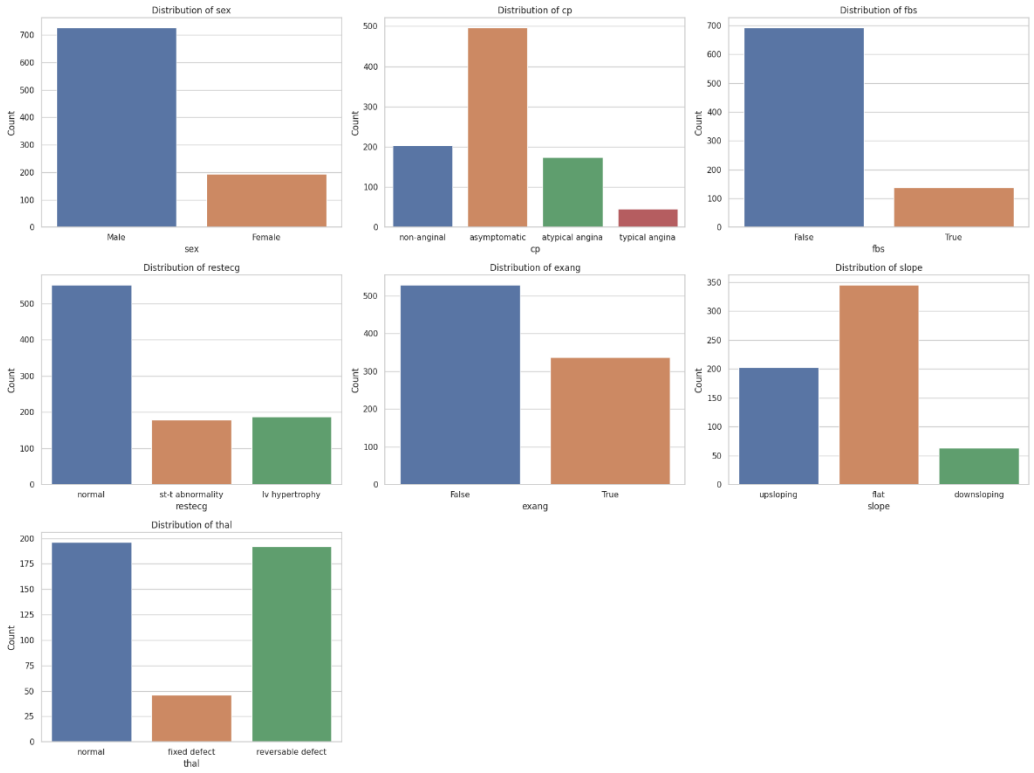
	Cleveland	Female	Hungary	Male	Switzerland	VA Long Beach	asymptomatic	atypical angina	non-anginal	typical angina	False	True	normal	lv hypertrophy	st-t abnormality	flat	upsloping	downsloping	reversable defect	fixed defect
sex	NaN	194.0	NaN	726.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
dataset	304.0	NaN	293.0	NaN	123.0	200.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
cp	NaN	NaN	NaN	NaN	NaN	NaN	496.0	174.0	204.0	46.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
fbs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	692.0	138.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
restecg	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	551.0	188.0	179.0	NaN	NaN	NaN	NaN	NaN
exang	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	528.0	337.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
slope	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	345.0	203.0	63.0	NaN	NaN
thal	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	196.0	NaN	NaN	NaN	NaN	NaN	192.0	46.0

- ‘sex’: There are 194 females and 726 males in the dataset.
- ‘dataset’: The data comes from multiple sources with the following counts - Cleveland: 304, Hungary: 293, Switzerland: 123, VA Long Beach: 200.
- ‘cp’ (Chest Pain Type): There are four types of chest pain recorded in the dataset with the following counts - asymptomatic: 496, atypical angina: 174, non-anginal: 204, typical angina: 46.
- ‘fbs’ (Fasting Blood Sugar): 138 individuals have fasting blood sugar greater than 120 mg/dl (True), and 692 do not (False).
- ‘restecg’ (Resting Electrocardiographic Results): The results are distributed as - normal: 551, LV hypertrophy: 188, ST-T abnormality: 179.
- ‘exang’ (Exercise-Induced Angina): 337 individuals experience exercise-induced angina (True), while 528 do not (False).
- ‘slope’ (Slope of the Peak Exercise ST Segment): The slopes are recorded as - flat: 345, upsloping: 203, downsloping: 63.
- ‘thal’ (Thalium Stress Test Result): The results are distributed as - reversable defect: 192, fixed defect: 46, normal: 19

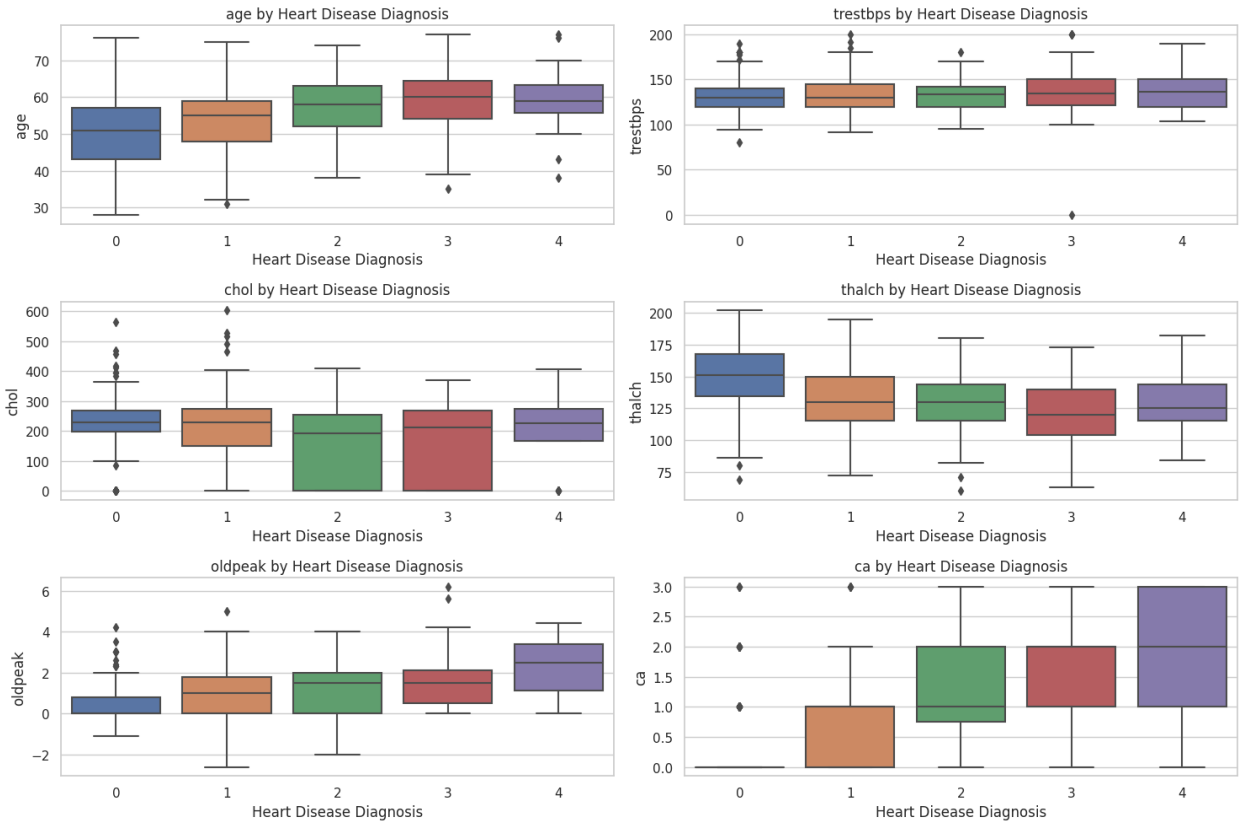
Exploratory Data Analysis (EDA):



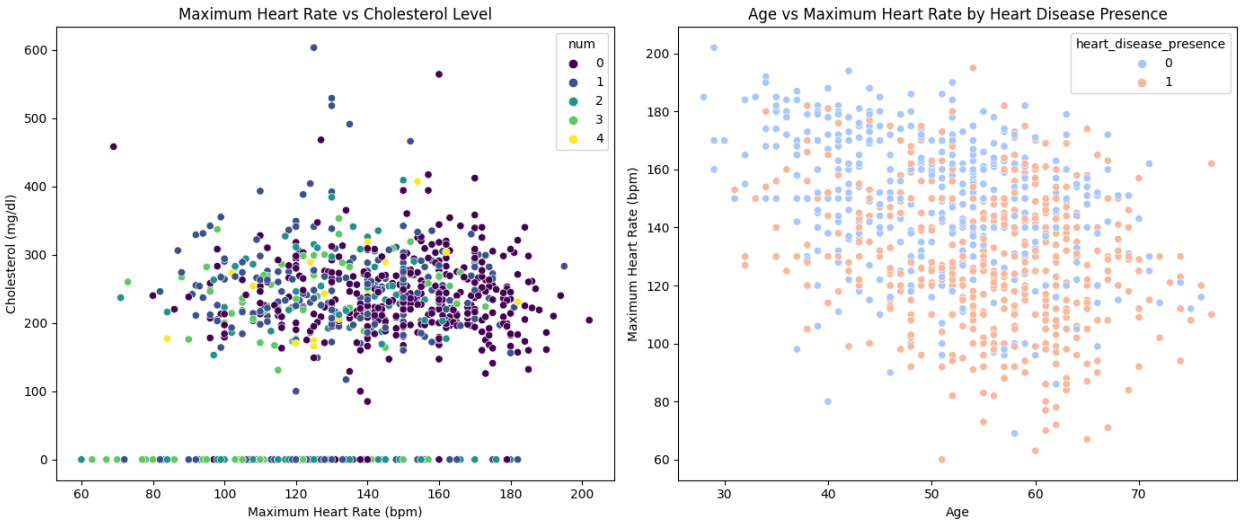
- 'age': The distribution is quite normal with a slight right skew, indicating a higher number of older individuals in the dataset.
- 'trestbps' (resting blood pressure): The distribution is roughly normal, but there are some outliers on the lower end (possibly incorrect entries as mentioned before).
- 'chol' (serum cholesterol): The distribution has a right skew, indicating that higher cholesterol levels are less common.
- 'thalch' (maximum heart rate achieved): The distribution is slightly left-skewed, with most individuals having a maximum heart rate between 120 and 160 bpm.
- 'oldpeak': Most values are clustered near 0 with a long tail to the right, which is typical for this kind of measurement.
- 'ca' (number of major vessels colored by fluoroscopy): The distribution is concentrated at 0, with fewer individuals having higher values.



- 'sex': There are more male participants than female in the dataset.
- 'cp' (chest pain type): The 'asymptomatic' category is the most frequent, followed by 'non-anginal pain' and 'atypical angina', with 'typical angina' being the least common.
- 'fbs' (fasting blood sugar): Most individuals have a fasting blood sugar below 120 mg/dl.
- 'restecg' (resting electrocardiographic results): The 'normal' category is the most common, followed by 'ST-T wave abnormality', and 'left ventricular hypertrophy' is the least common.
- 'exang' (exercise induced angina): Most individuals do not experience angina induced by exercise.
- 'slope' (the slope of the peak exercise ST segment): The 'flat' category is the most common, followed by 'upsloping', and 'downsloping'. However, many values are missing in this column.
- 'thal' (thalassemia): The 'normal' category is the most common, with 'reversable defect' and 'fixed defect' being less common. This column also has many missing values.

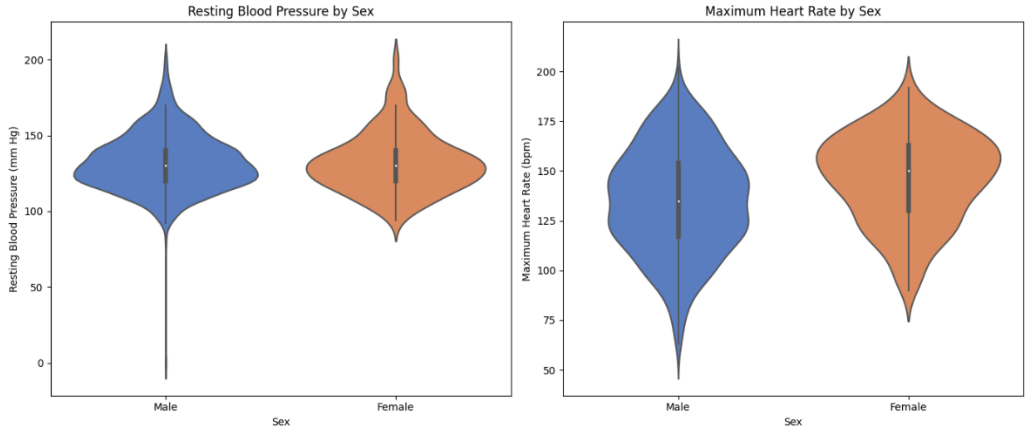


- 'age': Older age groups appear to have a higher median in the categories with a heart disease diagnosis (num > 0), suggesting a possible association between age and the presence of heart disease.
- 'trestbps' (resting blood pressure): There are some differences in median values across the heart disease diagnosis categories, with some outliers in each group.
- 'chol' (serum cholesterol): Similar to trestbps, there are variations in cholesterol levels across the heart disease diagnosis categories, with a number of outliers.
- 'thalch' (maximum heart rate achieved): Lower heart rates seem to be associated with higher categories of heart disease diagnosis.
- 'oldpeak': Higher values of oldpeak are more prevalent in individuals with a heart disease diagnosis, which aligns with the clinical expectation that ST depression is a sign of heart disease.
- 'ca' (number of major vessels colored by fluoroscopy): Individuals with a higher number of detectable vessels tend to have a higher heart disease diagnosis category.

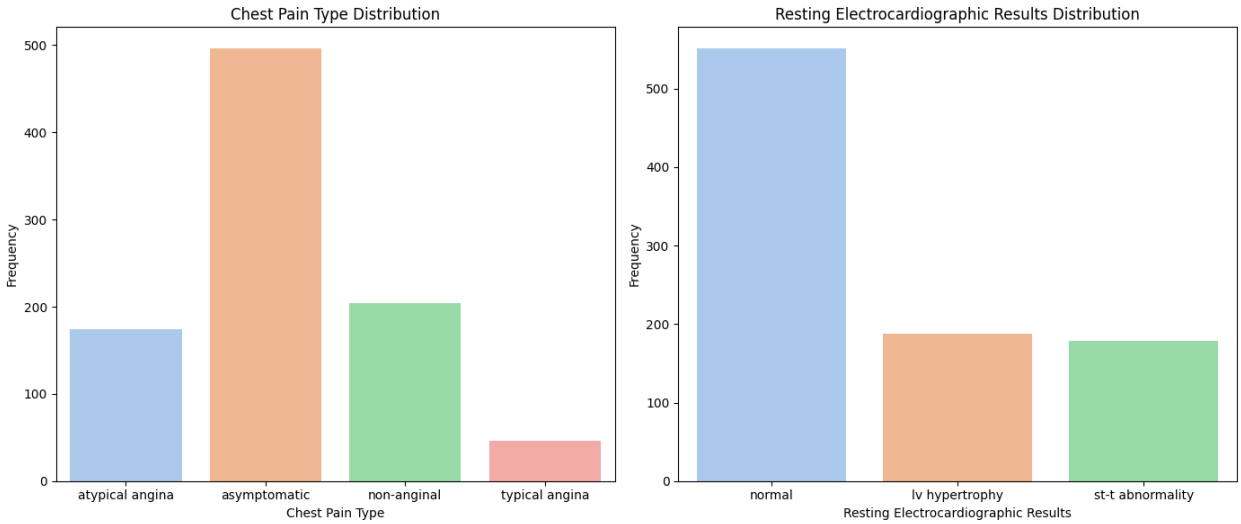


Maximum Heart Rate vs Cholesterol Level:

- There is no clear pattern or trend between maximum heart rate and cholesterol levels when considering the presence of heart disease. Individuals with and without heart disease are spread throughout the plot without distinct clustering.
- However, there's a visible concentration of individuals with lower maximum heart rate and higher cholesterol levels who have heart disease. Age vs Maximum Heart Rate by Heart Disease Presence:
- As age increases, the maximum heart rate achieved tends to decrease, which is a natural physiological trend.
- Individuals without heart disease (represented in blue) generally achieve higher maximum heart rates across all ages compared to those with heart disease (represented in red).
- There is a visible trend where older individuals with lower maximum heart rates tend to have a presence of heart disease.

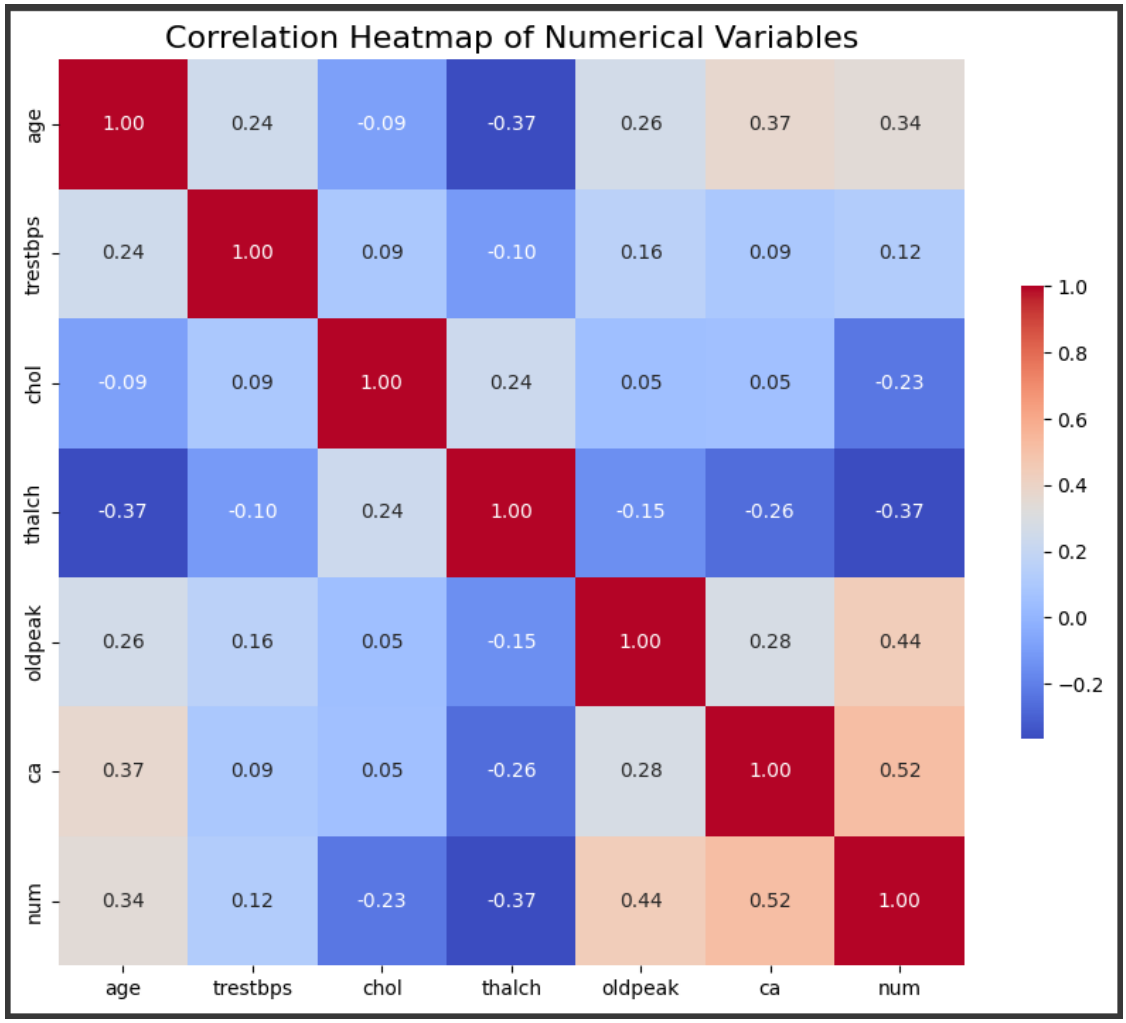


- Resting Blood Pressure by Sex: The distribution of resting blood pressure for males and females appears to be similar, with a slightly wider distribution for males, indicating more variability in their resting blood pressure.
- Maximum Heart Rate by Sex: There is a noticeable difference in the distribution of maximum heart rate between males and females. Females tend to have a higher maximum heart rate than males, which is consistent with general physiological differences between the sexes.



- Chest Pain Type (cp): The majority of observations are of the "asymptomatic" chest pain type, followed by "non-anginal pain", "atypical angina", and a few cases of "typical angina".
- Resting Electrocardiographic Results (restecg): The most common electrocardiographic result is "normal", with "hypertrophy" and "ST-T wave abnormality" being less frequent.

Correlation Between Variables:



From the correlation matrix:

- 'age' has a moderate positive correlation with num (the diagnosis of heart disease), this shows that older individuals may have a higher risk of heart disease.
- 'oldpeak' (ST depression) also shows a moderate positive correlation with num, suggesting that higher ST depression could be associated with heart disease.
- 'ca' (number of major vessels) has a strong positive correlation with num, which is a significant indicator as the presence of more vessels could be associated with a higher likelihood of heart disease.
- There are no very strong correlations between the other variables, indicating that no single factor is overwhelmingly dominant in predicting heart disease.

Preprocessing Steps:

Handling Missing Values:

We removed columns with a high percentage of missing values (‘ca’, ‘thal’, ‘slope’) as their absence could impact model accuracy and imputation may introduce bias. For the numerical columns, we used Median Imputation on columns 'trestbps', 'chol', 'thalch', 'oldpeak'. The reason of using median imputation is less sensitive to outliers than the mean. For categorical columns, we used Mode Imputation on columns 'fbs', 'restecg', 'exang', 'slope', 'thal'

```python
# List of numerical columns for median imputation
numerical_columns = ['trestbps', 'chol', 'thalch', 'oldpeak', 'ca']

# Applying median imputation to the numerical columns
for column in numerical_columns:
    median_value = df[column].median()
    df[column].fillna(median_value, inplace=True)

# List of categorical columns for mode imputation
categorical_columns = ['fbs', 'restecg', 'exang', 'slope', 'thal']

# Applying mode imputation to the categorical columns
for column in categorical_columns:
    mode_value = df[column].mode()[0]
    df[column].fillna(mode_value, inplace=True)

# Check if there are any missing values left
missing_values_after_imputation = df.isnull().sum()
missing_values_after_imputation[missing_values_after_imputation > 0]
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 920 entries, 178 to 868
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object
 8   restecg   918 non-null    object
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object
 13  ca        309 non-null    float64
 14  thal      434 non-null    object
 15  num       920 non-null    int64
dtypes: float64(5), int64(3), object(8)
memory usage: 122.2+ KB
```

Before Imputation

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 920 entries, 178 to 868
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  920 non-null    float64
 6   chol      920 non-null    float64
 7   fbs       920 non-null    bool
 8   restecg   920 non-null    object
 9   thalch    920 non-null    float64
 10  exang     920 non-null    bool
 11  oldpeak   920 non-null    float64
 12  slope     920 non-null    object
 13  ca        920 non-null    float64
 14  thal      920 non-null    object
 15  num       920 non-null    int64
dtypes: bool(2), float64(5), int64(3), object(6)
memory usage: 109.6+ KB
```

After Imputation

Data Transformation:

From what we observed, the columns 'num' contains the target as 0,1,2,3,4. However, for identifying, it is simply the presence of disease. So, we binarize it to such:

- 0 will represent patients without heart disease.
- 1 will represent patients with any level of heart disease (originally labeled as 1, 2, 3, or 4).

```
df['thal'].replace({'fixed defect':'fixed_defect' , 'reversable defect': 'reversable_defect' }, inplace =True)
df['cp'].replace({'typical angina':'typical_angina', 'atypical angina': 'atypical_angina' }, inplace =True)


data_tmp = df[['age','sex','cp', 'trestbps', 'chol', 'fbs',  'thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal']].copy()
data_tmp['target'] = ((df['num'] > 0)*1).copy()
data_tmp['sex'] = (df['sex'] == 'Male')*1
data_tmp['fbs'] = (df['fbs'])*1
data_tmp['exang'] = (df['exang'])*1

data_tmp.columns = ['age', 'sex', 'chest_pain_type', 'resting_blood_pressure',
                    'cholesterol', 'fasting_blood_sugar',
                    'max_heart_rate_achieved', 'exercise_induced_angina',
                    'st_depression', 'st_slope_type', 'num_major_vessels',
                    'thalassemia_type', 'target']
data_tmp.head(15)
```

The numerical features have been standardized, and one-hot encoding has been applied to the categorical features, with the first category dropped to avoid the dummy variable trap (which is a situation where independent variable are multicollinear). The column names of the dataset were renamed to make it easier to interpret.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 920 entries, 412 to 160
Data columns (total 20 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   age                                 920 non-null    int64
 1   sex                                 920 non-null    int64
 2   resting_blood_pressure              920 non-null    float64
 3   cholesterol                         920 non-null    float64
 4   fasting_blood_sugar                 920 non-null    int64
 5   max_heart_rate_achieved             920 non-null    float64
 6   exercise_induced_angina             920 non-null    int64
 7   st_depression                       920 non-null    float64
 8   num_major_vessels                   920 non-null    float64
 9   target                              920 non-null    int64
 10  chest_pain_type_asymptomatic        920 non-null    uint8
 11  chest_pain_type_atypical_angina     920 non-null    uint8
 12  chest_pain_type_non-anginal         920 non-null    uint8
 13  chest_pain_type_typical_angina      920 non-null    uint8
 14  st_slope_type_downsloping           920 non-null    uint8
 15  st_slope_type_flat                  920 non-null    uint8
 16  st_slope_type_upsloping             920 non-null    uint8
 17  thalassemia_type_fixed_defect       920 non-null    uint8
 18  thalassemia_type_normal             920 non-null    uint8
 19  thalassemia_type_reversable_defect  920 non-null    uint8
dtypes: float64(5), int64(5), uint8(10)
memory usage: 88.0 KB
```

Feature Engineering:

In the context of predicting heart disease, we might consider interactions between features that could be indicative of higher risk. For instance, combining age with cholesterol levels, blood pressure, and heart rate could highlight risk patterns that are not evident when considering these features independently.

Here is the new Features we came up with:

- **Blood Pressure Categories**: Create categories for blood pressure (e.g., 'normal', 'elevated', 'high') based on medical guidelines.
    - Normal: Systolic <120 and Diastolic < 80
    - Elevated: Systolic 120-129 and Diastolic < 80
    - Hypertension Stage 1: Systolic 130-139 or Diastolic 80-89
    - Hypertension Stage 2: Systolic >=140 or Diastolic >=90

```python
# Feature 1: Blood Pressure Categories
def categorize_blood_pressure(systolic_bp):
    if systolic_bp < 120:
        return 0  # Normal
    elif 120 <= systolic_bp < 130:
        return 1  # Elevated
    elif 130 <= systolic_bp < 140:
        return 2  # Hypertension Stage 1
    else:
        return 3  # Hypertension Stage 2

# Apply the function to the resting blood pressure column
df['blood_pressure_category'] = df['resting_blood_pressure'].apply(categorize_blood_pressure)
```

- **Cholesterol Ratio**: The ratio of total cholesterol to HDL cholesterol, which is a known risk factor for heart disease.

```python
# Feature 2: Cholesterol Ratio Binarized
def categorize_cholesterol(cholesterol_level):
    return 1 if cholesterol_level >= 240 else 0

# Apply the function to the cholesterol column
df['cholesterol_risk'] = df['cholesterol'].apply(categorize_cholesterol)
```

- **Heart Rate Pressure Product**: Multiply resting blood pressure by maximum heart rate, an index of myocardial oxygen consumption.

```python
# Feature 3: Heart Rate Pressure Product
df['heart_rate_pressure_product'] = df['resting_blood_pressure'] * df['max_heart_rate_achieved']
```

- **Combined Stress Test Features**: Combine the results of exercise-induced angina (‘exang’), ST depression (‘oldpeak’), and the slope of the peak exercise ST segment into a single risk score.

```
# Identify the columns for ST slope types
st_slope_columns = [col for col in df.columns if 'st_slope_type' in col]
st_slope_weights = {'st_slope_type_upsloping': 0, 'st_slope_type_flat': 1, 'st_slope_type_downsloping': 2}

# Feature 4: Combined Stress Test Features
df['combined_stress_test_score'] = (
    df['exercise_induced_angina'] +
    df['st_depression'] +
    sum(df[col] * weight for col, weight in st_slope_weights.items())
)
```

- **Interaction Terms**: Create interaction terms for features that might amplify each other's effects, such as age with ‘trestbps’ and ‘chol’.

```
# Feature 5: Interaction Terms
df['age_trestbps_interaction'] = df['age'] * df['resting_blood_pressure']
df['age_chol_interaction'] = df['age'] * df['cholesterol']
```

Here are the new features extracted from the available feature in the dataset:

cholesterol_risk	heart_rate_pressure_product	combined_stress_test_score	age_trestbps_interaction	age_chol_interaction
1	19440.0	1.1	6000.0	12200.0
0	23360.0	1.0	6720.0	6174.0
1	21528.0	0.0	6486.0	12079.0
0	19964.0	3.0	6944.0	12544.0
1	13920.0	4.0	7975.0	13640.0

Model Training:

The dataset is split into training and testing sets, with 644 samples in the training set and 276 samples in the testing set. The ratio of train/test is 70% and 30%.

In this session, we evaluated and compared 6 models from the machine learning techniques:

1. Decision Tree Classifier
2. Random Forest Classifier
3. XGB Classifier
4. LGBM Classifier
5. Extra Trees Classifier
6. AdaBoost Classifier
7. Logistic Regression

Every model is fine-tuning using ‘RandomizedSearchCV’ as it is often preferred when dealing with a large hyperparameter space or when the computational budget is limited. The key reasons why you might choose ‘RandomizedSearchCV’:

1. It can sample a specified number of candidates from a parameter space with a specified distribution.
2. ‘RandomizedSearchCV’ allows you to control the number of parameter combinations that are attempted.
3. It can often find a combination that is good enough without exhaustively searching the entire space.

Decision Tree

```
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 6],
    'max_features': [None, 'sqrt', 'log2']
}

dt_classifier = DecisionTreeClassifier(random_state=42)


random_search_dt = RandomizedSearchCV(
    estimator=dt_classifier,
    param_distributions=param_grid_dt,
    n_iter=100,
    cv=5,
    verbose=1,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)

random_search_dt.fit(X_train, y_train)
best_dt_classifier = random_search_dt.best_estimator_
y_pred_dt = best_dt_classifier.predict(X_test)

accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt, recall_dt, f1_score_dt, _ = precision_recall_fscore_support(y_test, y_pred_dt, average='binary')

performance_metrics_dt = {
    "Accuracy": accuracy_dt,
    "Precision": precision_dt,
    "Recall": recall_dt,
    "F1-score": f1_score_dt
}

performance_metrics_dt
```

Result

```
{'Accuracy': 0.7753623188405797,
 'Precision': 0.7861635220125787,
 'Recall': 0.8169934640522876,
 'F1-score': 0.8012820512820513}
```

Random Forest

```
param_grid_rf = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 6],
    'bootstrap': [True, False],
    'max_features': ['auto', 'sqrt', 'log2']
}

rf_classifier = RandomForestClassifier(random_state=42)
random_search_rf = RandomizedSearchCV(
    estimator=rf_classifier,
    param_distributions=param_grid_rf,
    n_iter=100,
    cv=5,
    verbose=1,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)

random_search_rf.fit(X_train, y_train)
best_rf_classifier = random_search_rf.best_estimator_
y_pred_rf = best_rf_classifier.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf, recall_rf, f1_score_rf, _ = precision_recall_fscore_support(y_test, y_pred_rf, average='binary')

performance_metrics_rf = {
    "Accuracy": accuracy_rf,
    "Precision": precision_rf,
    "Recall": recall_rf,
    "F1-score": f1_score_rf
}

performance_metrics_rf
```

Result

```
{'Accuracy': 0.8333333333333334,
 'Precision': 0.8282208588957055,
 'Recall': 0.8823529411764706,
 'F1-score': 0.8544303797468354}
```

XGB Classifier

```
param_grid_xgb = {
    'n_estimators': [100, 200, 500],
    'max_depth': [3, 4, 5, 6, 7],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.5, 1, 1.5, 2],
    'min_child_weight': [1, 5, 10],
    'reg_alpha': [0, 0.5, 1],
    'reg_lambda': [1, 1.5, 2, 3]
}

xgb_classifier = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

random_search_xgb = RandomizedSearchCV(
    estimator=xgb_classifier,
    param_distributions=param_grid_xgb,
    n_iter=100,
    cv=5,
    verbose=1,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)

random_search_xgb.fit(X_train, y_train)
best_xgb_classifier = random_search_xgb.best_estimator_
y_pred_xgb = best_xgb_classifier.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
precision_xgb, recall_xgb, f1_score_xgb, _ = precision_recall_fscore_support(y_test, y_pred_xgb, average='binary')

performance_metrics_xgb = {
    "Accuracy": accuracy_xgb,
    "Precision": precision_xgb,
    "Recall": recall_xgb,
    "F1-score": f1_score_xgb
}

performance_metrics_xgb
```

Result

```
{'Accuracy': 0.8188405797101449,
 'Precision': 0.8366013071895425,
 'Recall': 0.8366013071895425,
 'F1-score': 0.8366013071895425}
```

LGBM Classifier

```
param_grid_lgbm = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1],
    'max_depth': [10, 15],
    'num_leaves': [20, 31],
    'boosting_type': ['gbdt'],
}

lgbm_classifier = LGBMClassifier(random_state=42, n_jobs=-1)

random_search_lgbm = RandomizedSearchCV(
    estimator=lgbm_classifier,
    param_distributions=param_grid_lgbm,
    n_iter=10,
    cv=5,
    verbose=1,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)


fit_params={"early_stopping_rounds":30,
            "eval_set":[(X_test, y_test)],
            "verbose":0}

random_search_lgbm.fit(X_train, y_train)
best_lgbm_classifier = random_search_lgbm.best_estimator_

y_pred_lgbm = best_lgbm_classifier.predict(X_test)

accuracy_lgbm = accuracy_score(y_test, y_pred_lgbm)
precision_lgbm, recall_lgbm, f1_score_lgbm, _ = precision_recall_fscore_support(y_test, y_pred_lgbm, average='binary')

performance_metrics_lgbm = {
    "Accuracy": accuracy_lgbm,
    "Precision": precision_lgbm,
    "Recall": recall_lgbm,
    "F1-score": f1_score_lgbm
}

performance_metrics_lgbm
```

Result

```
{'Accuracy': 0.8297101449275363,
 'Precision': 0.8441558441558441,
 'Recall': 0.8496732026143791,
 'F1-score': 0.8469055374592832}
```

ExtraTrees

```python
param_grid_et = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 6],
    'bootstrap': [True, False],
    'max_features': ['auto', 'sqrt', 'log2'],
    'class_weight': [None, 'balanced', 'balanced_subsample']
}

et_classifier = ExtraTreesClassifier(random_state=42)
random_search_et = RandomizedSearchCV(
    estimator=et_classifier,
    param_distributions=param_grid_et,
    n_iter=100,
    cv=5,
    verbose=1,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)

random_search_et.fit(X_train, y_train)
best_et_classifier = random_search_et.best_estimator_
y_pred_et = best_et_classifier.predict(X_test)
accuracy_et = accuracy_score(y_test, y_pred_et)
precision_et, recall_et, f1_score_et, _ = precision_recall_fscore_support(y_test, y_pred_et, average='binary')

performance_metrics_et = {
    "Accuracy": accuracy_et,
    "Precision": precision_et,
    "Recall": recall_et,
    "F1-score": f1_score_et
}

performance_metrics_et
```

Result

```
{'Accuracy': 0.8297101449275363,
 'Precision': 0.8397435897435898,
 'Recall': 0.8562091503267973,
 'F1-score': 0.8478964401294498}
```

AdaBoost Classifier

```
param_grid_ab = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1, 0.3, 1.0],
    'base_estimator__max_depth': [1, 2, 3, 4, 5],
    'base_estimator__min_samples_split': [2, 5, 10, 20],
    'base_estimator__min_samples_leaf': [1, 2, 4, 6]
}

base_estimator = DecisionTreeClassifier(random_state=42)
ab_classifier = AdaBoostClassifier(base_estimator=base_estimator, random_state=42)
random_search_ab = RandomizedSearchCV(
    estimator=ab_classifier,
    param_distributions=param_grid_ab,
    n_iter=100,
    cv=5,
    verbose=1,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)

random_search_ab.fit(X_train, y_train)
best_ab_classifier = random_search_ab.best_estimator_
y_pred_ab = best_ab_classifier.predict(X_test)

accuracy_ab = accuracy_score(y_test, y_pred_ab)
precision_ab, recall_ab, f1_score_ab, _ = precision_recall_fscore_support(y_test, y_pred_ab, average='binary')

performance_metrics_ab = {
    "Accuracy": accuracy_ab,
    "Precision": precision_ab,
    "Recall": recall_ab,
    "F1-score": f1_score_ab
}

performance_metrics_ab
```

Result

```
{'Accuracy': 0.8188405797101449,
 'Precision': 0.8238993710691824,
 'Recall': 0.8562091503267973,
 'F1-score': 0.8397435897435896}
```

Logistic Regression

```python
param_grid_lr = {
    'C': np.logspace(-4, 4, 20),
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [100, 200, 300],
    'class_weight': [None, 'balanced']
}

lr_classifier = LogisticRegression(random_state=42)
random_search_lr = RandomizedSearchCV(
    estimator=lr_classifier,
    param_distributions=param_grid_lr,
    n_iter=100,
    cv=5,
    verbose=1,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)

random_search_lr.fit(X_train, y_train)
best_lr_classifier = random_search_lr.best_estimator_
y_pred_lr = best_lr_classifier.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr, recall_lr, f1_score_lr, _ = precision_recall_fscore_support(y_test, y_pred_lr, average='binary')

performance_metrics_lr = {
    "Accuracy": accuracy_lr,
    "Precision": precision_lr,
    "Recall": recall_lr,
    "F1-score": f1_score_lr
}

performance_metrics_lr
```

Result

```
{'Accuracy': 0.8405797101449275,
 'Precision': 0.8385093167701864,
 'Recall': 0.8823529411764706,
 'F1-score': 0.8598726114649682}
```

After completing all the model training. The results are as follows:

	Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
6	Logistic Regression	85.869565	88.118812	86.407767	87.254902
1	Random Forest	85.326087	87.254902	86.407767	86.829268
3	LightGBM	84.239130	85.576923	86.407767	85.990338
5	AdaBoost	84.239130	86.274510	85.436893	85.853659
2	XGBoost	82.065217	83.653846	84.466019	84.057971
4	Extra Trees	82.065217	88.043478	78.640777	83.076923
0	Decision Tree	78.804348	83.333333	77.669903	80.402010

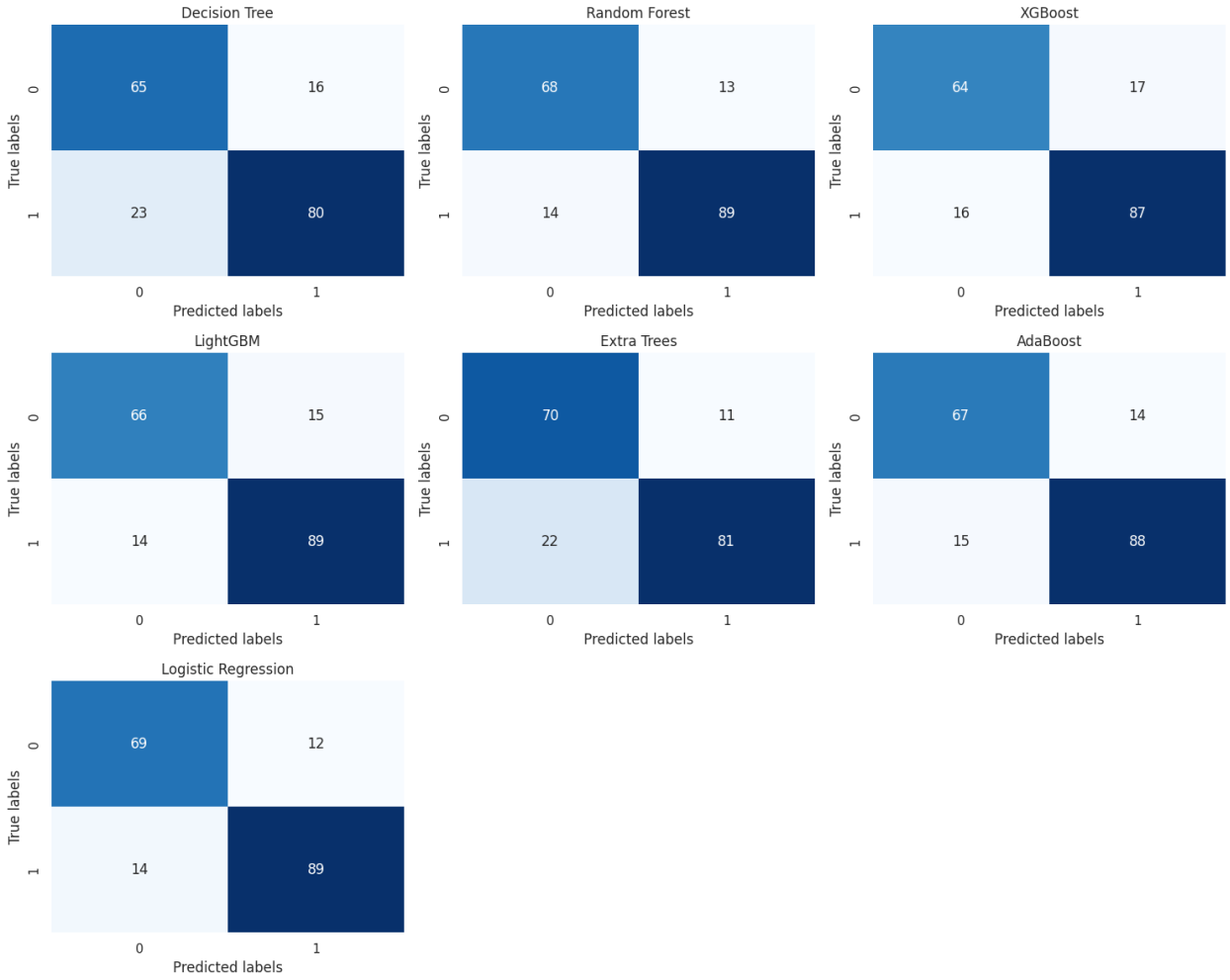
Logistic Regression has the highest accuracy of 85.87% followed by Random Forest of 85.33%. LGBM and LightGBM got the same accuracy of 84.24%, as well as AdaBoost of 84.24%. The lowest accuracy of all was Decision Tree with the score of 78.80%.

In terms of medical applications, the choice of evaluation metrics depends on the specific goals and characteristics of the problem you are trying to solve. Different metrics provide different insights into the performance of a model. In this case, accuracy, precision, and recall are used for the evaluation of detecting the patient with heart disease. With the highest precision from Logistic Regression of 88.12%, so out of all the instances predicted by the model as positive (indicating the presence of heart disease), approximately 88.12% are true positives, while the remaining 11.88% are false positives.

However, the highest recall is from Logistic Regression, Random Forest and LGBM of 86.41%, this means that the model is effective at capturing and identifying approximately 86.41% of individuals who truly have heart disease. The remaining 13.59% represent cases of heart disease that the model missed.

To summarize, to be able to apply the model to the real world applications, Logistic Regression is the most all-round model with the highest Accuracy, 3rd highest precision and the highest recall out of all models.

Confusion Matrix:



From what we observed, the confusion matrix showed the result of each model compared quite similar to each other. This concluded the fact that all the models used to predict the patient who was at risk of getting heart disease could be used to forecast the outcome depending on the given data accurately from the range of 78.80% to 85.87%.