# *Heart Disease Prediction On A Medical Dataset*

Sana Afreen, Zhiqi ZHU

11 December, 2024

# *Team Members*

Sana Afreen

Zhiqi ZHU

# *Outline*

1. Introduction (Context, datasource, etc)
2. Workflow & Methodology
3. Data Pre-Processing
4. Exploratory data analysis
5. Demographic Analysis
6. Performance evaluation metrics and parameters tunning methods
7. Machine learning algorithms
    - Supervised learning
    - Unsupervised learning

# *Introduction*

Heart disease is a leading cause of mortality globally, and early prediction is crucial for effective intervention and management. This dataset was designed to facilitate the development of algorithms that could **predict heart disease** based on **various clinical and demographic features**.

# *Dataset Source*

## Origin

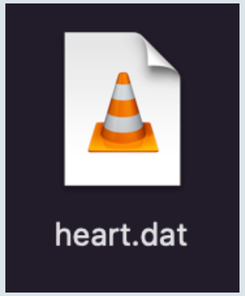
- Repository: UCI Machine Learning Repository

## Link

- The original dataset https://archive.ics.uci.edu/dataset/145/statlog+heart
- The generated dataset: https://www.openml.org/search?type=data&status=active&qualities.NumberOfInstances=gte_1000000&qualities.NumberOfFeatures=between_10_100&id=267&sort=runs

# *Original Dataset overview*

- Initially, the dataset we found was in .dat file
- The layout was not structured
- only 270 records
- Columns without names




This dataset is a heart disease database similar to a database already present in the repository (Heart Disease databases) but in a slightly different form

| Dataset Characteristics | Subject Area | Associated Tasks |
|---|---|---|
| Multivariate | Health and Medicine | Classification |
| **Feature Type** | **# Instances** | **# Features** |
| Categorical, Real | 270 | 13 |

```
1  70.0 1.0 4.0 130.0 322.0 0.0 2.0 109.0 0.0 2.4 2.0 3.0 3.0 2
2  67.0 0.0 3.0 115.0 564.0 0.0 2.0 160.0 0.0 1.6 2.0 0.0 7.0 1
3  57.0 1.0 2.0 124.0 261.0 0.0 0.0 141.0 0.0 0.3 1.0 0.0 7.0 2
4  64.0 1.0 4.0 128.0 263.0 0.0 0.0 105.0 1.0 0.2 2.0 1.0 7.0 1
5  74.0 0.0 2.0 120.0 269.0 0.0 2.0 121.0 1.0 0.2 1.0 1.0 3.0 1
6  65.0 1.0 4.0 120.0 177.0 0.0 0.0 140.0 0.0 0.4 1.0 0.0 7.0 1
7  56.0 1.0 3.0 130.0 256.0 1.0 2.0 142.0 1.0 0.6 2.0 1.0 6.0 2
8  59.0 1.0 4.0 110.0 239.0 0.0 2.0 142.0 1.0 1.2 2.0 1.0 7.0 2
9  60.0 1.0 4.0 140.0 293.0 0.0 2.0 170.0 0.0 1.2 2.0 2.0 7.0 2
10 63.0 0.0 4.0 150.0 407.0 0.0 2.0 154.0 0.0 4.0 2.0 3.0 7.0 2
11 59.0 1.0 4.0 135.0 234.0 0.0 0.0 161.0 0.0 0.5 2.0 0.0 7.0 1
12 53.0 1.0 4.0 142.0 226.0 0.0 2.0 111.0 1.0 0.0 1.0 0.0 7.0 1
13 44.0 1.0 3.0 140.0 235.0 0.0 2.0 180.0 0.0 0.0 1.0 0.0 3.0 1
14 61.0 1.0 1.0 134.0 234.0 0.0 0.0 145.0 0.0 2.6 2.0 2.0 3.0 2
15 57.0 0.0 4.0 128.0 303.0 0.0 2.0 159.0 0.0 0.0 1.0 1.0 3.0 1
16 71.0 0.0 4.0 112.0 149.0 0.0 0.0 125.0 0.0 1.6 2.0 0.0 3.0 1
17 46.0 1.0 4.0 140.0 311.0 0.0 0.0 120.0 1.0 1.8 2.0 2.0 7.0 2
18 53.0 1.0 4.0 140.0 203.0 1.0 2.0 155.0 1.0 3.1 3.0 0.0 7.0 2
19 64.0 1.0 1.0 110.0 211.0 0.0 2.0 144.0 1.0 1.8 2.0 0.0 3.0 1
20 40.0 1.0 1.0 140.0 199.0 0.0 0.0 178.0 1.0 1.4 1.0 0.0 7.0 1
21 67.0 1.0 4.0 120.0 229.0 0.0 2.0 129.0 1.0 2.6 2.0 2.0 7.0 2
22 48.0 1.0 2.0 130.0 245.0 0.0 2.0 180.0 0.0 0.2 2.0 0.0 3.0 1
23 43.0 1.0 4.0 115.0 303.0 0.0 0.0 181.0 0.0 1.2 2.0 0.0 3.0 1
24 47.0 1.0 4.0 112.0 204.0 0.0 0.0 143.0 0.0 0.1 1.0 0.0 3.0 1
25 54.0 0.0 2.0 132.0 288.0 1.0 2.0 159.0 1.0 0.0 1.0 1.0 3.0 1
26 48.0 0.0 3.0 130.0 275.0 0.0 0.0 139.0 0.0 0.2 1.0 0.0 3.0 1
27 46.0 0.0 4.0 138.0 243.0 0.0 2.0 152.0 1.0 0.0 2.0 0.0 3.0 1
28 51.0 0.0 3.0 120.0 295.0 0.0 2.0 157.0 0.0 0.6 1.0 0.0 3.0 1
```

# *Dataset Overview*

- Number of samples: **100 0000 samples** in total, generated to mimic the patterns in the original dataset.
- Number of features:  **13 feature variables** and 1 target variable.
- Target variable: **whether the patient has heart disease** (binary classification, 1 means the patient has heart disease, 0 means the patient does not have heart disease).

# *Feature Variables Overview*

| Feature | Description | Data Type |
|---|---|---|
| Age | Age of the patient (in years). | Integer |
| Sex | Gender of the patient (1 = Male, 0 = Female). | Binary (0 or 1) |
| Chest Pain Type | Type of chest pain experienced by the patient:<br>1 = Typical Angina,<br>2 = Atypical Angina,<br>3 = Non-Anginal Pain, | Categorical (1–4) |
| Resting Blood Pressure | Resting blood pressure in mm Hg at the time of h | Integer |
| Serum Cholesterol | Serum cholesterol level in mg/dl. | Integer |
| Fasting Blood Sugar | Fasting blood sugar > 120 mg/dl (1 = Yes, 0 = No) | Binary (0 or 1) |
| Resting ECG Results | Results of resting electrocardiographic tests:<br>0 = Normal,<br>1 = ST-T wave abnormality,<br>2 = Possible left ventricular hypertrophy by Estes' | Categorical (0–2) |

# *Feature Variables Overview*

| Feature | Description | Data Type |
|---|---|---|
| Maximum Heart Rate Achieved | Maximum heart rate achieved during exercise. | Integer |
| Exercise Induced Angina | Whether exercise induced angina (1 = Yes, 0 = No | Binary (0 or 1) |
| Oldpeak | ST depression induced by exercise relative to rest | Continuous (Float) |
| Slope | The slope of the peak exercise ST segment:<br>1 = Upsloping,<br>2 = Flat,<br>3 = Downsloping. | Categorical (1–3) |
| Number of Major Vessels Colored | Number of major vessels (0–3) colored by fluoros | Integer (0–3) |
| Thalassemia | Thalassemia condition:<br>3 = Normal,<br>6 = Fixed Defect,<br>7 = Reversible Defect. | Categorical (3, 6, 7) |
| Class | Target variable : heart disease is present on the patient or not | Binary (0 or 1) |

# *Workflow and Methodology*

## *Data Exploration with PySpark*

Link to my Code

### Connection to Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

# Workflow and Methodology

## Importing SparkSession & Initialization

- **Import** `SparkSession`:
  - Entry point for PySpark SQL and DataFrame APIs.
- **Terminate Existing** `SparkContext`:
  - Stops any active SparkContext to avoid conflicts with the new session.
- **Initialize** `SparkSession`:
  - Sets up a unified environment for distributed data processing.
- **Custom Configuration Parameters**:
  - `appName`: Defines the Spark application name (*Heart Disease Classification*).
  - `spark.executor.memory`: Allocates 4GB RAM for each executor process.
  - `spark.driver.memory`: Allocates 4GB RAM for the driver program.
  - `spark.executor.cores`: Assigns 2 CPU cores per executor for task parallelism.
  - `spark.default.parallelism`: Sets default number of partitions for parallel tasks (value: 4).
  - `spark.sql.shuffle.partitions`: Configures shuffle partitions for optimized SQL operations (value: 4).
- **Output**:
  - `SparkSession` initialized, ready for scalable data manipulation, query execution, and advanced analytics.

Link to my Code

Importing SparkSession & Initialization

```
from pyspark.sql import SparkSession  # Import SparkSession
try:
    sc.stop()
except:
    pass

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("Heart Disease Classification") \
    .config("spark.executor.memory", "4g") \
    .config("spark.driver.memory", "4g") \
    .config("spark.executor.cores", "2") \
    .config("spark.default.parallelism", "4") \
    .config("spark.sql.shuffle.partitions", "4") \
    .getOrCreate()

print("SparkSession Initialized:", spark)
```

```
SparkSession Initialized: <pyspark.sql.session.SparkSession object at 0x7d6f1b2397b0>
```

# Workflow and Methodology

## Data Importing & Visualization

Link to my Code

```
+---------+---+--------+--------------------+----------------+------------------+----------------------------------+-------------------------+---------------------+--------+-----+----------------------+----+--------------------+
|      age|sex|   chest|resting_blood_pressure|serum_cholestoral|fasting_blood_sugar|resting_electrocardiographic_results|maximum_heart_rate_achieved|exercise_induced_angina| oldpeak|slope|number_of_major_vessels|thal|               class|
+---------+---+--------+--------------------+----------------+------------------+----------------------------------+-------------------------+---------------------+--------+-----+----------------------+----+--------------------+
|53.494725|1.0|1.150395|           117.978412|       242.00937|               0.0|                               0.0|                133.361344|                  0.0|3.089391|  2.0|                   1.0| 3.0|[70 72 65 73 65 6...|
|37.320375|0.0|1.887693|            118.45567|      218.156844|               1.0|                               2.0|                148.458625|                  0.0|     0.0|  3.0|                   0.0| 3.0| [61 62 73 65 6E 74]|
|48.520214|1.0|     3.0|           141.819366|      173.382704|               0.0|                               2.0|                141.198191|                  0.0|1.071691|  2.0|                   0.0| 6.0| [61 62 73 65 6E 74]|
|59.587959|0.0|     4.0|           106.368725|      222.732859|               0.0|                               2.0|                141.659888|                  1.0|0.866638|  2.0|                   0.0| 7.0|[70 72 65 73 65 6...|
|58.805677|1.0|     3.0|           121.035286|      257.257441|               0.0|                               0.0|                145.333117|                  0.0|  1.2126|  3.0|                   0.0| 7.0| [61 62 73 65 6E 74]|
|68.956985|1.0|1.376265|            131.62802|      199.435235|               0.0|                               2.0|                150.496641|                  0.0| 1.65531|  1.0|                   1.0| 7.0|[70 72 65 73 65 6...|
|57.499464|0.0|  2.5611|           131.511388|      224.138569|               0.0|                               0.0|                176.408111|                  0.0|     0.0|  2.0|                   0.0| 3.0| [61 62 73 65 6E 74]|
|42.372611|0.0|     3.0|            107.34438|      266.412229|               0.0|                               2.0|                179.539938|                  0.0|0.704339|  1.0|                   2.0| 3.0| [61 62 73 65 6E 74]|
|45.068842|1.0|     3.0|           126.469256|      262.110165|               0.0|                               2.0|                187.588937|                  0.0|2.960573|  2.0|                   2.0| 3.0|[70 72 65 73 65 6...|
|43.779592|1.0|     4.0|           123.366212|       385.02484|               0.0|                               2.0|                124.521006|                  1.0|2.479594|  2.0|                   0.0| 7.0|[70 72 65 73 65 6...|
|51.462914|1.0|     4.0|           157.724738|      198.125241|               0.0|                               0.0|                135.630633|                  1.0|0.846988|  2.0|                   0.0| 3.0| [61 62 73 65 6E 74]|
|54.966228|1.0|     3.0|            154.04259|      221.608249|               0.0|                               2.0|                117.267713|                  0.0|1.522555|  2.0|                   1.0| 7.0|[70 72 65 73 65 6...|
|51.829753|1.0|     3.0|           160.381404|      275.841702|               0.0|                               0.0|                123.657456|                  0.0|1.039853|  2.0|                   0.0| 7.0| [61 62 73 65 6E 74]|
|54.114823|1.0|     4.0|           177.597933|      220.985428|               0.0|                               0.0|                120.563005|                  1.0|     0.0|  2.0|                   0.0| 6.0|[70 72 65 73 65 6...|
|51.901646|1.0| 1.33215|            114.61721|       236.92799|               0.0|                               2.0|                 189.32679|                  0.0|     0.0|  2.0|                   0.0| 7.0| [61 62 73 65 6E 74]|
| 54.89783|0.0|     3.0|           158.973951|      189.439548|               0.0|                               0.0|                119.707052|                  0.0|     0.0|  1.0|                   1.0| 7.0|[70 72 65 73 65 6...|
|53.373373|0.0|     4.0|           141.666238|      275.922381|               0.0|                               0.0|                165.346044|                  0.0|     0.0|  1.0|                   0.0| 3.0| [61 62 73 65 6E 74]|
|60.333476|1.0|     4.0|           128.184262|      240.488159|               1.0|                               2.0|                 105.31602|                  1.0|0.905643|  2.0|                   1.0| 7.0|[70 72 65 73 65 6...|
|54.797051|1.0|     4.0|           124.687273|      238.527548|               0.0|                               2.0|                112.708825|                  1.0|1.572746|  2.0|                   0.0| 7.0|[70 72 65 73 65 6...|
|51.911193|1.0|     3.0|           109.662857|      224.878564|               1.0|                               0.0|                174.538801|                  0.0|0.440702|  2.0|                   0.0| 3.0| [61 62 73 65 6E 74]|
+---------+---+--------+--------------------+----------------+------------------+----------------------------------+-------------------------+---------------------+--------+-----+----------------------+----+--------------------+
only showing top 20 rows
...
 |-- number_of_major_vessels: double (nullable = true)
 |-- thal: double (nullable = true)
 |-- class: binary (nullable = true)

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

# Data Pre-Processing

## Data Transformation

Link to my Code

```
# Step 3: Decode byte strings in the Pandas DataFrame
for col in df.select_dtypes([object]):  # Select columns with object dtype (potentially byte strings)
    df[col] = df[col].apply(lambda x: x.decode('utf-8') if isinstance(x, bytes) else x)

# Step 4: Convert Pandas DataFrame to PySpark DataFrame
spark_df = spark.createDataFrame(df)

# Step 5: Show the data (optional)
spark_df.show()
```

**Before**

```
+--------------------+
|               class|
+--------------------+
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
| [61 62 73 65 6E 74]|
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
| [61 62 73 65 6E 74]|
|[70 72 65 73 65 6...|
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
|[70 72 65 73 65 6...|
|[70 72 65 73 65 6...|
| [61 62 73 65 6E 74]|
+--------------------+
```

**After**

```
--------+-------+-------+
_vessels|tha    |  class|
--------+-------+-------+
     1.0|  3.0  |present|
     0.0|  6.0  | absent|
     0.0|  3.0  | absent|
     0.0|  7.0  |present|
     0.0|  7.0  | absent|
     1.0|  7.0  |present|
     0.0|  3.0  | absent|
     2.0|  3.0  | absent|
     2.0|  3.0  |present|
     0.0|  7.0  |present|
     0.0|  3.0  | absent|
     1.0|  7.0  |present|
     0.0|  7.0  | absent|
     0.0|  6.0  |present|
     0.0|  7.0  | absent|
     1.0|  7.0  |present|
     0.0|  3.0  | absent|
     1.0|  7.0  |present|
     0.0|  6.0  |present|
     0.0|  3.0  | absent|
--------+-------+-------+
```

# Data Pre-Processing

## Data Type to Understand the Data

Link to my Code

```
# Initialize Spark session
spark = SparkSession.builder.appName("Pandas to PySpark").getOrCreate()

# Assuming `df` is a Pandas DataFrame
pyspark_df = spark.createDataFrame(df)

# Now, access the schema
columns_info = [(field.name, field.dataType) for field in pyspark_df.schema]
for column, dtype in columns_info:
    print(f"Column: {column}, Data Type: {dtype}")
```

```
Column: age, Data Type: DoubleType()
Column: sex, Data Type: DoubleType()
Column: chest, Data Type: DoubleType()
Column: resting_blood_pressure, Data Type: DoubleType()
Column: serum_cholestoral, Data Type: DoubleType()
Column: fasting_blood_sugar, Data Type: DoubleType()
Column: resting_electrocardiographic_results, Data Type: DoubleType()
Column: maximum_heart_rate_achieved, Data Type: DoubleType()
Column: exercise_induced_angina, Data Type: DoubleType()
Column: oldpeak, Data Type: DoubleType()
Column: slope, Data Type: DoubleType()
Column: number_of_major_vessels, Data Type: DoubleType()
Column: thal, Data Type: DoubleType()
Column: class, Data Type: StringType()
```

# Data Pre-Processing

## Data Cleaning (Float to INT for categorical variable & Age)

Link to my Code

```
from pyspark.sql.functions import round as pyspark_round, col

# Round and convert following columns to integers in PySpark
spark_df = spark_df.withColumn("age", pyspark_round(col("age")).cast("int"))
spark_df = spark_df.withColumn("chest", pyspark_round(col("chest")).cast("int"))
spark_df = spark_df.withColumn("sex", pyspark_round(col("sex")).cast("int"))
spark_df = spark_df.withColumn("fasting_blood_sugar", pyspark_round(col("fasting_blood_sugar")).cast("int"))
spark_df = spark_df.withColumn("resting_electrocardiographic_results", pyspark_round(col("resting_electrocardiographic_results")).cast("int"))
spark_df = spark_df.withColumn("exercise_induced_angina", pyspark_round(col("exercise_induced_angina")).cast("int"))
spark_df = spark_df.withColumn("slope", pyspark_round(col("slope")).cast("int"))
spark_df = spark_df.withColumn("number_of_major_vessels", pyspark_round(col("number_of_major_vessels")).cast("int"))
spark_df = spark_df.withColumn("thal", pyspark_round(col("thal")).cast("int"))
# Show the updated DataFrame
spark_df.show()
```

**Before**

| age | sex | chest | resting_blood_pressure | serum_cholestoral | fasting_blood_sugar | r |
|---|---|---|---|---|---|---|
| 53.494725 | 1.0 | 1.150395 | 117.978412 | 242.00937 | 0.0 | |
| 37.320375 | 0.0 | 1.887693 | 118.45567 | 218.156844 | 1.0 | |
| 48.520214 | 1.0 | 3.0 | 141.819366 | 173.382704 | 0.0 | |
| 59.587959 | 0.0 | 4.0 | 106.368725 | 222.732859 | 0.0 | |
| 58.805677 | 1.0 | 3.0 | 121.035286 | 257.257441 | 0.0 | |
| 68.956985 | 1.0 | 1.376265 | 131.62802 | 199.435235 | 0.0 | |
| 57.499464 | 0.0 | 2.5611 | 131.511388 | 224.138569 | 0.0 | |
| 42.372611 | 0.0 | 3.0 | 107.34438 | 266.412229 | 0.0 | |
| 45.068842 | 1.0 | 3.0 | 126.469256 | 262.110165 | 0.0 | |

**After**

| age | sex | chest | resting_blood_pressure | serum_cholestoral | fasting_blood_sugar | r |
|---|---|---|---|---|---|---|
| 53 | 1 | 1 | 117.978412 | 242.00937 | 0 | |
| 37 | 0 | 2 | 118.45567 | 218.156844 | 1 | |
| 49 | 1 | 3 | 141.819366 | 173.382704 | 0 | |
| 60 | 0 | 4 | 106.368725 | 222.732859 | 0 | |
| 59 | 1 | 3 | 121.035286 | 257.257441 | 0 | |
| 69 | 1 | 1 | 131.62802 | 199.435235 | 0 | |
| 57 | 0 | 3 | 131.511388 | 224.138569 | 0 | |
| 42 | 0 | 3 | 107.34438 | 266.412229 | 0 | |
| 45 | 1 | 3 | 126.469256 | 262.110165 | 0 | |
| 44 | 1 | 4 | 123.366212 | 385.02484 | 0 | |
| 51 | 1 | 4 | 157.724738 | 198.125241 | 0 | |
| 55 | 1 | 3 | 154.04259 | 221.608249 | 0 | |
| 52 | 1 | 3 | 160.381404 | 275.841702 | 0 | |
| 54 | 1 | 4 | 177.597933 | 220.985428 | 0 | |
| 52 | 1 | 1 | 114.61721 | 236.92799 | 0 | |
| 55 | 0 | 3 | 158.973951 | 189.439548 | 0 | |
| 53 | 0 | 4 | 141.666238 | 275.922381 | 0 | |
| 60 | 1 | 4 | 128.184262 | 240.488159 | 1 | |
| 55 | 1 | 4 | 124.687273 | 238.527548 | 0 | |
| 52 | 1 | 3 | 109.662857 | 224.878564 | 1 | |

only showing top 20 rows

# Data Pre-Processing

## Data Encoding (For Class Column)

```
from pyspark.sql.functions import when, col

# Modify the 'class' column to have 1 for 'present' and 0 for 'absent'
spark_df = spark_df.withColumn("class", when(col("class") == "present", 1).otherwise(0))

# Show the updated DataFrame
spark_df.show()
```

```
+-------+
|  class|
+-------+
|present|
| absent|
| absent|
|present|
| absent|
|present|
| absent|
```

Link to my Code

```
+----------------------+----+-----+
|number_of_major_vessels|thal|class|
+----------------------+----+-----+
|                     1|   3|    1|
|                     0|   3|    0|
|                     0|   6|    0|
|                     0|   7|    1|
|                     0|   7|    0|
|                     1|   7|    1|
|                     0|   3|    0|
|                     2|   3|    0|
|                     2|   3|    1|
|                     0|   7|    1|
|                     0|   3|    0|
|                     1|   7|    1|
|                     0|   7|    0|
|                     0|   6|    1|
|                     0|   7|    0|
|                     1|   7|    1|
|                     0|   3|    0|
|                     1|   7|    1|
|                     0|   7|    1|
|                     0|   3|    0|
+----------------------+----+-----+
```

# *Key Tasks of Exploratory Data Analysis*

- Heart Disease by Age Distribution
- Heart Disease by Gender Analysis
- Combined Analysis: Age and Gender in Heart Disease
- Key Influencing Factors: Exercise, Blood Pressure, Sugar Levels, and More

# *Exploratory Data Analysis*

## *Importing Important Libraries*

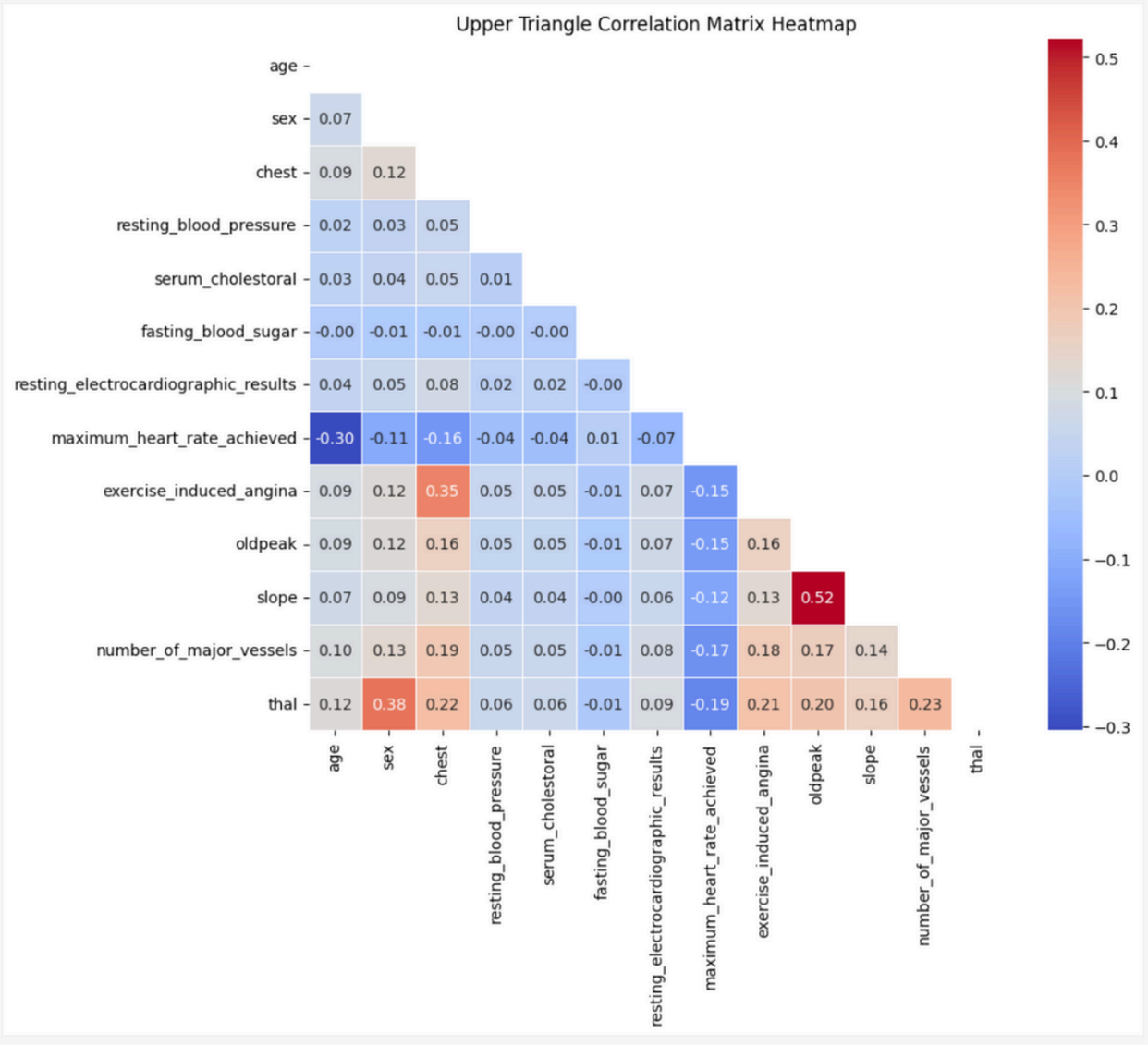
Link to my Code

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

# *Exploratory Data Analysis*

## *Correlation Metrics*

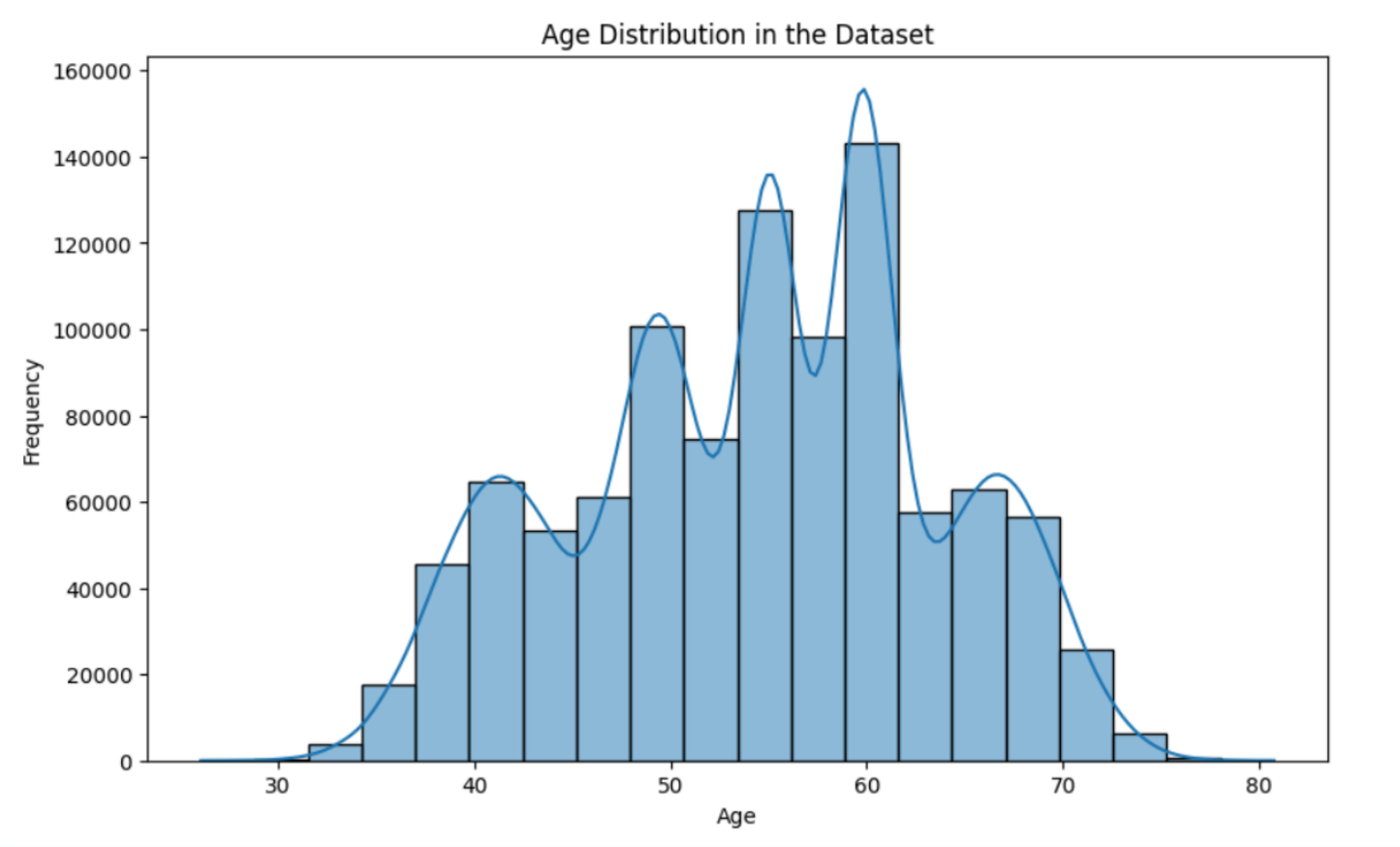
Link to my Code



# *Exploratory Data Analysis*

## *Distribution Analysis*

Link to my Code

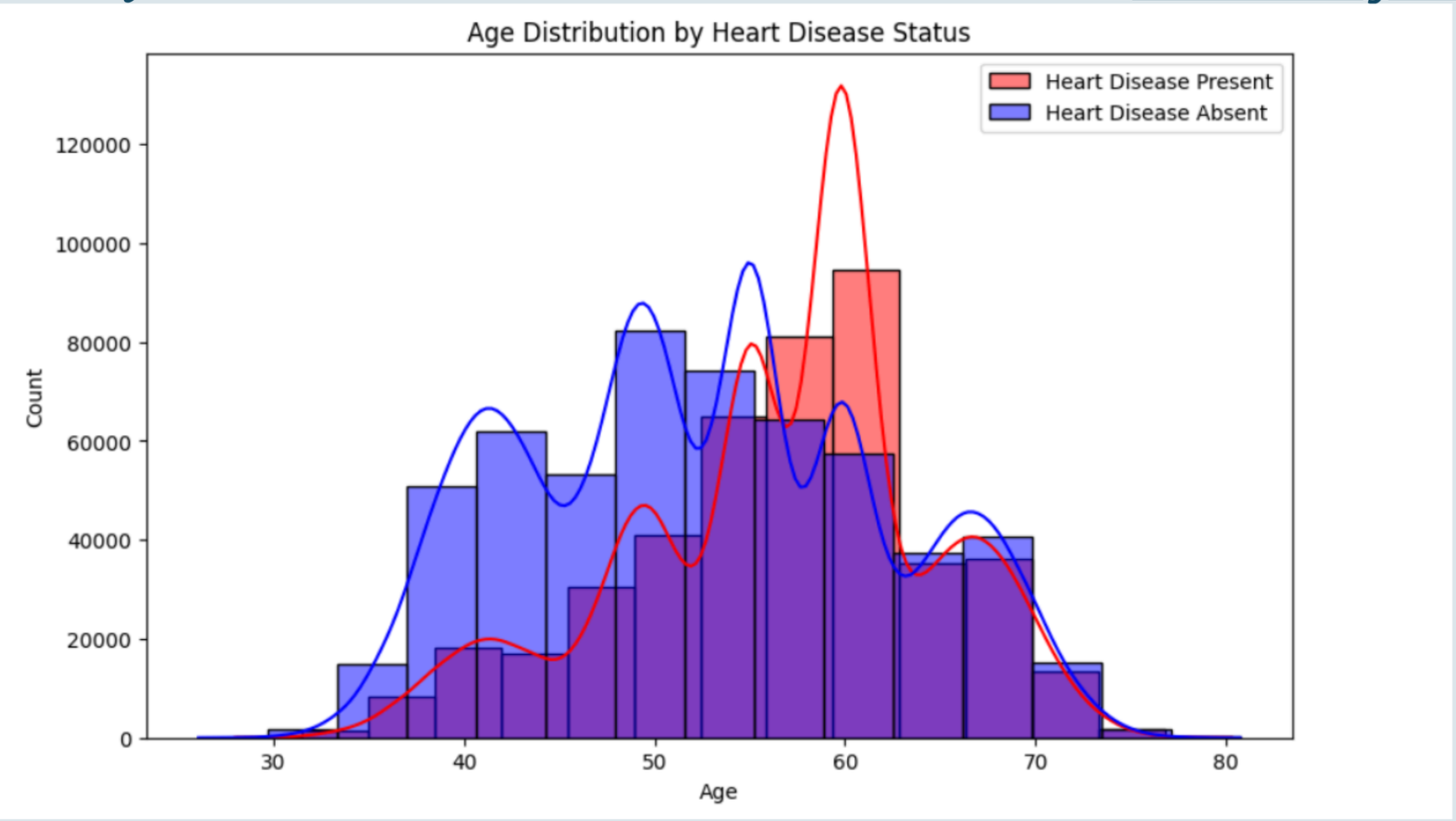



The target population of this data set is mainly middle-aged people and the elderly, which is a common range in the age group with high incidence of heart disease.

# Exploratory Data Analysis

## Distribution Analysis

Link to my Code

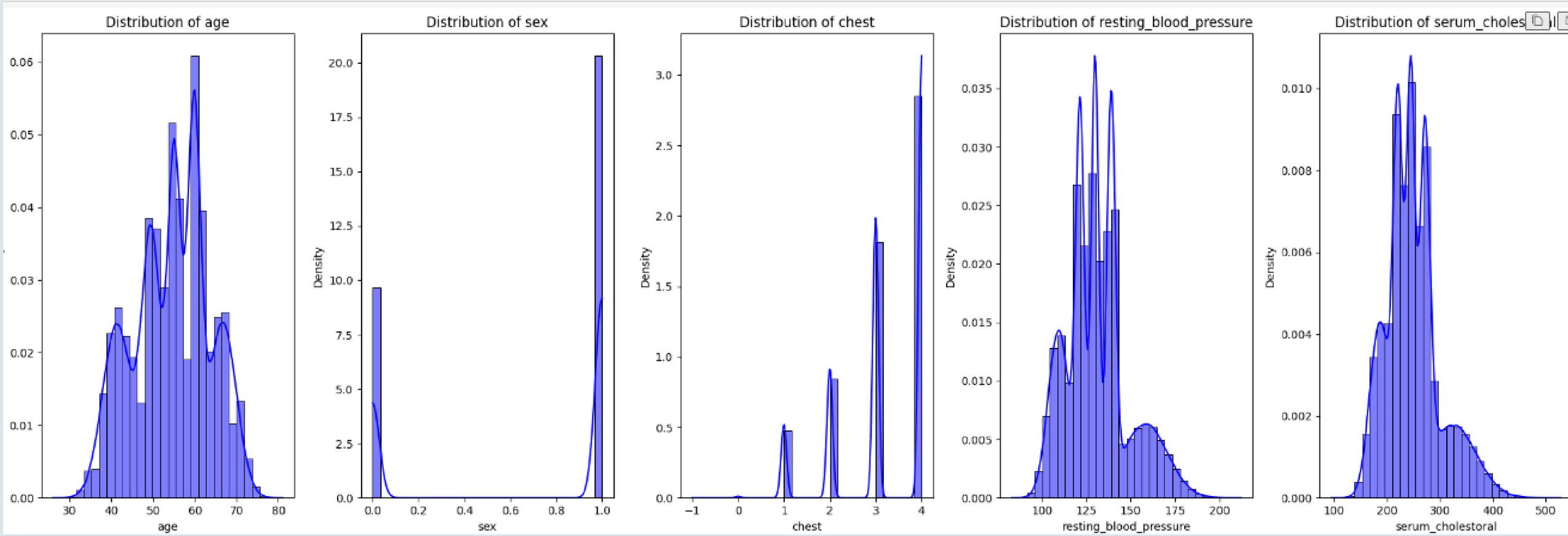



The density peak of the red distribution (people with heart disease) is centered around age 55 to 65. This shows that the highest number of people with heart disease is in this age group.

# *Exploratory Data Analysis*

## *Distribution Analysis*

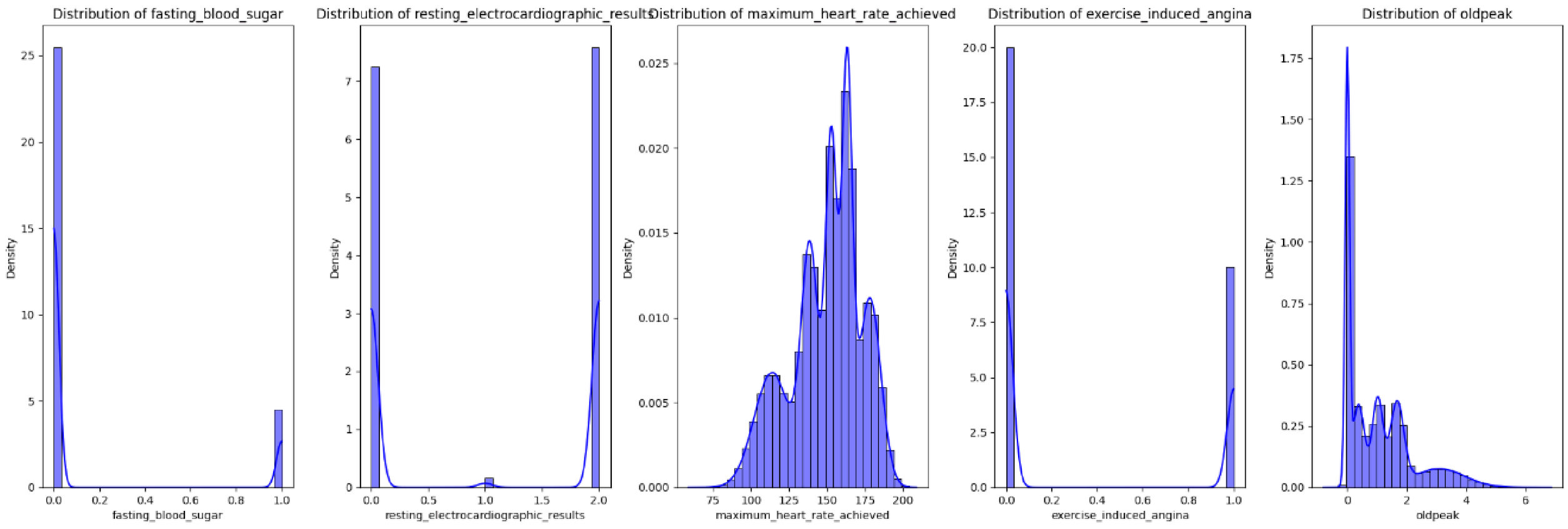
Link to my Code



# Exploratory Data Analysis

## Distribution Analysis

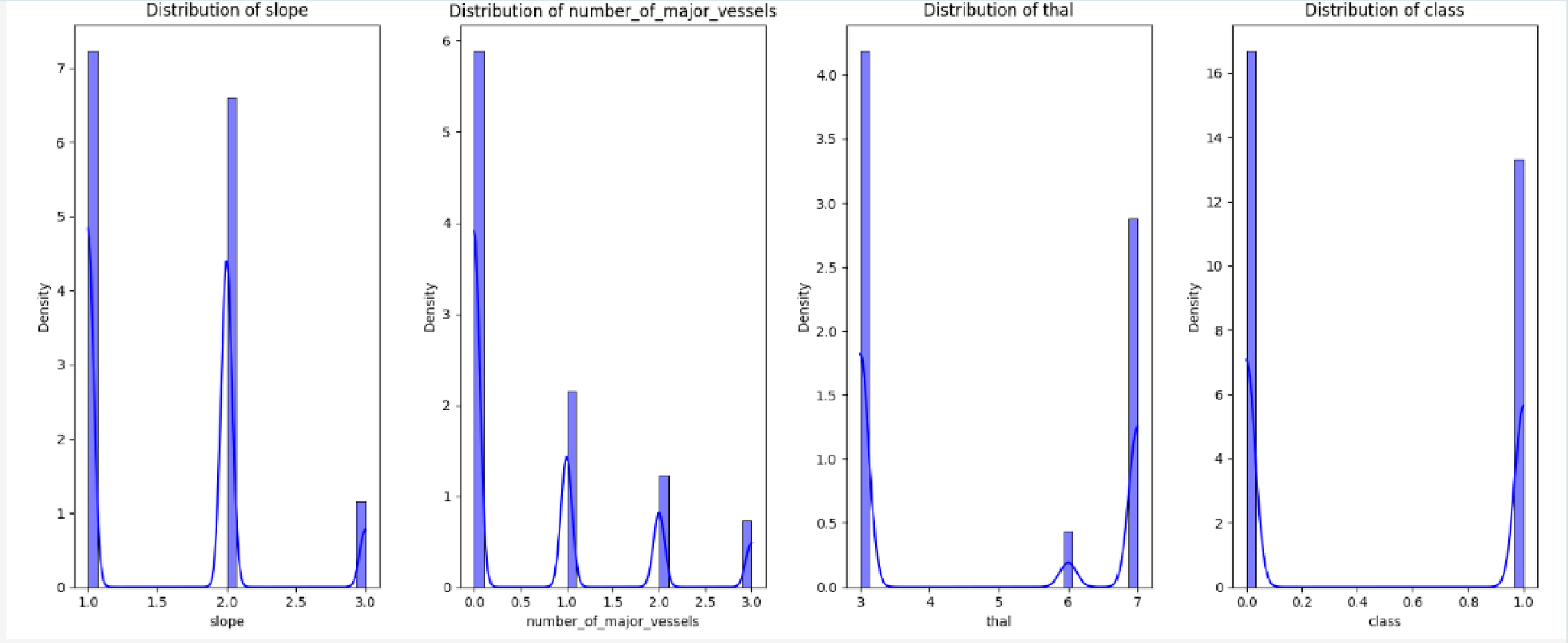
Link to my Code



# *Exploratory Data Analysis*

## *Distribution Analysis*

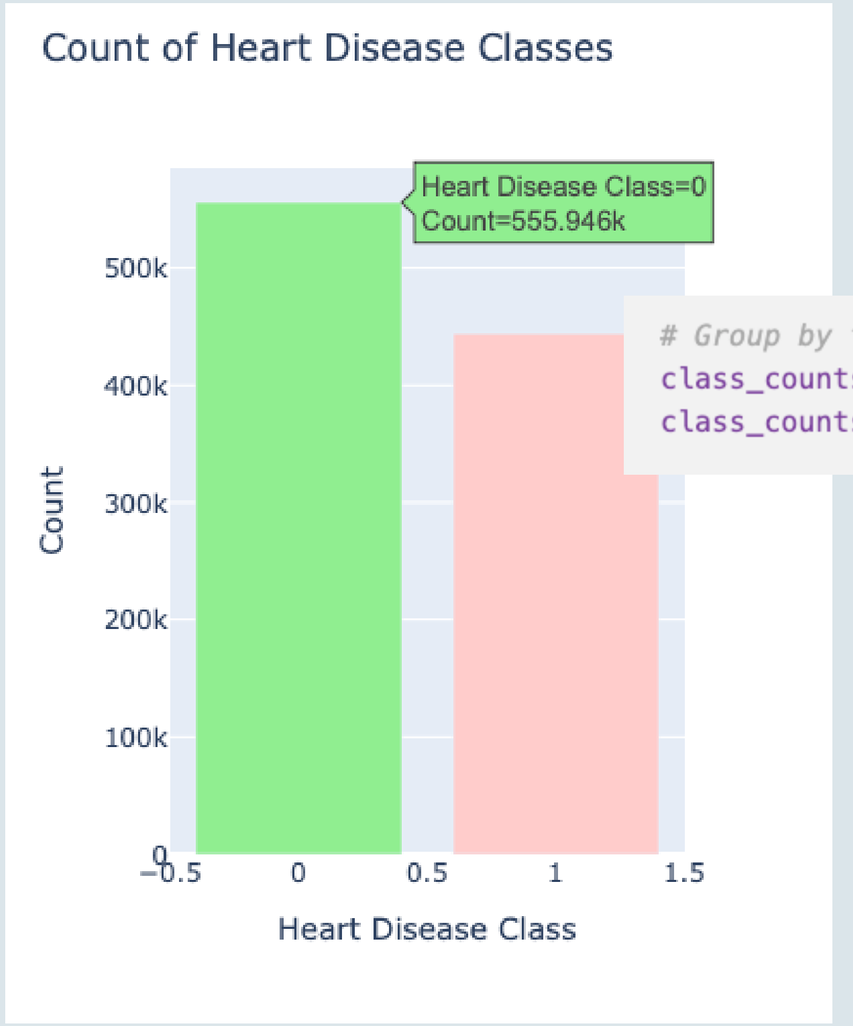
Link to my Code



# Exploratory Data Analysis

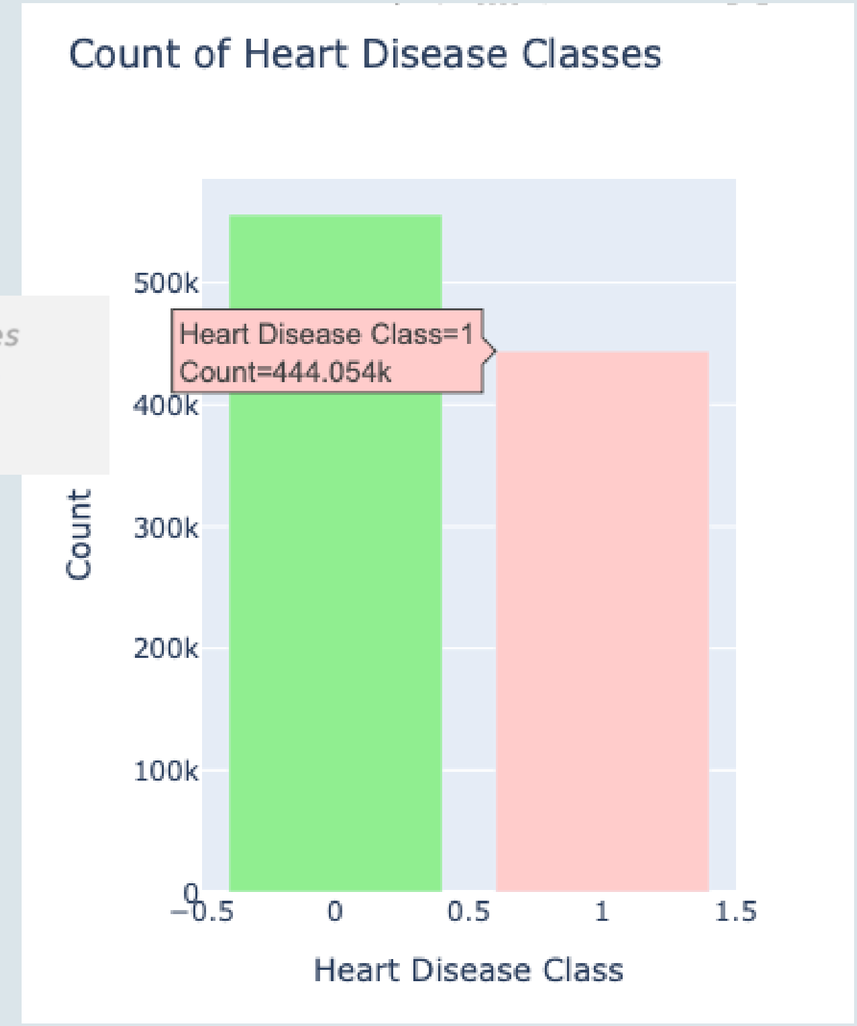
## Distribution of Class in Target Variable

Link to my Code




```
# Group by the class column and count occurrences
class_counts = spark_df.groupBy("class").count()
class_counts.show()
```

```
+-----+------+
|class| count|
+-----+------+
|    1|444054|
|    0|555946|
+-----+------+
```



# Task 1: Data Analysis

## Demographic Of Class by Age using Spark

Link to my Code

```
# Count class occurrences grouped by age group and class
age_class_counts = spark_df.groupBy("age", "class").count().orderBy("age")
age_class_counts.show()
```

```
+---+-----+-----+
|age|class|count|
+---+-----+-----+
| 26|    0|    1|
| 27|    0|    2|
| 28|    1|    4|
| 28|    0|   11|
| 29|    0|   28|
| 29|    1|    6|
| 30|    1|   20|
| 30|    0|   78|
| 31|    1|   62|
| 31|    0|  189|
| 32|    0|  476|
| 32|    1|  155|
| 33|    0| 1032|
```

```
from pyspark.sql.functions import when, col

# Create age group column
spark_df = spark_df.withColumn(
    "age_group",
    when(col("age") < 30, "<30")
    .when((col("age") >= 30) & (col("age") < 40), "30-39")
    .when((col("age") >= 40) & (col("age") < 50), "40-49")
    .when((col("age") >= 50) & (col("age") < 60), "50-59")
    .when((col("age") >= 60) & (col("age") < 70), "60-69")
    .otherwise("70+")
)
```

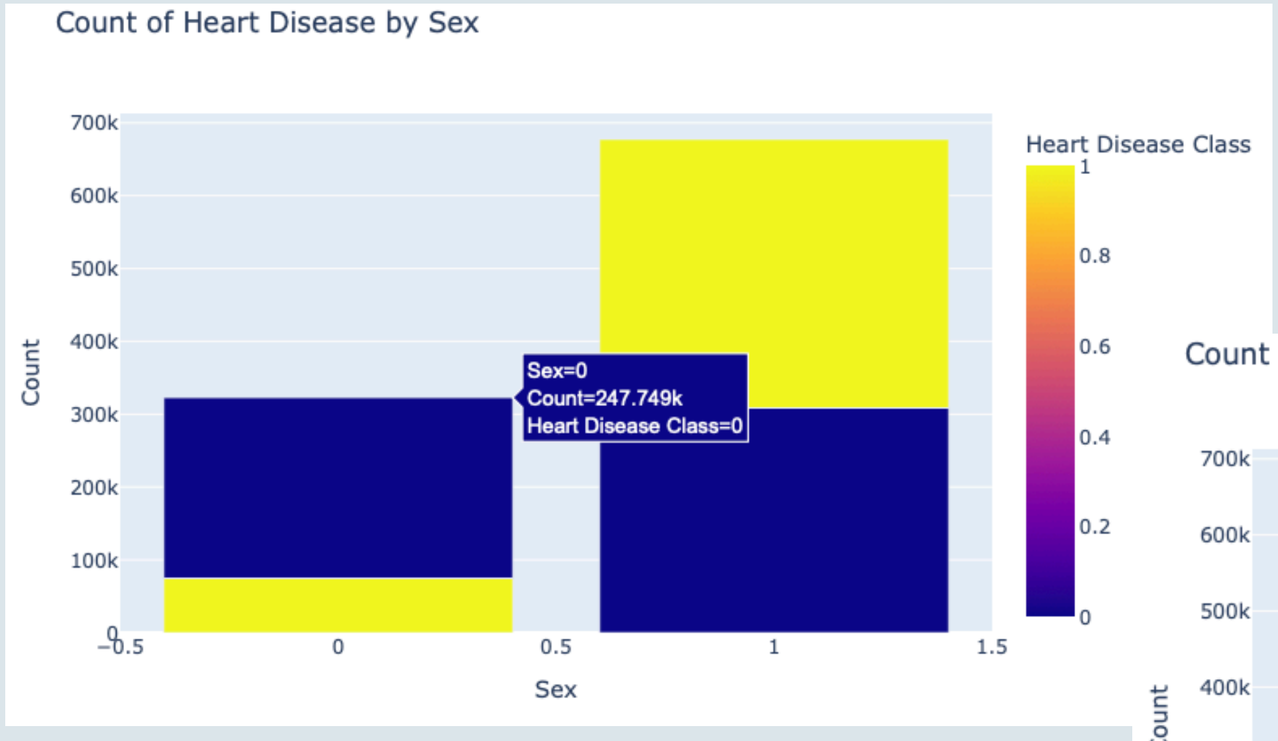
```
spark_df.groupBy("age_group", "class").count().orderBy("age_group").show()
```

```
+---------+-----+------+
|age_group|class| count|
+---------+-----+------+
|    30-39|    0| 47562|
|    30-39|    1| 14558|
|    40-49|    0|172298|
|    40-49|    1| 69137|
|    50-59|    1|183438|
|    50-59|    0|194994|
|    60-69|    1|158817|
|    60-69|    0|121314|
```

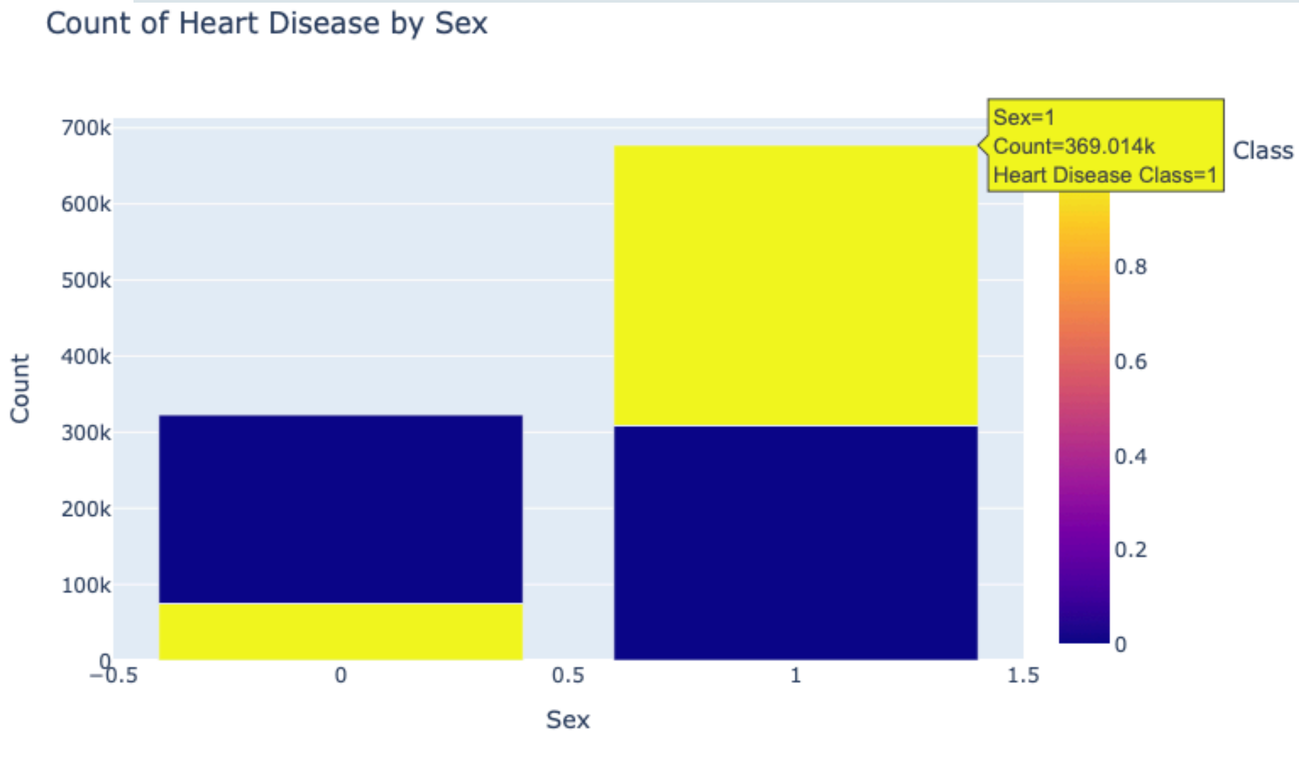
# Task 2: Data Analysis

## Demographic Of Class by Sex using Spark

Link to my Code




```
+---+-----+------+
|sex|class| count|
+---+-----+------+
|  1|    0|308197|
|  0|    1| 75040|
|  0|    0|247749|
|  1|    1|369014|
+---+-----+------+
```




```
# Group by 'class' and 'sex' and count the occurrences
class_sex_counts = spark_df.groupBy("sex", "class").count()

# Show the result
class_sex_counts.show()
```

# Task 3: Data Analysis

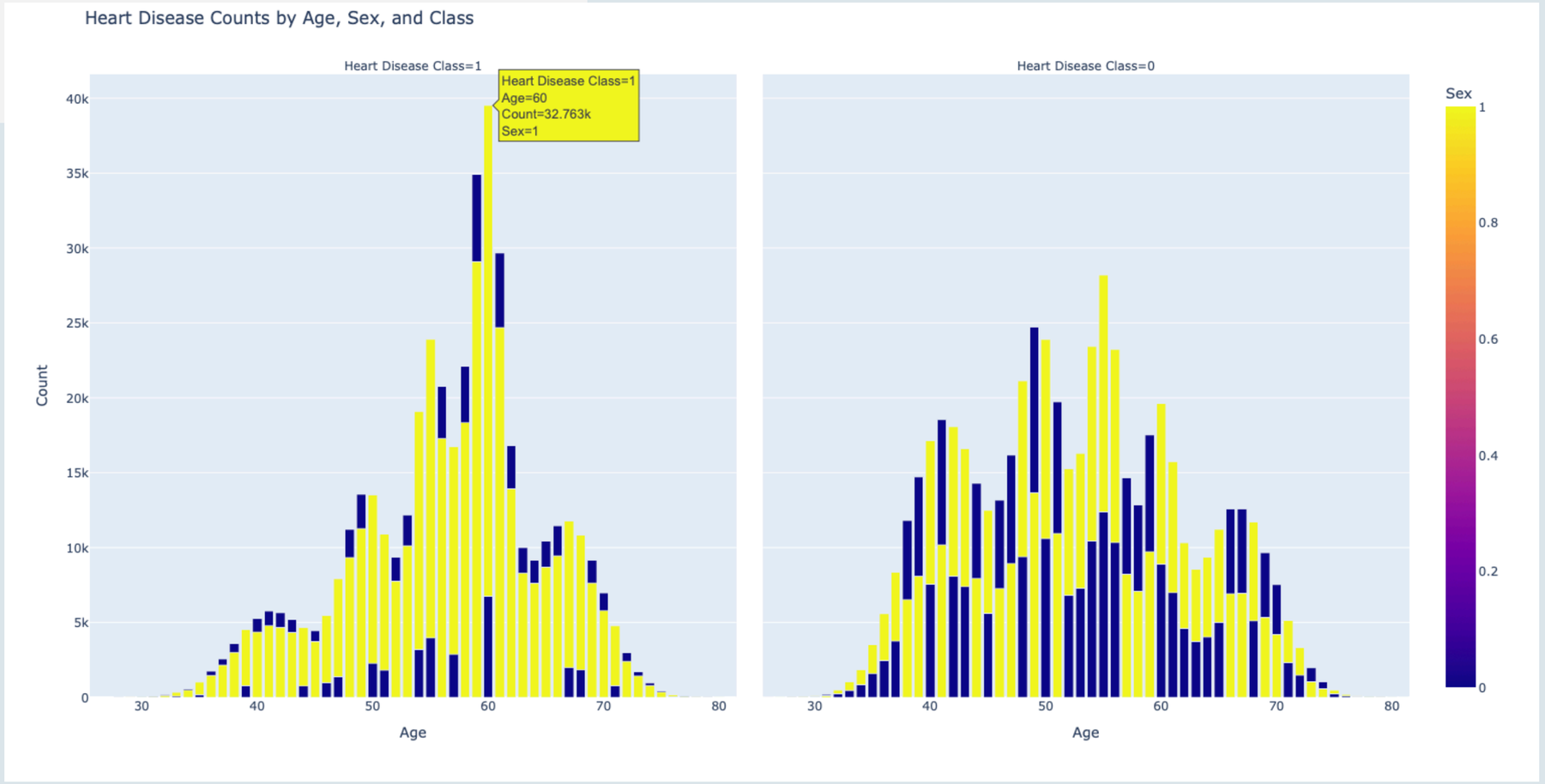
## Demographic Of Class by Age & Sex using Spark

Link to my Code

```
# Group by 'sex' and 'age' and count occurrences of each class
sex_age_class_counts = spark_df.groupBy("sex", "age", "class").count()

# Show the result
sex_age_class_counts.show()
```

```
+---+---+-----+-----+
|sex|age|class|count|
+---+---+-----+-----+
|  0| 60|    1| 6756|
|  1| 59|    0| 9726|
|  1| 69|    1| 7639|
|  0| 42|    0| 8095|
|  1| 45|    1| 3709|
|  0| 55|    1| 3968|
|  0| 53|    0| 7282|
|  1| 62|    1|13906|
|  1| 59|    1|29087|
|  0| 65|    0| 4992|
```



# Task 4: Data Analysis

## Demographic Of Class by Exercise, Thelessemia (other Features)

```
+----+-----+------+
|thal|class| count|
+----+-----+------+
|   6|    0| 25564|
|   7|    0| 94286|
|   6|    1| 32541|
|   3|    1|121956|
|   7|    1|289557|
|   3|    0|436096|
+----+-----+------+
```

```python
# Group by 'thal' and 'class' and count occurrences
thal_class_counts = spark_df.groupBy("thal", "class").count()

# Show the result

thal_class_counts.show()
```

```python
# Group by 'exercise_induced_angina' and 'class' and count occurrences
exercise_angina_class_counts = spark_df.groupBy("exercise_induced_angina", "class").count()

# Show the result
exercise_angina_class_counts.show()
```

```
+----------------------+-----+------+
|exercise_induced_angina|class| count|
+----------------------+-----+------+
|                     1|    0| 89007|
|                     0|    1|199726|
|                     0|    0|466939|
|                     1|    1|244328|
+----------------------+-----+------+
```

Link to my Code

# *Research Questions/Project Objectives*

## 1.Supervised Learning (Classifications Task)

- **Heart Disease Prediction**： Can we accurately predict the presence of heart disease based on patient demographic and clinical features?

## 2.Unsupervised Learning (Clustering)

# Machine Learning without Feature Engineering

## Supervised Learning

Link to my Code

```
# Assemble features
assembler = VectorAssembler(inputCols=['age', 'sex', 'chest', 'resting_blood_pressure',
                                       'serum_cholestoral', 'fasting_blood_sugar',
                                       'resting_electrocardiographic_results', 'maximum_heart_rate_achieved',
                                       'exercise_induced_angina', 'oldpeak', 'slope',
                                       'number_of_major_vessels', 'thal'],
                            outputCol='features')
assembled_df = assembler.transform(spark_df)

# Split data
train_data, test_data = assembled_df.randomSplit([0.8, 0.2], seed=42)

# Train logistic regression model
lr = LogisticRegression(featuresCol='features', labelCol='class')
lr_model = lr.fit(train_data)

# Make predictions
predictions = lr_model.transform(test_data)

# Evaluate the model using AUC (Area Under ROC Curve)
evaluator_auc = BinaryClassificationEvaluator(labelCol="class", metricName="areaUnderROC")
auc = evaluator_auc.evaluate(predictions)
print(f"Logistic Regression - Area Under ROC Curve (AUC): {auc}")

# Evaluate the model using F1 Score, Accuracy, Precision, and Recall
# F1 Score
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="class", metricName="f1")
f1 = evaluator_f1.evaluate(predictions)
print(f"Logistic Regression - F1 Score: {f1}")

# Accuracy
evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="class", metricName="accuracy")
accuracy = evaluator_accuracy.evaluate(predictions)
print(f"Logistic Regression - Accuracy: {accuracy}")

# Show predictions (optional)
predictions.select("class", "prediction", "probability").show(10)
```

```
Logistic Regression - Area Under ROC Curve (AUC): 0.947883217805503
Logistic Regression - F1 Score: 0.8774010116864452
Logistic Regression - Accuracy: 0.8775467422661889
+-----+----------+--------------------+
|class|prediction|         probability|
+-----+----------+--------------------+
|    0|       0.0|[0.97774438666500...|
|    1|       1.0|[0.18589726813829...|
|    0|       0.0|[0.91947153656234...|
|    0|       0.0|[0.86573474725702...|
|    1|       0.0|[0.54495370407276...|
|    0|       0.0|[0.85388183028037...|
|    1|       1.0|[0.04582104031686...|
|    0|       1.0|[0.49284096841373...|
|    0|       0.0|[0.99445603378227...|
|    0|       0.0|[0.96977032502975...|
+-----+----------+--------------------+
only showing top 10 rows
```

### 1- Logistic Regression

# Machine Learning without Feature Engineering

## Supervised Learning

Link to my Code

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator

# Train Random Forest model
rf = RandomForestClassifier(featuresCol='features', labelCol='class', numTrees=100)  # Adjustable numTrees
rf_model = rf.fit(train_data)

# Make predictions
rf_predictions = rf_model.transform(test_data)

# Evaluate the model using AUC (Area Under ROC Curve)
rf_evaluator_auc = BinaryClassificationEvaluator(labelCol="class", metricName="areaUnderROC")
rf_auc = rf_evaluator_auc.evaluate(rf_predictions)
print(f"Random Forest - Area Under ROC Curve (AUC): {rf_auc}")

# Evaluate the model using F1 Score, Accuracy, Precision, and Recall
# F1 Score
rf_evaluator_f1 = MulticlassClassificationEvaluator(labelCol="class", metricName="f1")
rf_f1 = rf_evaluator_f1.evaluate(rf_predictions)
print(f"Random Forest - F1 Score: {rf_f1}")

# Accuracy
rf_evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="class", metricName="accuracy")
rf_accuracy = rf_evaluator_accuracy.evaluate(rf_predictions)
print(f"Random Forest - Accuracy: {rf_accuracy}")


# Show predictions (optional)
rf_predictions.select("class", "prediction", "probability").show(10)
```

```
Random Forest - Area Under ROC Curve (AUC): 0.9464112263549657
Random Forest - F1 Score: 0.8757634867025537
Random Forest - Accuracy: 0.8759445780967687
+-----+----------+--------------------+
|class|prediction|         probability|
+-----+----------+--------------------+
|    0|       0.0|[0.92272482595370...|
|    1|       1.0|[0.35022118475375...|
|    0|       0.0|[0.66802526455073...|
|    0|       0.0|[0.72514739689726...|
|    1|       1.0|[0.29219797264854...|
|    0|       0.0|[0.74156980869055...|
|    1|       1.0|[0.15391025064654...|
|    0|       1.0|[0.37639020903494...|
|    0|       0.0|[0.92408527693104...|
|    0|       0.0|[0.82130602448269...|
+-----+----------+--------------------+
only showing top 10 rows
```

**2- Random Forest**

# Machine Learning without Feature Engineering

## Feature Importance

Link to my Code

```python
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import VectorAssembler

# Prepare features
feature_cols = [col for col in spark_df.columns if col != 'class']
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
df_transformed = assembler.transform(spark_df)

# Train Random Forest model
rf = RandomForestClassifier(featuresCol='features', labelCol='class', numTrees=100, seed=42)
rf_model = rf.fit(df_transformed)

# Extract feature importance
importances = rf_model.featureImportances

# Match feature names with their importance
feature_importances = [(feature, importance) for feature, importance in zip(feature_cols, importances.toArray())]
feature_importances_sorted = sorted(feature_importances, key=lambda x: x[1], reverse=True)

# Display important features
print("Feature Importances:")
for feature, importance in feature_importances_sorted:
    print(f"{feature}: {importance}")
```

```
Feature Importances:
thal: 0.3206979916145547
chest: 0.19511642638665375
number_of_major_vessels: 0.18554664101883503
exercise_induced_angina: 0.10104220713742058
slope: 0.06900530612243316
oldpeak: 0.05302505339551762
maximum_heart_rate_achieved: 0.05013729870382913
sex: 0.01183854920202481
age: 0.011055998149588764
resting_electrocardiographic_results: 0.0020158840158752693
serum_cholestoral: 0.000447197363251296
resting_blood_pressure: 7.144689001576017e-05
fasting_blood_sugar: 0.0
```

# Machine Learning without Feature Engineering

## UnSupervised Learning

Link to my Code

```
# Initialize KMeans
kmeans = KMeans(k=3, seed=1, featuresCol="features", predictionCol="prediction")
```

```
Cluster Centers:  [array([5.40654642e+01, 6.56814800e-01, 3.11169498e+00, 1.30962203e+02,
       2.02571603e+02, 1.52214297e-01, 9.92897829e-01, 1.51299024e+02,
       3.04112998e-01, 9.84520851e-01, 1.56553087e+00, 6.15312773e-01,
       4.56056858e+00]), array([5.47276280e+01, 6.98314749e-01, 3.22611693e+00, 1.31578567e+02,
       3.47973891e+02, 1.50868678e-01, 1.04763961e+00, 1.48316353e+02,
       3.59708803e-01, 1.11683912e+00, 1.62707121e+00, 7.43847791e-01,
       4.84690566e+00]), array([5.46191382e+01, 6.87956749e-01, 3.20343631e+00, 1.31607573e+02,
       2.60328622e+02, 1.49872457e-01, 1.03902983e+00, 1.48610642e+02,
       3.49820767e-01, 1.08658063e+00, 1.61144357e+00, 7.17798024e-01,
       4.79306939e+00])]
+--------------------+----------+
|            features|prediction|
+--------------------+----------+
|[53.0,1.0,1.0,117...|         2|
|[37.0,0.0,2.0,118...|         0|
|[49.0,1.0,3.0,141...|         0|
|[60.0,0.0,4.0,106...|         0|
|[59.0,1.0,3.0,121...|         2|
+--------------------+----------+
only showing top 5 rows

Silhouette with squared euclidean distance = 0.5014800464374891
```

### 1- Clustering with all Features

```
# Prepare features for clustering
assembler = VectorAssembler(inputCols=['resting_blood_pressure', 'serum_cholestoral', 'fasting_blood_sugar',
                                       'resting_electrocardiographic_results', 'maximum_heart_rate_achieved',
                                       'exercise_induced_angina', 'oldpeak', 'slope',
                                       'number_of_major_vessels', 'thal'],
                            outputCol='features')
```

```
Silhouette with squared euclidean distance = 0.5160035264726472
+--------------------+-------+
|            features|cluster|
+--------------------+-------+
|[117.978412,242.0...|      1|
|[118.45567,218.15...|      0|
|[141.819366,173.3...|      0|
|[106.368725,222.7...|      0|
|[121.035286,257.2...|      1|
|[131.62802,199.43...|      0|
|(10,[0,1,4,7,9],[...|      0|
|[107.34438,266.41...|      1|
|[126.469256,262.1...|      1|
|[123.366212,385.0...|      2|
```

### 2- Clustering with selected Features

# *Performance evaluation metrics -- Main idea*

In our heart disease prediction task, the primary focus is to minimize missed diagnoses (**false negatives**), which means paying special attention to cases where patients truly have the condition but are predicted as healthy by the model.

Based on this objective, our primary evaluation metric should emphasize **recall**, as it reflects the model's ability to identify actual positive cases. When necessary, we also consider the **F1 score** to balance precision and recall for a more comprehensive assessment.

# Performance evaluation metrics

```
def print_metrics(predictions, model_name):
    """
    Calculate and print the model evaluation metrics.

    Parameters:
        predictions: DataFrame containing the prediction results
        model_name: The name of the model, for display purposes
    """
    # Calculate weighted recall
    evaluator_recall = MulticlassClassificationEvaluator(
        labelCol="label",
        predictionCol="prediction",
        metricName="weightedRecall"
    )

    # Calculate F1 score
    evaluator_f1 = MulticlassClassificationEvaluator(
        labelCol="label",
        predictionCol="prediction",
        metricName="f1"    # This is the key modification
    )

    recall = evaluator_recall.evaluate(predictions)
    f1 = evaluator_f1.evaluate(predictions)

    print(f"\n{model_name} Evaluation Metrics:")
    print(f"Weighted Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
```

We define a function print_metrics to evaluate the performance of the machine learning model. We mainly use the weighted recall and F1 score.

# *Pipeline provided by Spark*

```
from pyspark.ml import Pipeline

# Create a Random Forest pipeline
rf_pipeline = Pipeline(stages=[
        rf_assembler,
        RandomForestClassifier(labelCol="label",
                                                featuresCol="features_rf",
                                                seed=42)
])

# Get the Random Forest model from the pipeline
rf_model = rf_pipeline.fit(train_df).stages[-1]
```

We used the Pipeline provided by Spark to simplify and standardize the machine learning workflow. By combining steps such as data preprocessing and model training, the code is modularized.

# *Parameters tunning methods*

The core of hyperparameter tuning is to **identify a set of hyperparameters that maximize the model's performance**. We employ a combination of **Grid Search** and **Cross-Validation** for this process.

Grid Search systematically explores the parameter grid, while Cross-Validation evaluates the performance of each parameter set, ensuring that the selected parameters generalize well to unseen data. This approach helps us optimize the model's performance while **minimizing the risk of overfitting**.

# *Supervised learning*

1.Random Forest

2.Logistic Regression

# *Random Forest*

# *Grid Search and Cross Validation*

Link to my Code

```
# Define the parameter grid
# Select a smaller range of parameters to balance training time and model performance
rf_paramGrid = ParamGridBuilder() \
        .addGrid(rf_model.numTrees, [10, 20]) \
        .addGrid(rf_model.maxDepth, [5, 7]) \
        .build()
```

```
# Configure cross-validation
rf_cv = CrossValidator(
        estimator=rf_pipeline,
        estimatorParamMaps=rf_paramGrid,
        evaluator=evaluator_rf,
        numFolds=3,
        seed=42,
        parallelism=2    # Enable parallel training for efficiency
)
```

# *Optimal Parameters Comibination and Model Performance*

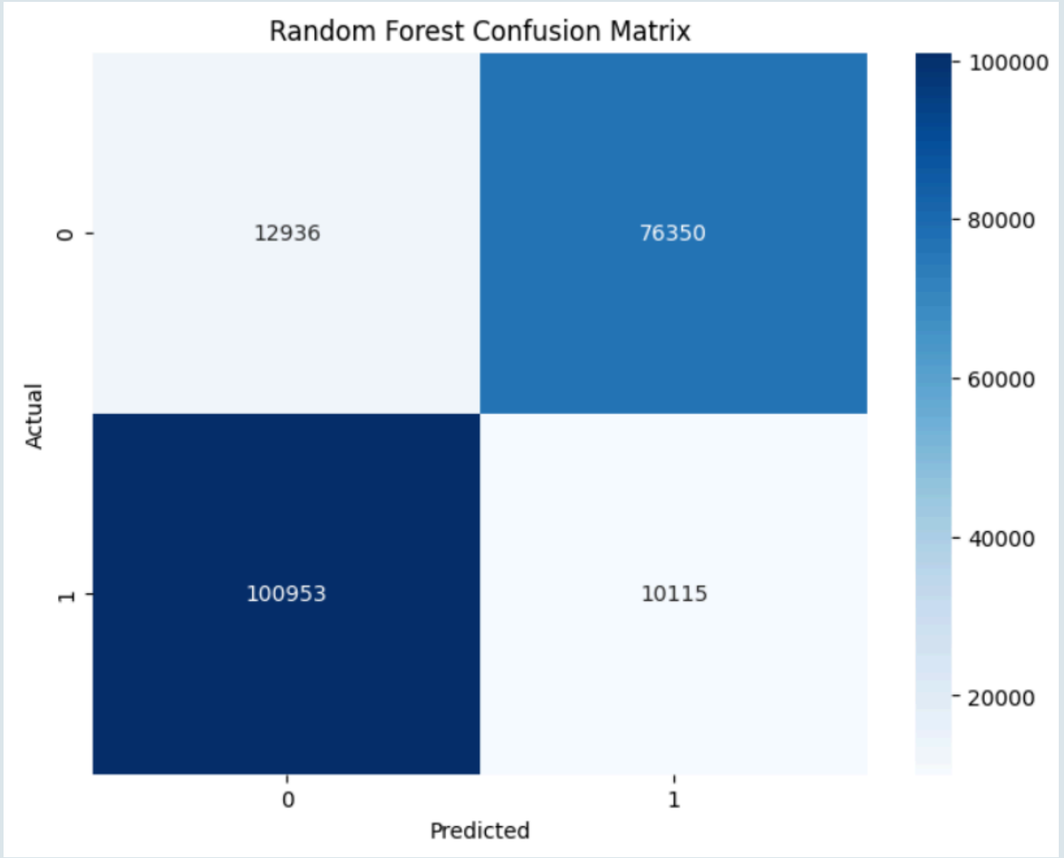
Link to my Code

```
Training the Random Forest model...

Best Random Forest Parameters:
Number of Trees (numTrees): 20
Tree Depth (maxDepth): 7

Evaluating the Random Forest model on the test set...

Random Forest Evaluation Metrics:
Weighted Recall: 0.8849
F1 Score: 0.8847
```



# *Logistic Regression*

# *Feature Engineering : One-hot Coding and Features Standirization*

Link to my Code

```
# Create Logistic Regression complete pipeline
lr_pipeline = Pipeline(stages=[
        # Feature engineering steps
        lr_encoder_chest,                # One-hot encoding for chest vari
        lr_scaler_age,                   # Standardize age
        lr_scaler_rbp,                   # Standardize blood pressure
        lr_scaler_schol,                 # Standardize cholesterol
        lr_scaler_max_hr,                # Standardize maximum heart rate
        lr_scaler_oldpeak,               # Standardize ST depression
        lr_assembler,                    # Assemble all features
        LogisticRegression(labelCol="label",
                           featuresCol="features_lr",
                           family="binomial")    # Specify
])
```

Logistic regression uses the **gradient descent optimization** algorithm to minimize the loss function, When calculating gradient updates, the distribution and range of features directly affect the learning speed. Therefore we need to preprocess the features appropriately (Feature Engineering)

# *Grid Search and Cross Validation*

Link to my Code

```
lr_paramGrid = ParamGridBuilder() \
        .addGrid(lr_model.regParam, [0.01, 0.1]) \
        .addGrid(lr_model.elasticNetParam, [0.0, 0.5]) \
        .build()
```

```
lr_cv = CrossValidator(
        estimator=lr_pipeline,
        estimatorParamMaps=lr_paramGrid,
        evaluator=evaluator_lr,
        numFolds=3,
        seed=42
)
```

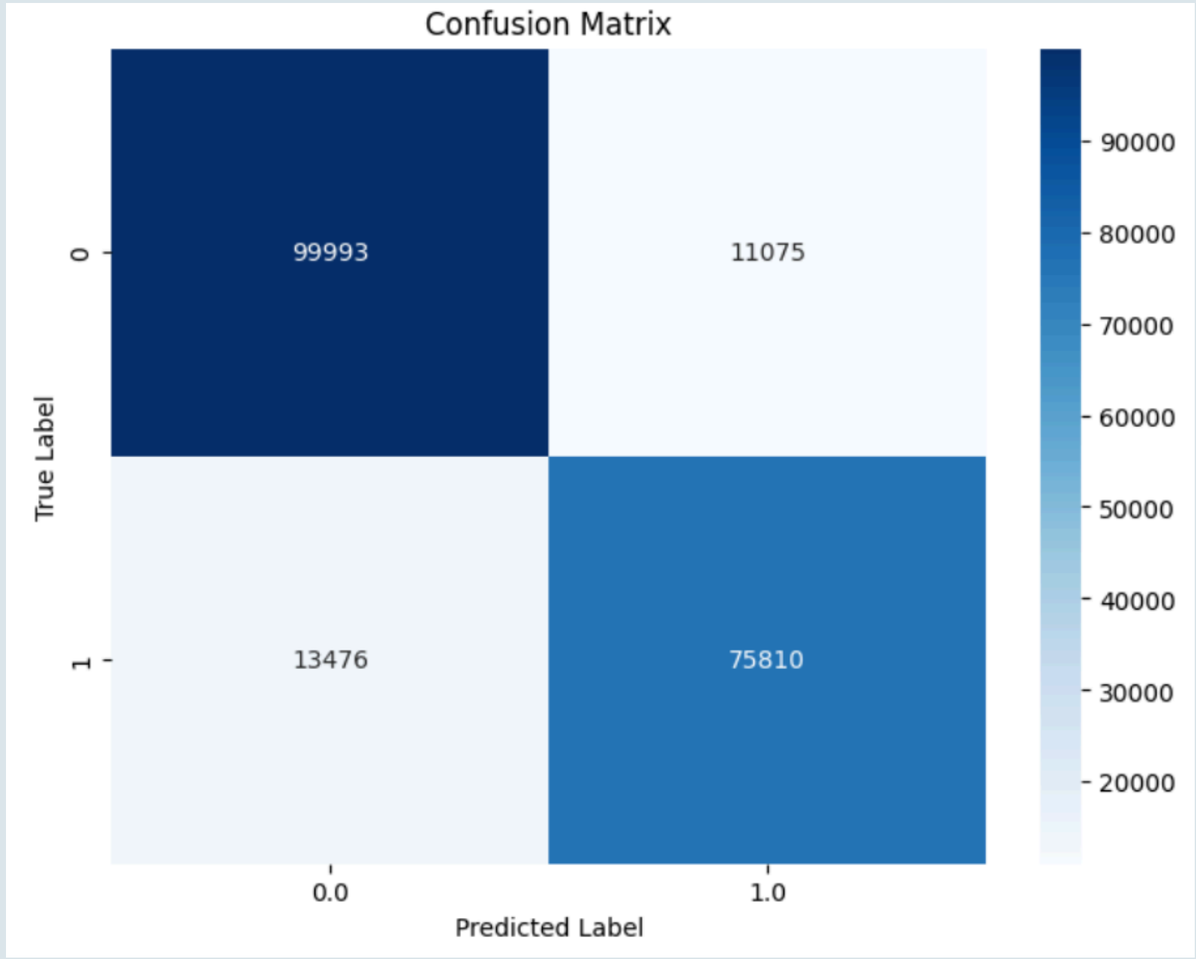
# *Optimal Parameters Comibination and Model Performance*

Link to my Code

```
RegParam: 0.01, ElasticNetParam: 0.0
RegParam: 0.01, ElasticNetParam: 0.5
RegParam: 0.1, ElasticNetParam: 0.0
RegParam: 0.1, ElasticNetParam: 0.5

Best Logistic Regression Parameters:
RegParam: 0.01
ElasticNetParam: 0.0

Test Set Metrics:
Recall: 0.8775
F1 Score: 0.8773
```



# *Optimal Parameters Comibination and Model Performance*

```
Training the Random Forest model...

Best Random Forest Parameters:
Number of Trees (numTrees): 20
Tree Depth (maxDepth): 7

Evaluating the Random Forest model on the test set...

Random Forest Evaluation Metrics:
Weighted Recall: 0.8849
F1 Score: 0.8847
```

Link to my Code

```
RegParam: 0.01, ElasticNetParam: 0.0
RegParam: 0.01, ElasticNetParam: 0.5
RegParam: 0.1, ElasticNetParam: 0.0
RegParam: 0.1, ElasticNetParam: 0.5

Best Logistic Regression Parameters:
RegParam: 0.01
ElasticNetParam: 0.0

Test Set Metrics:
Recall: 0.8775
F1 Score: 0.8773
```

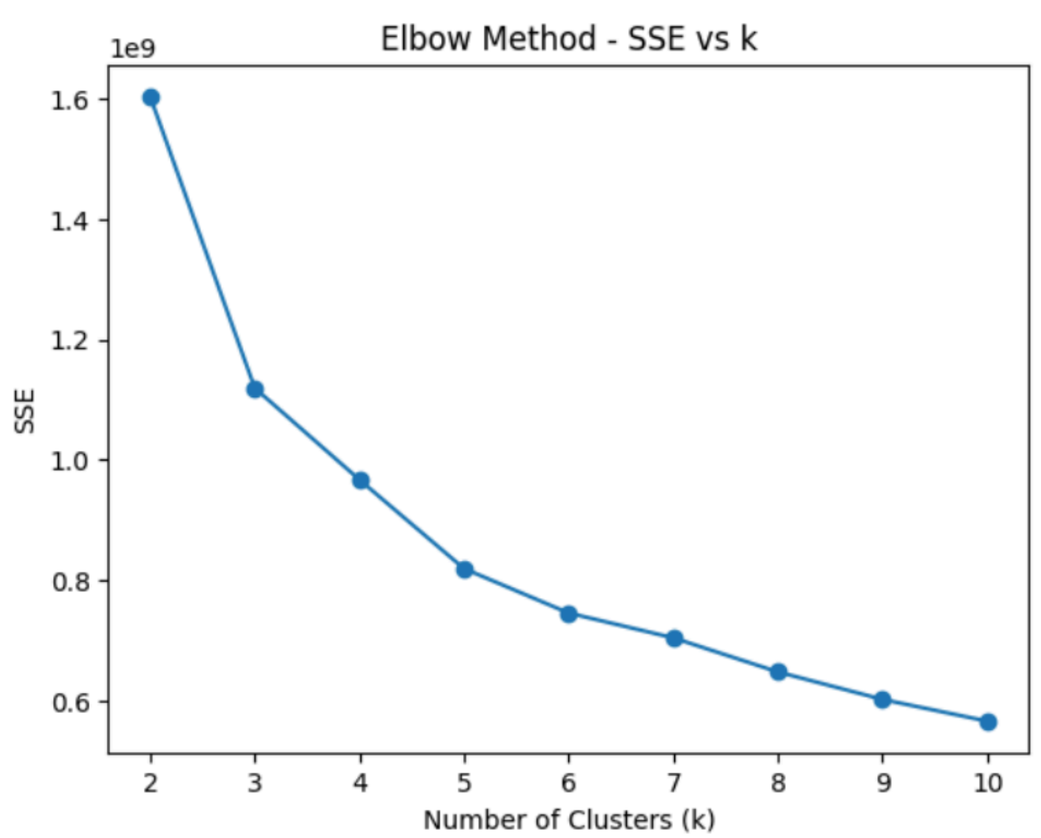
Comparing the recall and F1 score of the two models on the same test set, we can find that the performance of random forest on these two indicators is better than that of the logistic regression model.

# *Unsupervised learning -- Clustering*

```
k_values = range(2, 11)
sse_values = []

for k in k_values:
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features_ul")
    model = kmeans.fit(train_df_ul)
    sse_values.append(model.summary.trainingCost)


plt.plot(k_values, sse_values, marker='o')
plt.title('Elbow Method - SSE vs k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('SSE')
plt.show()
```




k = 3 or k = 4 yielded the best results

# *Conclusion*

1. Both supervised learning algorithms performed quite well.
the Random Forest model slightly outperformed a little better Logistic Regression

2.The performance of teh logistic regression bfore and after featuring engineering is similar.

*Thank you*