ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA HỆ THỐNG THÔNG TIN

Big Data

Heart Failure Prediction using PySpark

Big Data - MSIS405. N21. CTTT

Lecturer: Do Trong Hop

Author: Pham Thuy Dung - 20521214

Table of Contents

Introduction

Cardiovascular diseases (CVDs) are a major global health challenge, as they are responsible for an estimated 17.9 million deaths annually, representing 31% of all mortality worldwide. The majority of CVD deaths (80%) are caused by heart attacks and strokes, and many of them (one-third) occur prematurely in people younger than 70 years of age. Heart failure is a frequent complication of CVDs and this dataset provides 11 features that can be utilized to predict the likelihood of heart disease.

Early detection and management of cardiovascular disease or high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or established disease) is essential for improving outcomes and reducing morbidity and mortality. A machine learning model can be a valuable tool for this purpose, as it can analyze the data and identify patterns and associations that can inform clinical decision making.

About Dataset

Source

This dataset [1] was created by combining different datasets already available independently but not combined before. In this dataset, 5 heart datasets are combined over 11 common features which makes it the largest heart disease dataset available so far for research purposes. The five datasets used for its curation are:

Cleveland: 303 observations

Hungarian: 294 observations

Switzerland: 123 observations

Long Beach VA: 200 observations

Stalog (Heart) Data Set: 270 observations

Total: 1190 observations

Duplicated: 272 observations

Final dataset: 918 observations

Every dataset used can be found under the Index of heart disease datasets from UCI Machine Learning Repository on the following link: https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/

Attribute Information

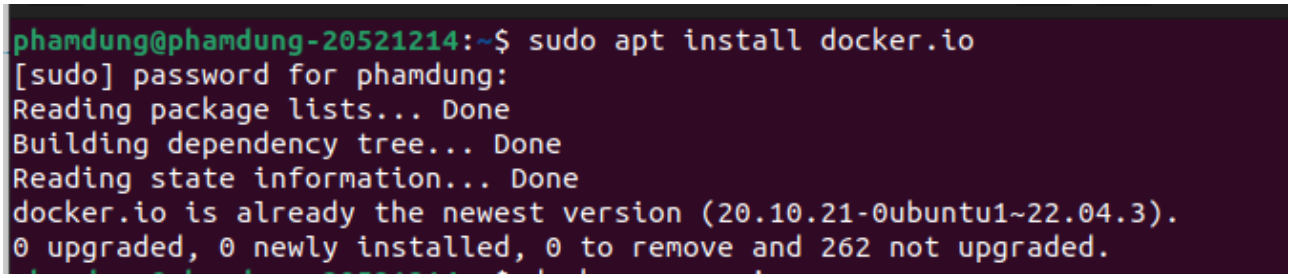
- Age: age of the patient [years]
- Sex: sex of the patient [M: Male, F: Female]
- ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- RestingBP: resting blood pressure [mm Hg]

- Cholesterol: serum cholesterol [mm/dl]
- FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
- RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
- ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
- Oldpeak: oldpeak = ST [Numeric value measured in depression]
- ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
- HeartDisease: output class [1: heart disease, 0: Normal

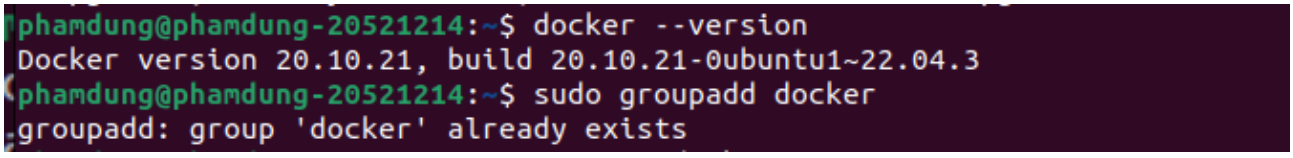
Code Detail Explaining

Install Jupyter notebook

Step 1: Install Docker



Step 2: See docker version after install



Step 3: run hello-world to check docker working

```
phamdung@phamdung-20521214:~$ newgrp docker
phamdung@phamdung-20521214:~$ sudo chmod 666 /var/run/docker.sock
phamdung@phamdung-20521214:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Step 4: Install Jupyter note book, in here, because I install it before, show it show it messeages

```
phamdung@phamdung-20521214:~$ docker run -p 8888:8888 -e JUPYTER_ENALBE_LAB=yes --name pyspark jupyter/pyspark-notebo
docker: Error response from daemon: Conflict. The container name "/pyspark" is already in use by container "0fbc9db72
e) that container to be able to reuse that name.
See 'docker run --help'.
```

Step 5: Checking Jupyter notebook install successfully

```
phamdung@phamdung-20521214:~$ docker start -a pyspark
Entered start.sh with args: jupyter lab
/usr/local/bin/start.sh: running hooks in /usr/local/bin/before-notebook.d as uid / gid: 1000 / 100
/usr/local/bin/start.sh: running script /usr/local/bin/before-notebook.d/spark-config.sh
/usr/local/bin/start.sh: done running hooks in /usr/local/bin/before-notebook.d
Executing the command: jupyter lab
[I 2023-07-23 06:10:04.742 ServerApp] Package jupyterlab took 0.0000s to import
[I 2023-07-23 06:10:04.773 ServerApp] Package jupyter_lsp took 0.0306s to import
[W 2023-07-23 06:10:04.773 ServerApp] A `_jupyter_server_extension_points` function was not found in jupyt
 This function name will be deprecated in future releases of Jupyter Server.
[I 2023-07-23 06:10:04.777 ServerApp] Package jupyter_server_mathjax took 0.0036s to import
[I 2023-07-23 06:10:04.795 ServerApp] Package jupyter_server_terminals took 0.0171s to import
[I 2023-07-23 06:10:04.861 ServerApp] Package jupyterlab_git took 0.0651s to import
[I 2023-07-23 06:10:04.864 ServerApp] Package nbclassic took 0.0027s to import
[W 2023-07-23 06:10:04.869 ServerApp] A `_jupyter_server_extension_points` function was not found in nbcla
his function name will be deprecated in future releases of Jupyter Server.
[I 2023-07-23 06:10:04.869 ServerApp] Package nbdime took 0.0000s to import
[I 2023-07-23 06:10:04.869 ServerApp] Package notebook_shim took 0.0000s to import
[W 2023-07-23 06:10:04.869 ServerApp] A `_jupyter_server_extension_points` function was not found in noteb
w. This function name will be deprecated in future releases of Jupyter Server.
[I 2023-07-23 06:10:04.870 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2023-07-23 06:10:04.872 ServerApp] jupyter_server_mathjax | extension was successfully linked.
[I 2023-07-23 06:10:04.874 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2023-07-23 06:10:04.877 ServerApp] jupyterlab | extension was successfully linked.
[I 2023-07-23 06:10:04.877 ServerApp] jupyterlab_git | extension was successfully linked.
[W 2023-07-23 06:10:04.882 NotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config will be
[W 2023-07-23 06:10:04.882 NotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config will be
[I 2023-07-23 06:10:04.884 ServerApp] nbclassic | extension was successfully linked.
```

Step 6: Copy the link and paste it in browser to activate the jupyter notebook

```
    To access the server, open this file in a browser:
        file:///home/jovyan/.local/share/jupyter/runtime/jpserver-8-open.html
    Or copy and paste one of these URLs:
        http://0fbc9db727c5:8888/lab?token=eccbdd3b7e9aaeb56d8e3faebf9241f887474012ca25ff6f
        http://127.0.0.1:8888/lab?token=eccbdd3b7e9aaeb56d8e3faebf9241f887474012ca25ff6f
[I 2023-07-23 06:10:06.393 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-l
er, pyright, python-language-server, python-lsp-server, r-languageserver, sql-language-server, texlab, ty
uageserver-bin, vscode-json-languageserver-bin, yaml-language-server
0.00s - Debugger warning: It seems that frozen modules are being used, which may
0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off
0.00s - to python to disable frozen modules.
0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to disable this validation.
[W 2023-07-23 06:10:14.887 LabApp] The extension "nbdime-jupyterlab" is outdated.
```

Explaining the code in Jupyter notebook

Step 1: Import necessary libraries

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType
from pyspark.sql.types import IntegerType, DoubleType
from pyspark.sql.functions import col, when, lit, median, sum as spark_sum
from pyspark.ml.feature import StringIndexer
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml import Pipeline
from pyspark.ml.stat import Correlation
from pyspark.sql.functions import expr
from pyspark_dist_explore import hist
from pyspark.sql.functions import count, round
from pyspark.sql import functions as F
from IPython.display import display
from pyspark.ml.feature import StandardScaler
from pyspark.ml.linalg import Vectors
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.linalg import Vectors
from pyspark.sql.functions import udf
from pyspark.sql.types import DoubleType
from pyspark.ml.classification import LinearSVC
from pyspark.ml.classification import RandomForestClassifier
```

Step 2: Create a SparkSession

```
spark = SparkSession.builder.appName("HeartFailurePrediction").getOrCreate()
```

Step 3: Read the data using PySpark

```
heart = spark.read.csv("heart.csv", header=True, inferSchema=True)
```

Step 4: Set matplotlib configuration

```
plt.rcParams['figure.figsize'] = [8, 8]
plt.rcParams.update({'font.size': 15})
plt.rcParams['font.family'] = 'sans-serif'
```

Step 5: Display more rows and set precision for floating-point numbers

```
spark.conf.set("spark.sql.repl.eagerEval.maxNumRows", 1000)
spark.conf.set("spark.sql.repl.eagerEval.truncate", 1000)
spark.conf.set("spark.sql.precision", 3)
```

Step 6: Read the data using PySpark and display the first 20 rows

```
heart.show(20)
```

Here is result

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
| 40|  M|          ATA|      140|        289|        0|    Normal|  172|             N|    0.0|      Up|           0|
| 49|  F|          NAP|      160|        180|        0|    Normal|  156|             N|    1.0|    Flat|           1|
| 37|  M|          ATA|      130|        283|        0|        ST|   98|             N|    0.0|      Up|           0|
| 48|  F|          ASY|      138|        214|        0|    Normal|  108|             Y|    1.5|    Flat|           1|
| 54|  M|          NAP|      150|        195|        0|    Normal|  122|             N|    0.0|      Up|           0|
| 39|  M|          NAP|      120|        339|        0|    Normal|  170|             N|    0.0|      Up|           0|
| 45|  F|          ATA|      130|        237|        0|    Normal|  170|             N|    0.0|      Up|           0|
| 54|  M|          ATA|      110|        208|        0|    Normal|  142|             N|    0.0|      Up|           0|
| 37|  M|          ASY|      140|        207|        0|    Normal|  130|             Y|    1.5|    Flat|           1|
| 48|  F|          ATA|      120|        284|        0|    Normal|  120|             N|    0.0|      Up|           0|
| 37|  F|          NAP|      130|        211|        0|    Normal|  142|             N|    0.0|      Up|           0|
| 58|  M|          ATA|      136|        164|        0|        ST|   99|             Y|    2.0|    Flat|           1|
| 39|  M|          ATA|      120|        204|        0|    Normal|  145|             N|    0.0|      Up|           0|
| 49|  M|          ASY|      140|        234|        0|    Normal|  140|             Y|    1.0|    Flat|           1|
| 42|  F|          NAP|      115|        211|        0|        ST|  137|             N|    0.0|      Up|           0|
| 54|  F|          ATA|      120|        273|        0|    Normal|  150|             N|    1.5|    Flat|           0|
| 38|  M|          ASY|      110|        196|        0|    Normal|  166|             N|    0.0|    Flat|           1|
| 43|  F|          ATA|      120|        201|        0|    Normal|  165|             N|    0.0|      Up|           0|
| 60|  M|          ASY|      100|        248|        0|    Normal|  125|             N|    1.0|    Flat|           1|
| 36|  M|          ATA|      120|        267|        0|    Normal|  160|             N|    3.0|    Flat|           1|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
only showing top 20 rows
```

Step 7: Filter the DataFrame based on the condition 'Cholesterol == 0'

```
cholesterol = heart.filter(col('Cholesterol') == 0)

cholesterol.show()
```

Here is result

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
| 65|  M|          ASY|      115|          0|        0|    Normal|   93|             Y|    0.0|    Flat|           1|
| 32|  M|           TA|       95|          0|        1|    Normal|  127|             N|    0.7|      Up|           1|
| 61|  M|          ASY|      105|          0|        1|    Normal|  110|             Y|    1.5|      Up|           1|
| 50|  M|          ASY|      145|          0|        1|    Normal|  139|             Y|    0.7|    Flat|           1|
| 57|  M|          ASY|      110|          0|        1|        ST|  131|             Y|    1.4|      Up|           1|
| 51|  M|          ASY|      110|          0|        1|    Normal|   92|             N|    0.0|    Flat|           1|
| 47|  M|          ASY|      110|          0|        1|        ST|  149|             N|    2.1|      Up|           1|
| 60|  M|          ASY|      160|          0|        1|    Normal|  149|             N|    0.4|    Flat|           1|
| 55|  M|          ATA|      140|          0|        0|        ST|  150|             N|    0.2|      Up|           0|
| 53|  M|          ASY|      125|          0|        1|    Normal|  120|             N|    1.5|      Up|           1|
| 62|  F|          ASY|      120|          0|        1|        ST|  123|             Y|    1.7|    Down|           1|
| 51|  M|          ASY|       95|          0|        1|    Normal|  126|             N|    2.2|    Flat|           1|
| 51|  F|          ASY|      120|          0|        1|    Normal|  127|             Y|    1.5|      Up|           1|
| 55|  M|          ASY|      115|          0|        1|    Normal|  155|             N|    0.1|    Flat|           1|
| 53|  M|          ATA|      130|          0|        0|        ST|  120|             N|    0.7|    Down|           0|
| 58|  M|          ASY|      115|          0|        1|    Normal|  138|             N|    0.5|      Up|           1|
| 57|  M|          ASY|       95|          0|        1|    Normal|  182|             N|    0.7|    Down|           1|
| 65|  M|          ASY|      155|          0|        0|    Normal|  154|             N|    1.0|      Up|           0|
| 60|  M|          ASY|      125|          0|        1|    Normal|  110|             N|    0.1|      Up|           1|
| 41|  M|          ASY|      125|          0|        1|    Normal|  176|             N|    1.6|      Up|           1|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
only showing top 20 rows
```

Step 8: Display the number of rows and columns

```
num_rows = cholesterol.count()

num_columns = len(cholesterol.columns)

print("Number of Rows:", num_rows)
print("Number of Columns:", num_columns)
```

```
Number of Rows: 172
Number of Columns: 12
```

Step 9: Filter the DataFrame based on the condition 'MaxHR == 0'

```
max_hr = heart.filter(col('MaxHR') == 0)

max_hr.show()
```

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
```

Step 10: Replace zero values of 'Cholesterol' with NaN

```
heart = heart.withColumn('Cholesterol', when(col('Cholesterol') == 0, None).otherwise(col('Cholesterol')))

heart.show(1)
```

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
| 40|  M|          ATA|      140|        289|        0|    Normal|  172|             N|    0.0|      Up|           0|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
only showing top 1 row
```

Step 11: Calculate the median of the 'Cholesterol' column & Fill missing (null) values in 'Cholesterol' column with the calculated median value

```
median_value = heart.select(median(col("Cholesterol"))).collect()[0][0]
heart = heart.withColumn("Cholesterol", when(col("Cholesterol").isNull(), lit(median_value)).otherwise(col("Cholesterol")))
heart.show()
```

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
| 40|  M|          ATA|      140|      289.0|        0|    Normal|  172|             N|    0.0|      Up|           0|
| 49|  F|          NAP|      160|      180.0|        0|    Normal|  156|             N|    1.0|    Flat|           1|
| 37|  M|          ATA|      130|      283.0|        0|        ST|   98|             N|    0.0|      Up|           0|
| 48|  F|          ASY|      138|      214.0|        0|    Normal|  108|             Y|    1.5|    Flat|           1|
| 54|  M|          NAP|      150|      195.0|        0|    Normal|  122|             N|    0.0|      Up|           0|
| 39|  M|          NAP|      120|      339.0|        0|    Normal|  170|             N|    0.0|      Up|           0|
| 45|  F|          ATA|      130|      237.0|        0|    Normal|  170|             N|    0.0|      Up|           0|
| 54|  M|          ATA|      110|      208.0|        0|    Normal|  142|             N|    0.0|      Up|           0|
| 37|  M|          ASY|      140|      207.0|        0|    Normal|  130|             Y|    1.5|    Flat|           1|
| 48|  F|          ATA|      120|      284.0|        0|    Normal|  120|             N|    0.0|      Up|           0|
| 37|  F|          NAP|      130|      211.0|        0|    Normal|  142|             N|    0.0|      Up|           0|
| 58|  M|          ATA|      136|      164.0|        0|        ST|   99|             Y|    2.0|    Flat|           1|
| 39|  M|          ATA|      120|      204.0|        0|    Normal|  145|             N|    0.0|      Up|           0|
| 49|  M|          ASY|      140|      234.0|        0|    Normal|  140|             Y|    1.0|    Flat|           1|
| 42|  F|          NAP|      115|      211.0|        0|        ST|  137|             N|    0.0|      Up|           0|
| 54|  F|          ATA|      120|      273.0|        0|    Normal|  150|             N|    1.5|    Flat|           0|
| 38|  M|          ASY|      110|      196.0|        0|    Normal|  166|             N|    0.0|    Flat|           1|
| 43|  F|          ATA|      120|      201.0|        0|    Normal|  165|             N|    0.0|      Up|           0|
| 60|  M|          ASY|      100|      248.0|        0|    Normal|  125|             N|    1.0|    Flat|           1|
| 36|  M|          ATA|      120|      267.0|        0|    Normal|  160|             N|    3.0|    Flat|           1|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
only showing top 20 rows
```

Step 12: Filter the DataFrame based on the condition 'RestingBP == 0'

```
resting_bp = heart.filter(col('RestingBP') == 0)

resting_bp.show()
```

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
| 55|  M|          NAP|        0|      237.0|        0|    Normal|  155|             N|    1.5|    Flat|           1|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
```

Step 13: # Filter the DataFrame based on the condition 'RestingBP == 0'

```
filtered_heart = heart.filter(col('RestingBP') != 0)

heart = filtered_heart

heart.printSchema()
```

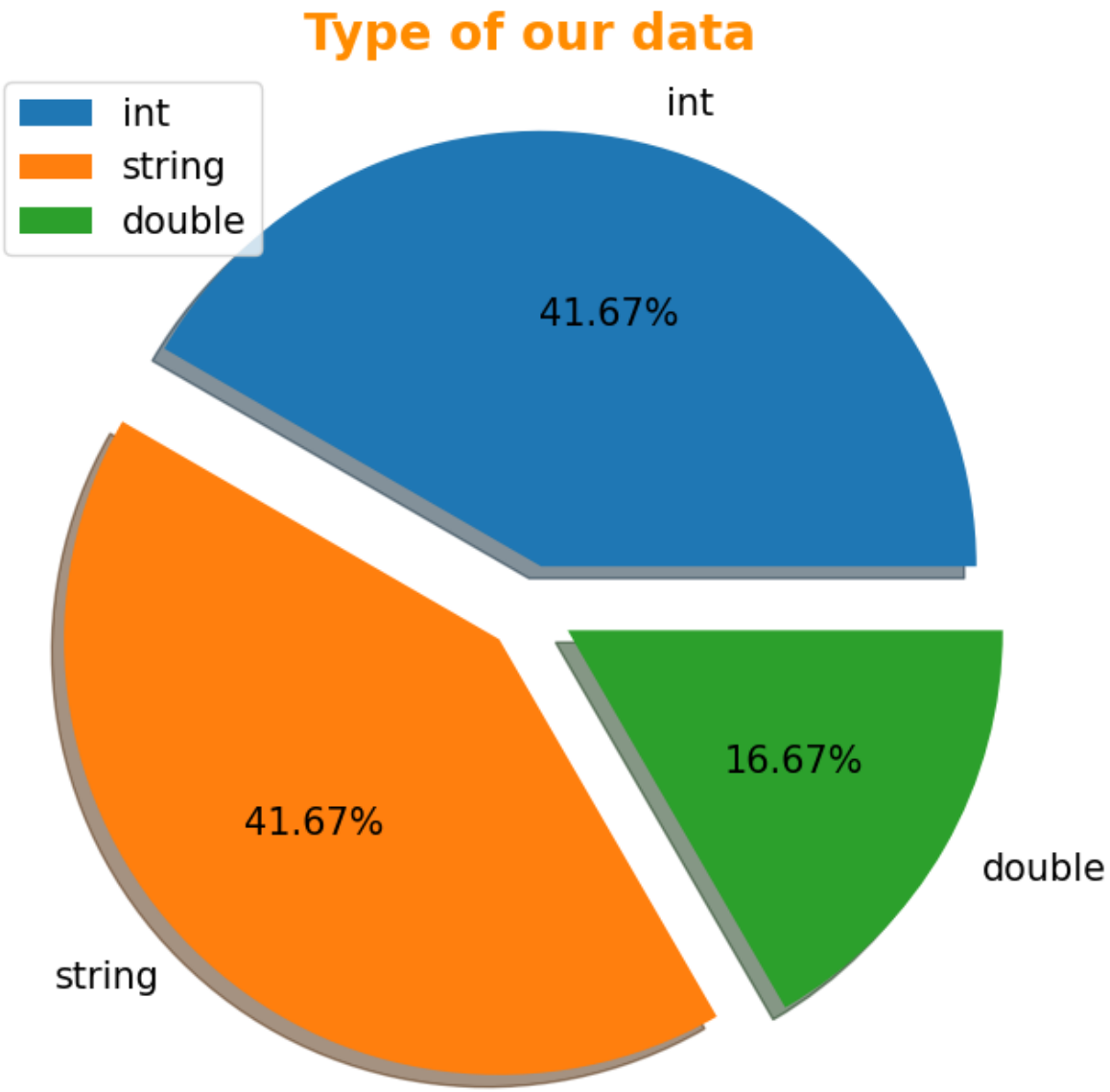
```
root
 |-- Age: integer (nullable = true)
 |-- Sex: string (nullable = true)
 |-- ChestPainType: string (nullable = true)
 |-- RestingBP: integer (nullable = true)
 |-- Cholesterol: double (nullable = true)
 |-- FastingBS: integer (nullable = true)
 |-- RestingECG: string (nullable = true)
 |-- MaxHR: integer (nullable = true)
 |-- ExerciseAngina: string (nullable = true)
 |-- Oldpeak: double (nullable = true)
 |-- ST_Slope: string (nullable = true)
 |-- HeartDisease: integer (nullable = true)
```

Step 14: # Calculate the count of each data type and show the pie chart

```
data_types = [dtype[1] for dtype in heart.dtypes]

data_type_counts = {}
for dtype in data_types:
    data_type_counts[dtype] = data_type_counts.get(dtype, 0) + 1

data_types = list(data_type_counts.keys())
counts = list(data_type_counts.values())

explode = [0.1] * len(data_types)
plt.figure(figsize=(8, 8))
plt.pie(counts, explode=explode, labels=data_types, autopct='%1.2f%%', shadow=True)
plt.title('Type of our data', fontsize=20, color='DarkOrange', fontweight='bold', fontname='Lucida Calligraphy')
plt.legend(labels=data_types)
plt.axis('off')
plt.show()
```

```
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result



Step 15: Print the schema of the DataFrame to identify the names of numerical columns

```
heart.printSchema()
```

```
root
 |-- Age: integer (nullable = true)
 |-- Sex: string (nullable = true)
 |-- ChestPainType: string (nullable = true)
 |-- RestingBP: integer (nullable = true)
 |-- Cholesterol: double (nullable = true)
 |-- FastingBS: integer (nullable = true)
 |-- RestingECG: string (nullable = true)
 |-- MaxHR: integer (nullable = true)
 |-- ExerciseAngina: string (nullable = true)
 |-- Oldpeak: double (nullable = true)
 |-- ST_Slope: string (nullable = true)
 |-- HeartDisease: integer (nullable = true)
```

Step 16: Get the list of numerical column names

```
numerical = []
for col_name, col_type in heart.dtypes:
    if col_type in (IntegerType(), DoubleType()):
        numerical.append(col_name)

print(numerical)
```

Step 17: Get the list of categorical column names

```
categorical = []
for col_name, col_type in heart.dtypes:
    if col_type == StringType():
        categorical.append(col_name)

print(categorical)
```

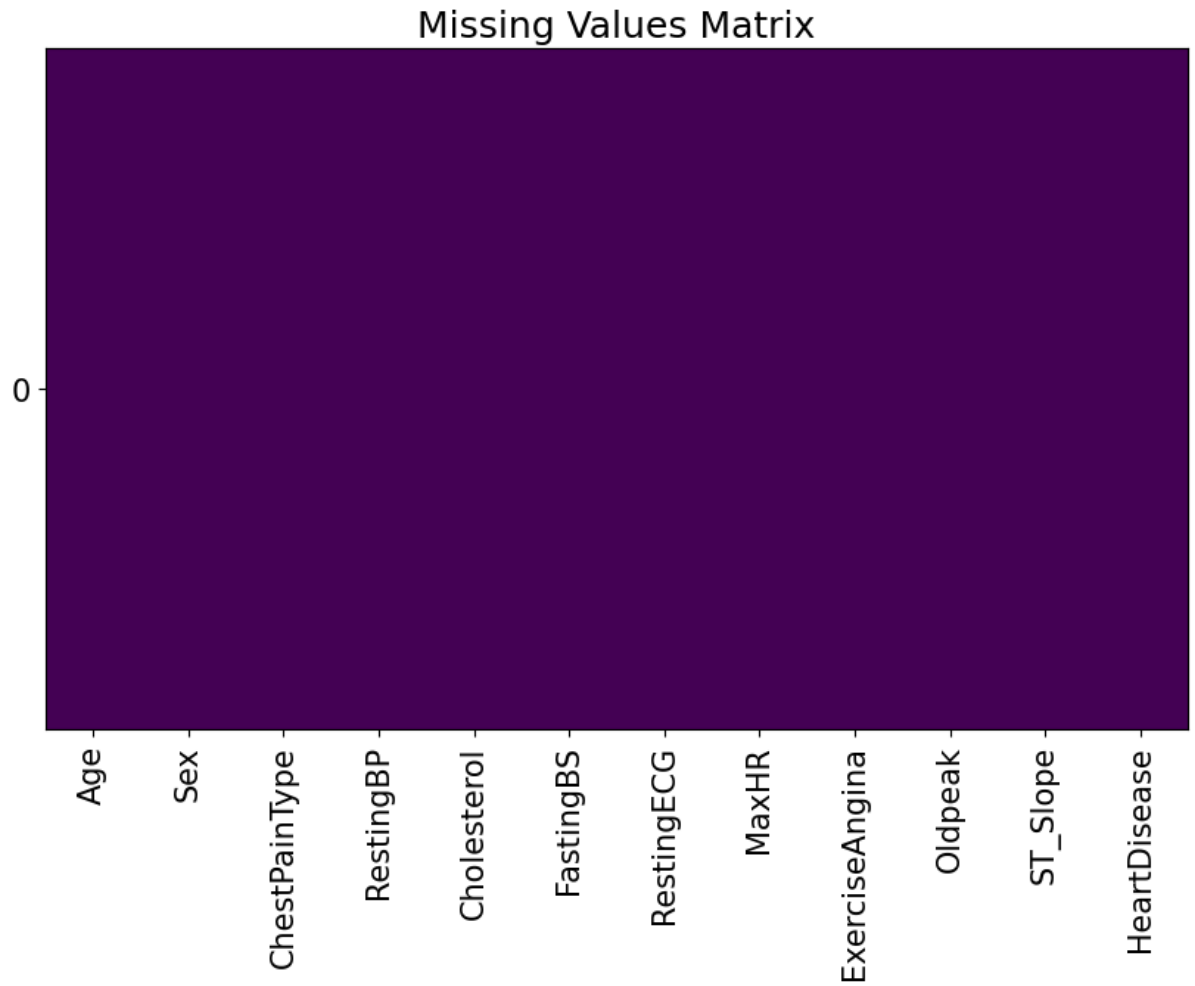
Step 18: Calculate the sum of null values in each column

```
null_counts = heart.select([col(column).isNull().alias(column) for column in heart.columns])\
                   .groupBy().sum().collect()

for column, count in zip(heart.columns, null_counts[0]):
    print(f"{column}: {count}")
```

Step 19: Calculate the count of null values in each column and show plot

```
missing_data = heart.select([spark_sum(col(column).isNull().cast("int")).alias(column) for column in heart.columns])

missing_df = missing_data.toPandas()

fig, ax = plt.subplots(figsize=(10, 6))
ax.imshow(missing_df.isnull(), cmap="viridis", aspect="auto")

ax.set_xticks(range(len(missing_df.columns)))
ax.set_xticklabels(missing_df.columns, rotation=90)
ax.set_yticks(range(len(missing_df)))
ax.set_yticklabels(missing_df.index)

plt.title("Missing Values Matrix")
plt.show()
```

Here is my result



Step 20: Iterate through the columns and print each column name

```
    for col in heart.columns:
        print(col)
```

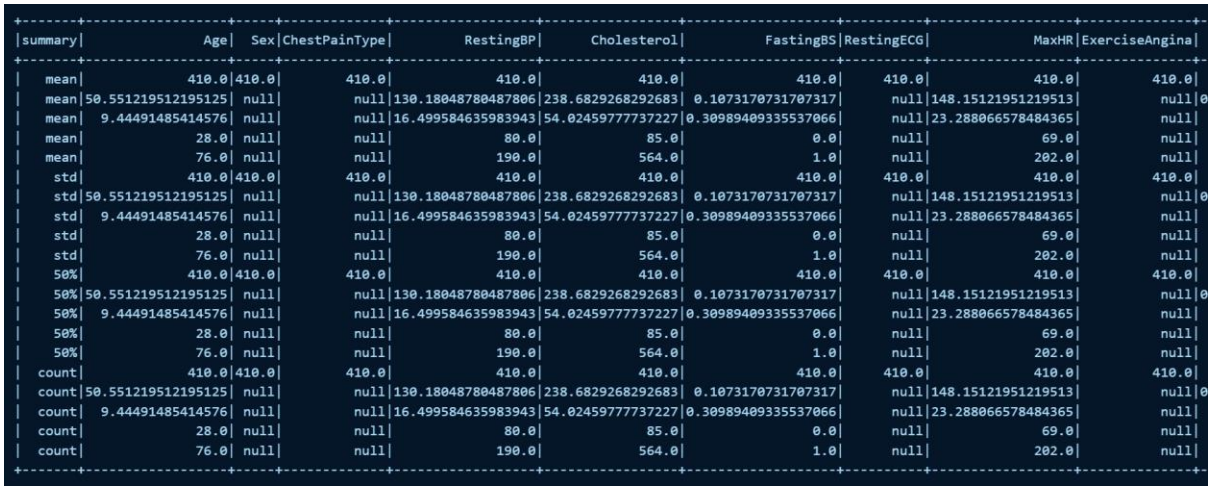
```
Age
Sex
ChestPainType
RestingBP
Cholesterol
FastingBS
RestingECG
MaxHR
ExerciseAngina
Oldpeak
ST_Slope
HeartDisease
```

Step 21: Filter the DataFrame where 'HeartDisease' is equal to 1

& Show the transposed DataFrame with background gradient

```
filtered_heart = heart.filter(col("HeartDisease") == 0)

statistics = filtered_heart.describe().cache()

statistics_transposed = statistics.select(*([lit('mean').alias('summary')] + [
    (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
])).union(
    statistics.select(*([lit('std').alias('summary')] + [
        (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
    ]))
).union(
    statistics.select(*([lit('50%').alias('summary')] + [
        (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
    ]))
).union(
    statistics.select(*([lit('count').alias('summary')] + [
        (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
    ]))
)


statistics_transposed.show()
```

Here is my result

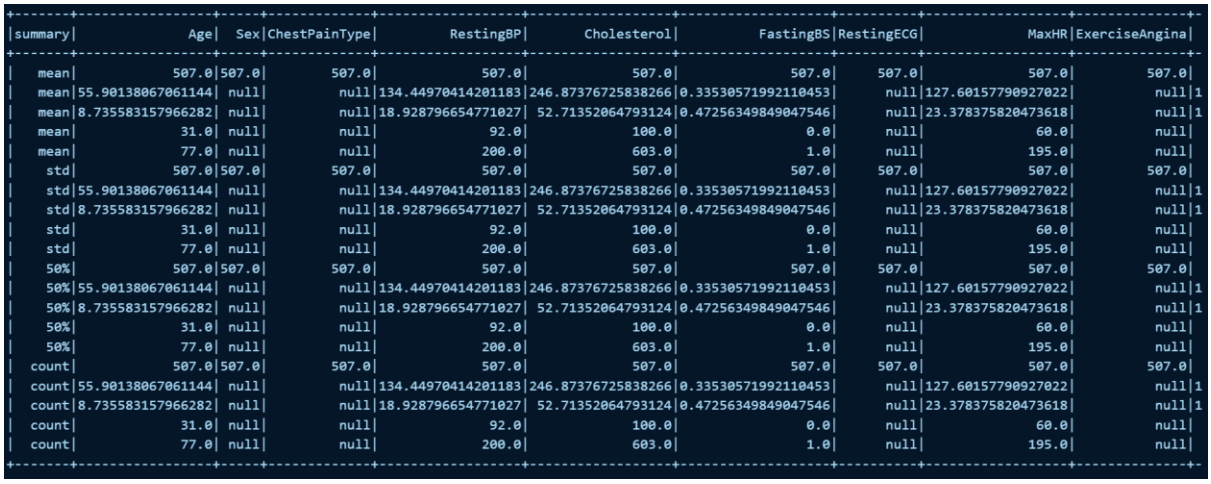


summary	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina
mean	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0
mean	50.551219512195125	null	null	130.18048780487806	238.6829268292683	0.1073170731707317	null	148.15121951219513	null
mean	9.44491485414576	null	null	16.499584635983943	54.02459777737227	0.30989409335537066	null	23.288066578484365	null
mean	28.0	null	null	80.0	85.0	0.0	null	69.0	null
mean	76.0	null	null	190.0	564.0	1.0	null	202.0	null
std	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0
std	50.551219512195125	null	null	130.18048780487806	238.6829268292683	0.1073170731707317	null	148.15121951219513	null
std	9.44491485414576	null	null	16.499584635983943	54.02459777737227	0.30989409335537066	null	23.288066578484365	null
std	28.0	null	null	80.0	85.0	0.0	null	69.0	null
std	76.0	null	null	190.0	564.0	1.0	null	202.0	null
50%	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0
50%	50.551219512195125	null	null	130.18048780487806	238.6829268292683	0.1073170731707317	null	148.15121951219513	null
50%	9.44491485414576	null	null	16.499584635983943	54.02459777737227	0.30989409335537066	null	23.288066578484365	null
50%	28.0	null	null	80.0	85.0	0.0	null	69.0	null
50%	76.0	null	null	190.0	564.0	1.0	null	202.0	null
count	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0	410.0
count	50.551219512195125	null	null	130.18048780487806	238.6829268292683	0.1073170731707317	null	148.15121951219513	null
count	9.44491485414576	null	null	16.499584635983943	54.02459777737227	0.30989409335537066	null	23.288066578484365	null
count	28.0	null	null	80.0	85.0	0.0	null	69.0	null
count	76.0	null	null	190.0	564.0	1.0	null	202.0	null

Step 22: Filter the DataFrame where 'HeartDisease' is equal to 1 and # Show the transposed DataFrame with background gradient

```
filtered_heart = heart.filter(col("HeartDisease") == 1)

statistics = filtered_heart.describe().cache()

statistics_transposed = statistics.select(*([lit('mean').alias('summary')] + [
    (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
])).union(
    statistics.select(*([lit('std').alias('summary')] + [
        (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
    ]))
).union(
    statistics.select(*([lit('50%').alias('summary')] + [
        (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
    ]))
).union(
    statistics.select(*([lit('count').alias('summary')] + [
        (col(c).cast('double')).alias(c) for c in statistics.columns[1:]
    ]))
)


statistics_transposed.show()
```

Here is my result



summary	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina
mean	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0
mean	55.90138067061144	null	null	134.44970414201183	246.87376725838266	0.3353057199211045 3	null	127.60157790927022	null	1
mean	8.735583157966282	null	null	18.928796654771027	52.71352064793124	0.4725634984904754 6	null	23.378375820473618	null	1
mean	31.0	null	null	92.0	100.0	0.0	null	60.0	null	
mean	77.0	null	null	200.0	603.0	1.0	null	195.0	null	
std	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0	
std	55.90138067061144	null	null	134.44970414201183	246.87376725838266	0.3353057199211045 3	null	127.60157790927022	null	1
std	8.735583157966282	null	null	18.928796654771027	52.71352064793124	0.4725634984904754 6	null	23.378375820473618	null	1
std	31.0	null	null	92.0	100.0	0.0	null	60.0	null	
std	77.0	null	null	200.0	603.0	1.0	null	195.0	null	
50%	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0	
50%	55.90138067061144	null	null	134.44970414201183	246.87376725838266	0.3353057199211045 3	null	127.60157790927022	null	1
50%	8.735583157966282	null	null	18.928796654771027	52.71352064793124	0.4725634984904754 6	null	23.378375820473618	null	1
50%	31.0	null	null	92.0	100.0	0.0	null	60.0	null	
50%	77.0	null	null	200.0	603.0	1.0	null	195.0	null	
count	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0	507.0	
count	55.90138067061144	null	null	134.44970414201183	246.87376725838266	0.3353057199211045 3	null	127.60157790927022	null	1
count	8.735583157966282	null	null	18.928796654771027	52.71352064793124	0.4725634984904754 6	null	23.378375820473618	null	1
count	31.0	null	null	92.0	100.0	0.0	null	60.0	null	
count	77.0	null	null	200.0	603.0	1.0	null	195.0	null	

Step 23: Calculate the summary statistics for the DataFrame, Find the minimum value for each column

```
summary = heart.describe()

min_values = summary.filter(col("summary") == "min")

min_values_list = min_values.collect()

def highlight_min_value(column_name, value):
    if isinstance(value, float):
        return when(col(column_name) == value, True).otherwise(False)
    else:
        return lit(False)

highlight_exprs = [
    highlight_min_value(column_name, float(row[column_name]))
    for row in min_values_list
    for column_name, dtype in zip(summary.columns[1:], heart.dtypes)
    if dtype[1] in ("int", "double")
]

highlighted_heart = heart.withColumn("highlight", lit(0))
for expr in highlight_exprs:
    highlighted_heart = highlighted_heart.withColumn(
        "highlight", when(expr, 1).otherwise(col("highlight"))
    )

highlighted_heart.show()
```

Here is my result

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+---------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|highlight|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+---------+
| 40|  M|          ATA|      140|      289.0|        0|    Normal|  172|             N|    0.0|      Up|           0|        1|
| 49|  F|          NAP|      160|      180.0|        0|    Normal|  156|             N|    1.0|    Flat|           1|        1|
| 37|  M|          ATA|      130|      283.0|        0|        ST|   98|             N|    0.0|      Up|           0|        1|
| 48|  F|          ASY|      138|      214.0|        0|    Normal|  108|             Y|    1.5|    Flat|           1|        1|
| 54|  M|          NAP|      150|      195.0|        0|    Normal|  122|             N|    0.0|      Up|           0|        1|
| 39|  M|          NAP|      120|      339.0|        0|    Normal|  170|             N|    0.0|      Up|           0|        1|
| 45|  F|          ATA|      130|      237.0|        0|    Normal|  170|             N|    0.0|      Up|           0|        1|
| 54|  M|          ATA|      110|      208.0|        0|    Normal|  142|             N|    0.0|      Up|           0|        1|
| 37|  M|          ASY|      140|      207.0|        0|    Normal|  130|             Y|    1.5|    Flat|           1|        1|
| 48|  F|          ATA|      120|      284.0|        0|    Normal|  120|             N|    0.0|      Up|           0|        1|
| 37|  F|          NAP|      130|      211.0|        0|    Normal|  142|             N|    0.0|      Up|           0|        1|
| 58|  M|          ATA|      136|      164.0|        0|        ST|   99|             Y|    2.0|    Flat|           1|        1|
| 39|  M|          ATA|      120|      204.0|        0|    Normal|  145|             N|    0.0|      Up|           0|        1|
| 49|  M|          ASY|      140|      234.0|        0|    Normal|  140|             Y|    1.0|    Flat|           1|        1|
| 42|  F|          NAP|      115|      211.0|        0|        ST|  137|             N|    0.0|      Up|           0|        1|
| 54|  F|          ATA|      120|      273.0|        0|    Normal|  150|             N|    1.5|    Flat|           0|        1|
| 38|  M|          ASY|      110|      196.0|        0|    Normal|  166|             N|    0.0|    Flat|           1|        1|
| 43|  F|          ATA|      120|      201.0|        0|    Normal|  165|             N|    0.0|      Up|           0|        1|
| 60|  M|          ASY|      100|      248.0|        0|    Normal|  125|             N|    1.0|    Flat|           1|        1|
| 36|  M|          ATA|      120|      267.0|        0|    Normal|  160|             N|    3.0|    Flat|           1|        1|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+---------+
only showing top 20 rows
```

Step 24: Calculate summary statistics for each categorical column

```python
categorical_columns = [column for column, dtype in heart.dtypes if dtype == "string"]

categorical_summary = heart
for column in categorical_columns:
    categorical_summary = categorical_summary.withColumn(f"Missing_{column}", col(column).isNull().cast("int"))
    categorical_summary = categorical_summary.withColumn(f"Count_{column}_F", (col(column) == "F").cast("int"))
    categorical_summary = categorical_summary.withColumn(f"Count_{column}_M", (col(column) == "M").cast("int"))
    categorical_summary = categorical_summary.withColumn(f"Count_{column}_Yes", (col(column) == "Yes").cast("int"))
    categorical_summary = categorical_summary.withColumn(f"Count_{column}_No", (col(column) == "No").cast("int"))

categorical_summary = categorical_summary.select(*categorical_columns,
                                                  *[f"Missing_{column}" for column in categorical_columns],
                                                  *[f"Count_{column}_F" for column in categorical_columns],
                                                  *[f"Count_{column}_M" for column in categorical_columns],
                                                  *[f"Count_{column}_Yes" for column in categorical_columns],
                                                  *[f"Count_{column}_No" for column in categorical_columns])

categorical_summary.show()
```

Here is my result

```
+---+-------------+---------+--------------+-------+----------+-------------------+-----------------+--------------------+---------------+---
|Sex|ChestPainType|RestingECG|ExerciseAngina|ST_Slope|Missing_Sex|Missing_ChestPainType|Missing_RestingECG|Missing_ExerciseAngina|Missing_ST_Slope|Cou
+---+-------------+---------+--------------+-------+----------+-------------------+-----------------+--------------------+---------------+---
|  M|          ATA|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  F|          NAP|   Normal|             N|   Flat|         0|                  0|                0|                   0|              0|
|  M|          ATA|       ST|             N|     Up|         0|                  0|                0|                   0|              0|
|  F|          ASY|   Normal|             Y|   Flat|         0|                  0|                0|                   0|              0|
|  M|          NAP|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  M|          NAP|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  F|          ATA|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  M|          ATA|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  M|          ASY|   Normal|             Y|   Flat|         0|                  0|                0|                   0|              0|
|  F|          ATA|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  F|          NAP|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  M|          ATA|       ST|             Y|   Flat|         0|                  0|                0|                   0|              0|
|  M|          ATA|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  M|          ASY|   Normal|             Y|   Flat|         0|                  0|                0|                   0|              0|
|  F|          NAP|       ST|             N|     Up|         0|                  0|                0|                   0|              0|
|  F|          ATA|   Normal|             N|   Flat|         0|                  0|                0|                   0|              0|
|  M|          ASY|   Normal|             N|   Flat|         0|                  0|                0|                   0|              0|
|  F|          ATA|   Normal|             N|     Up|         0|                  0|                0|                   0|              0|
|  M|          ASY|   Normal|             N|   Flat|         0|                  0|                0|                   0|              0|
|  M|          ATA|   Normal|             N|   Flat|         0|                  0|                0|                   0|              0|
+---+-------------+---------+--------------+-------+----------+-------------------+-----------------+--------------------+---------------+---
only showing top 20 rows
```

Step 25: Create a pipeline to apply all the transformations in order

```
string_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina", "ST_Slope"]
indexers = [StringIndexer(inputCol=col_name, outputCol=col_name + "_index", handleInvalid="keep")
            for col_name in string_cols]

encoder = OneHotEncoder(inputCols=[col_name + "_index" for col_name in string_cols],
                        outputCols=[col_name + "_encoded" for col_name in string_cols])

feature_cols = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"] + [col_name + "_encoded" for col_name in string_cols]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")


pipeline = Pipeline(stages=indexers + [encoder, assembler])
pipeline_model = pipeline.fit(heart)
encoded_heart = pipeline_model.transform(heart)

heart_vectorized = encoded_heart.select("features")
correlation_matrix = Correlation.corr(heart_vectorized, "features").head()[0]

correlation_df = spark.createDataFrame(
    correlation_matrix.toArray().tolist(),
    schema=["feature1", "feature2", "correlation"]
)

sorted_correlation_df = correlation_df \
    .withColumn("correlation", col("correlation").cast("double")) \
    .orderBy("correlation", ascending=False)

sorted_correlation_df.show()
```

Here is my result

```
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|            feature1|            feature2|         correlation|                  _4|                  _5|                  _6|                  _7|
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|0.045556145727709406| 0.08606246852427796|                 1.0|-0.00151007192132...| 0.05458314402490323|-0.11112150089013698| 0.11112150089013696|
|-0.05567006438588273|-0.00942672868180...| 0.11112150089013696| 0.18966836274630217|-0.10544446923827937|                -1.0|                 1.0|-
| 0.18549912487460138|  0.1101706868526033| 0.08748031459180552| -0.3436410117107143|  0.2828738424738372| 0.11559559904373086|-0.11559559904373085|
| 0.26308444022704613|                 1.0| 0.08606246852427796|-0.10969285221159106| 0.17425167137244033|0.009426728681801687|-0.00942672868180...|0
|  0.1458372130704135|  0.0506476668258099| 0.08428355251128529| 0.12624191895153192| 0.08714043038243313|-0.04924648453390435|0.049246484533904324|0
| 0.16690283964045108|0.041567557655027426| 0.07397121023677555|  -0.354443519198044| 0.28093568343430647| 0.18463087379881676|-0.18463087379881668|
| 0.21601738825867067| 0.15306417604243372|  0.0727506040327963|-0.37002279853998277|  0.4094938366525371|  0.1912259584176479| -0.1912259584176479|
|  0.2585626282953492| 0.17425167137244033| 0.05458314402490323|-0.16121252797352154|                 1.0|  0.1054444692382793|-0.10544446923827937|
|                 1.0| 0.26308444022704613|0.045556145727709406| -0.3822795417446569|  0.2585626282953492|0.055670064385882755|-0.05567006438588273|
| -0.3822795417446569|-0.10969285221159106|-0.00151007192132...|                 1.0|-0.16121252797352154|-0.18966836274630208| 0.18966836274630217|
|-0.21811299109880977|-0.05137387834278999|-0.00519876596508...| 0.25421401375406716|-0.26190266342172036|-0.16129434467979248| 0.16129434467979248|
| 0.13845121684784187|-0.01032309755031...|-0.01399334452224...| -0.0731274591363483|  0.3223709104493525| 0.06620102972368215|-0.06620102972368234|
| 0.13690272252374186| 0.08915647320834823|-0.03230396927333045|  -0.157561228779829|0.056283130370789435|  0.064008770886729|-0.06400877088672911|
| -0.2307925975477032|-0.11375882985240185|-0.04339163789901408|0.023181132342706596|-0.11729295015900128|-0.01109809850359...| 0.01109809850359468|
|0.032082572981474676| 0.04946436231663285|-0.04469382214558542| 0.10023494070509097| 0.03238204446517185|-0.00390253602229...|0.003902536022297...|-
|-0.01168081061326...|-0.02753602800029...|-0.06049609361750614|  0.1334108661006136|-0.10761467211717139|-0.06768191611656088| 0.06768191611656117|
|-0.21601738825867067|-0.15306417604243372| -0.0727506040327963| 0.37002279853998277| -0.4094938366525371| -0.1912259584176479|  0.1912259584176479|
| -0.2580272875652577|-0.10596943890088015|-0.08118346327584527|  0.3843394121002809| -0.4502987872758351|-0.15053696707006495| 0.15053696707006495|-
|0.055670064385882755|0.009426728681801687|-0.11112150089013698|-0.18966836274630208|  0.1054444692382793|                1.0|                -1.0|
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
```

Step 26: Create a pipeline to apply all the transformations in order

```
string_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina", "ST_Slope"]

indexers = [StringIndexer(inputCol=col_name, outputCol=col_name + "_index", handleInvalid="keep")
            for col_name in string_cols]

encoder = OneHotEncoder(inputCols=[col_name + "_index" for col_name in string_cols],
                        outputCols=[col_name + "_encoded" for col_name in string_cols])

feature_cols = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"] + [col_name + "_encoded" for col_name in string_cols]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")


pipeline = Pipeline(stages=indexers + [encoder, assembler])
pipeline_model = pipeline.fit(heart)
encoded_heart = pipeline_model.transform(heart)

correlation_matrix = Correlation.corr(encoded_heart, "features").head()[0]

correlation_df = spark.createDataFrame(
    correlation_matrix.toArray().tolist(),
    schema=encoded_heart.columns[-1:]
)

correlation_df.show(truncate=False)
```

Here is my result

features	_2	_3	_4	_5	_6	_7
1.0	0.26308444022704613	0.045556145727709406	-0.3822795417446569	0.2585626282953492	0.055670064385882755	-0.0556700643858
0.26308444022704613	1.0	0.08606246852427796	-0.10969285221159106	0.17425167137244033	0.009426728681801687	-0.0094267286818
0.045556145727709406	0.08606246852427796	1.0	-0.0015100719213286071	0.0545831440490323	-0.11112150089013698	0.11112150089013
-0.3822795417446569	-0.10969285221159106	-0.0015100719213286071	1.0	-0.16121252797352154	-0.18966836274630208	0.18966836274630
0.2585626282953492	0.17425167137244033	0.05458314402490323	-0.16121252797352154	1.0	0.1054444692382793	-0.1054444692382
0.055670064385882755	0.009426728681801687	-0.11112150089013698	-0.18966836274630208	0.1054444692382793	1.0	-1.0
-0.055670064385882735	-0.009426728681801739	0.11112150089013696	0.18966836274630217	-0.10544446923827937	-1.0	1.0
0.16690283964045108	0.041567557655027426	0.07397121023677555	-0.354443519198044	0.28093568343430647	0.18463087379881676	-0.1846308737988
-0.011680810613269695	-0.027536028000298213	-0.06049609361750614	0.1334108661006136	-0.10761467211717139	-0.06768191611656088	0.06768191611656
-0.21811299109880977	-0.05137387834278999	-0.005198765965088025	0.25421401375406716	-0.26190266342172036	-0.16129434467979248	0.16129434467979
0.032082572981474676	0.04946436231663285	-0.04469382214558542	0.10023494070509097	0.03238204446517185	-0.0039025360222973786	0.00390253602229
-0.2307925975477032	-0.11375882985240185	-0.04339163789901408	0.023181132342706596	-0.11729295015900128	-0.011098098503594762	0.01109809850359
0.1458372130704135	0.0506476668258099	0.08428355251128529	0.12624191895153192	0.08714043038243313	-0.04924648453390435	0.04924648453390
0.13690272252374186	0.08915647320834823	-0.032303969273330045	-0.157561228779829	0.05628313037078943 5	0.064008770886729	-0.0640087708867
-0.21601738825867067	-0.15306417604243372	-0.0727506040327963	0.37002279853998277	-0.4094938366525371	-0.1912259584176479	0.19122595841764
0.21601738825867067	0.15306417604243372	0.0727506040327963	-0.37002279853998277	0.4094938366525371	0.1912259584176479	-0.1912259584176
0.18549912487460138	0.1101706868526033	0.08748031459180552	-0.3436410117107143	0.2828738424738372	0.11555955990437086	-0.1155955990437
-0.2580272875652577	-0.10596943890088015	-0.08118346327584527	0.3843394121002809	-0.4502987872758351	-0.15053696707006495	0.15053696707006
0.13845121684784187	-0.010323097550312941	-0.013993344522240663	-0.0731274591363483	0.3223709104493525	0.06620102972368215	-0.0662010297236

Step 27: Create a pipeline to apply all the transformations in order

```
string_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina", "ST_Slope"]

indexers = [StringIndexer(inputCol=col_name, outputCol=col_name + "_index", handleInvalid="keep")
            for col_name in string_cols]

encoder = OneHotEncoder(inputCols=[col_name + "_index" for col_name in string_cols],
                        outputCols=[col_name + "_encoded" for col_name in string_cols])

feature_cols = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"] + [col_name + "_encoded" for col_name in string_cols]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")


pipeline = Pipeline(stages=indexers + [encoder, assembler])
pipeline_model = pipeline.fit(heart)
encoded_heart = pipeline_model.transform(heart)

correlation_matrix = Correlation.corr(encoded_heart, "features").head()[0]

correlation_df = spark.createDataFrame(
    correlation_matrix.toArray().tolist(),
    schema=encoded_heart.columns[-1:]
)

correlation_df.show(truncate=False)
```

Here is my result

```
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-------------
|features            |_2                  |_3                  |_4                  |_5                  |_6                  |_7
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-------------
|1.0                 |0.26308444022704613 |0.045556145727709406|-0.3822795417446569 |0.2585626282953492  |0.055670064385882755|-0.0556700643
|0.26308444022704613 |1.0                 |0.08606246852427796 |-0.10969285221159106|0.17425167137244033 |0.009426728681801687|-0.0094267286
|0.045556145727709406|0.08606246852427796 |1.0                 |-0.0015100719213286071|0.05458314402490323 |-0.11112150089013698|0.111121500890
|-0.3822795417446569 |-0.10969285221159106|-0.0015100719213286071|1.0                 |-0.16121252797352154|-0.18966836274630208|0.189668362746
|0.2585626282953492  |0.17425167137244033 |0.05458314402490323 |-0.16121252797352154|1.0                 |0.1054444692382793  |-0.1054444692
|0.055670064385882755|0.009426728681801687|-0.11112150089013698|-0.18966836274630208|0.1054444692382793  |1.0                 |-1.0
|-0.055670064385882273|-0.009426728681801739|0.11112150089013696 |0.18966836274630217 |-0.1054444692382793 |-1.0                |1.0
|0.16690283964045108 |0.041567557655027426|0.07397121023677555 |-0.354443519198044  |0.28093568343430647 |0.18463087379881676 |-0.1846308737
|-0.011680810613269695|-0.027536028000298213|-0.06049609361750614|0.1334108661006136  |-0.10761467211717139|-0.06768191611656088|0.067681916116
|-0.21811299109880977|-0.05137387834278999|-0.005198765965088025|0.25421401375406716 |-0.26190266342172036|-0.16129434467979248|0.161294344679
|0.032082572981474676|0.04946436231663285 |-0.04469382214558542|0.10023494070509097 |0.03238204446517185 |-0.0039025360222973786|0.003902536022
|-0.2307925975477032 |-0.11375882985240185|-0.04339163789901408|0.023181132342706596|-0.11729295015900128|-0.011098098503594762|0.011098098503
|0.1458372130704135  |0.0506476668258099  |0.08428355251128529 |0.12624191895153192 |0.08714043038243313 |-0.04924648453390435|0.049246484533
|0.13690272252374186 |0.08915647320834823 |-0.03230396927333045|-0.157561228779829  |0.056283130370789435|0.064008770886729   |-0.0640087708
|-0.21601738825867067|-0.15306417604243372|-0.0727506040327963 |0.37002279853998277 |-0.4094938366525371 |-0.1912259584176479 |0.191225958417
|0.21601738825867067 |0.15306417604243372 |0.0727506040327963  |-0.37002279853998277|0.4094938366525371  |0.1912259584176479  |-0.1912259584
|0.18549912487460138 |0.1101706868526033  |0.08748031459180552 |-0.3436410117107143 |0.2828738424738372  |0.11559559904373086 |-0.1155955990
|-0.2580272875652577 |-0.10596943890088015|-0.08118346327584527|0.3843394121002809  |-0.4502987872758351 |-0.15053696707006495|0.150536967070
|0.13845121684784187 |-0.010323097550312941|-0.013993344522240663|-0.0731274591363483 |0.3223709104493525  |0.06620102972368215 |-0.0662010297
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+-------------
```

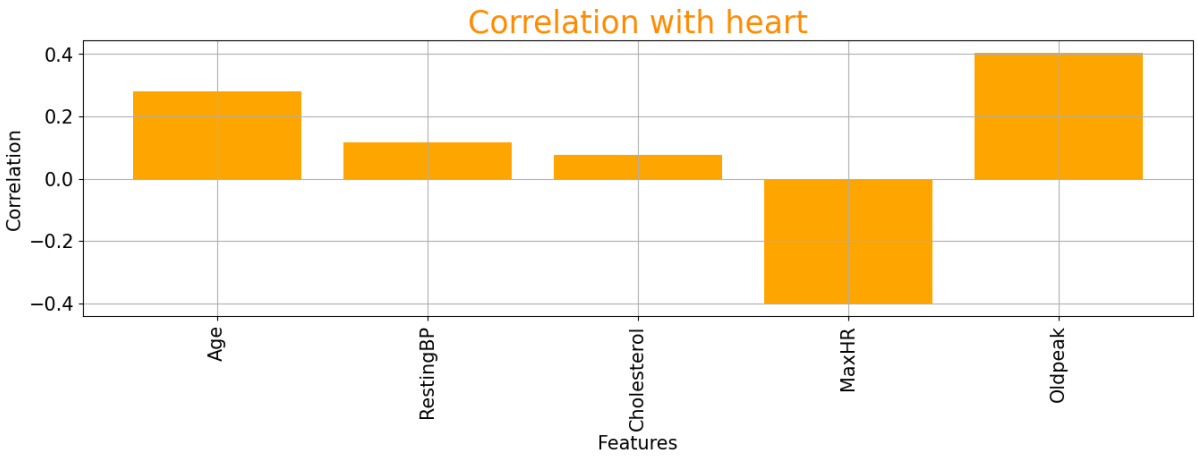
Step 28: Plot the correlation as a bar plot

```
string_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina", "ST_Slope"]

indexers = [StringIndexer(inputCol=col_name, outputCol=col_name + "_index", handleInvalid="keep")
            for col_name in string_cols]

encoder = OneHotEncoder(inputCols=[col_name + "_index" for col_name in string_cols],
                        outputCols=[col_name + "_encoded" for col_name in string_cols])

numerical_cols = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]
assembler_numerical = VectorAssembler(inputCols=numerical_cols, outputCol="numerical_features")

feature_cols = ["numerical_features"] + [col_name + "_encoded" for col_name in string_cols]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

pipeline = Pipeline(stages=indexers + [encoder, assembler_numerical, assembler])
pipeline_model = pipeline.fit(heart)
encoded_heart = pipeline_model.transform(heart)

encoded_heart = encoded_heart.drop("features", "numerical_features")

correlation_with_heart = {}
for col_name in numerical_cols:
    correlation = encoded_heart.stat.corr(col_name, "HeartDisease")
    correlation_with_heart[col_name] = correlation
```

```
correlation_df.show()

pandas_correlation_df = correlation_df.toPandas()

plt.figure(figsize=(16, 4))
plt.bar(pandas_correlation_df["Feature"], pandas_correlation_df["Correlation"], color='orange')
plt.xticks(rotation=90)
plt.title('Correlation with heart', fontsize=25, color='DarkOrange', fontname='Lucida Calligraphy')
plt.xlabel('Features')
plt.ylabel('Correlation')
plt.grid(True)
plt.show()
```

Here is my result

```
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
+-----------+--------------------+
|    Feature|         Correlation|
+-----------+--------------------+
|        Age|  0.2820117259633892|
|  RestingBP| 0.11799000562033152|
|Cholesterol| 0.07626169199449696|
|      MaxHR|-0.40140961436594785|
|    Oldpeak|  0.4036380155169544|
+-----------+--------------------+

findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
```

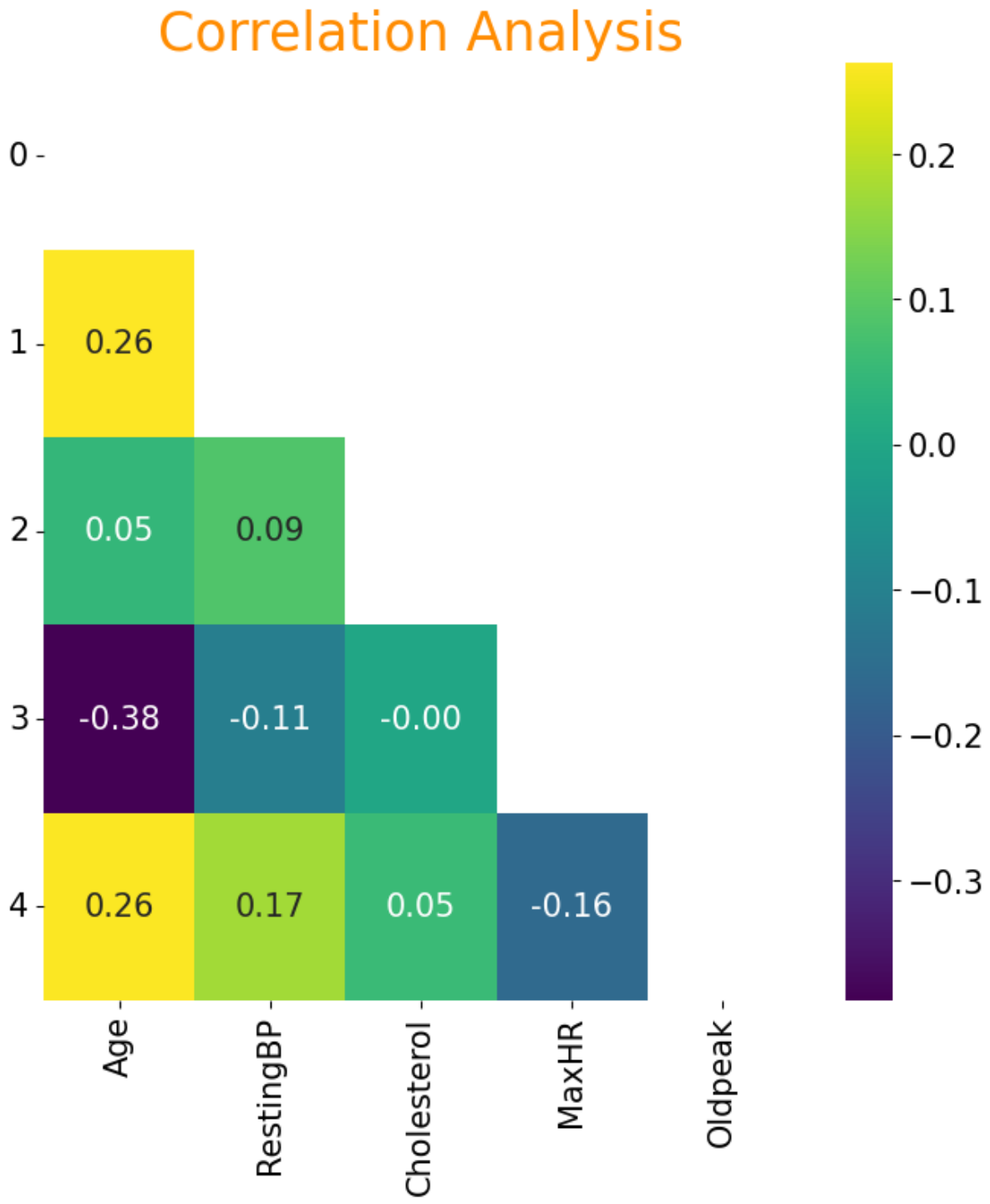


Step 29: Create a mask to hide the upper triangle of the heatmap

```
numerical_cols = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]

assembler = VectorAssembler(inputCols=numerical_cols, outputCol="numerical_features")
heart_vectorized = assembler.transform(heart)

correlation_matrix = Correlation.corr(heart_vectorized, "numerical_features").collect()[0][0]

corr_df = spark.createDataFrame(correlation_matrix.toArray(), numerical_cols).toPandas()
mask = np.triu(np.ones_like(corr_df, dtype=bool))

plt.figure(dpi=100)
plt.title('Correlation Analysis', fontsize=25, color='DarkOrange', fontname='Lucida Calligraphy')
sns.heatmap(corr_df, mask=mask, annot=True, fmt="0.2f", cmap='viridis', lw=0, linecolor='white')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```

```
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result



Step 30: Print value counts for each categorical column

```
categorical_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina", "ST_Slope"]

for col_name in categorical_cols:
    print(f"Value counts for '{col_name}':")
    heart.groupBy(col(col_name)).count().show()
    print("****" * 10)
```

Here is my result

```
Value counts for 'Sex':
+---+-----+
|Sex|count|
+---+-----+
|  F|  193|
|  M|  724|
+---+-----+

**************************************
Value counts for 'ChestPainType':
+-------------+-----+
|ChestPainType|count|
+-------------+-----+
|          NAP|  202|
|          ATA|  173|
|           TA|   46|
|          ASY|  496|
+-------------+-----+

**************************************
Value counts for 'RestingECG':
+----------+-----+
|RestingECG|count|
+----------+-----+
|       LVH|  188|
...
|    Down|   63|
+--------+-----+

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

Step 31: Find columns with skewness exceeding the threshold

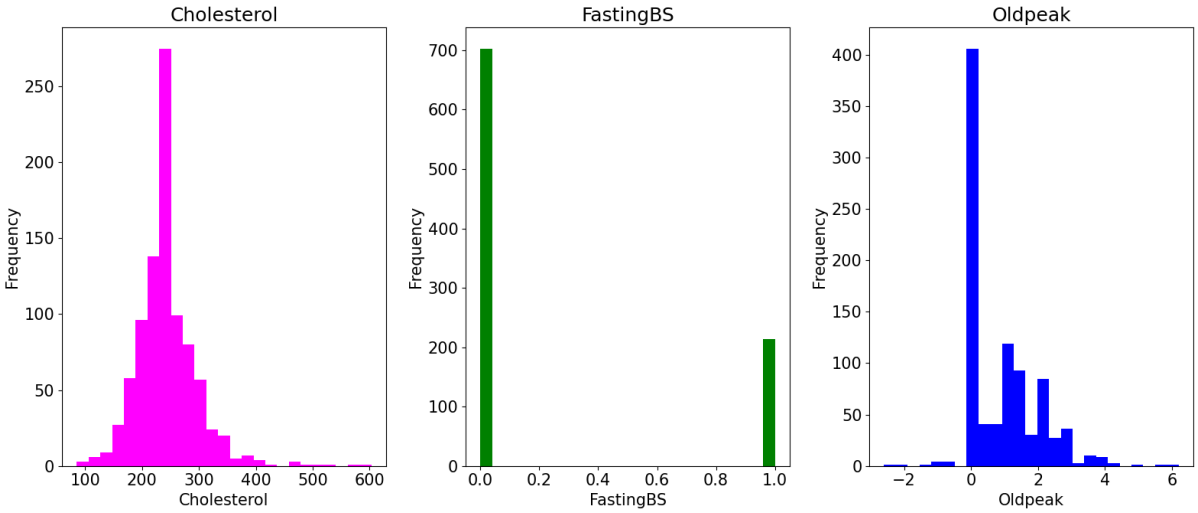
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import expr


numerical_cols = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]  # Update the ST_Depression to Oldpeak

skew_limit = 0.75

skew_vals = heart.agg(*(expr('skewness({}) as {}'.format(col, col)) for col in numerical_cols)).collect()[0].asDict()

skew_cols = {col: skew_vals[col] for col in numerical_cols if abs(skew_vals[col]) > skew_limit}
skew_cols = sorted(skew_cols.items(), key=lambda x: x[1], reverse=True)

for col, skewness in skew_cols:
    print(f"{col}: {skewness}")
```

```
Cholesterol: 1.443496892197544
Oldpeak: 1.0228679540445653
```

Step 32: See frequency of each column

```python
import matplotlib.pyplot as plt
from pyspark_dist_explore import hist
from pyspark.sql.functions import col


plt.figure(figsize=(16, 7))

plt.subplot(131)
plt.hist(heart.select("Cholesterol").rdd.flatMap(lambda x: x).collect(), bins=25, color="magenta")
plt.title('Cholesterol')
plt.xlabel("Cholesterol")
plt.ylabel("Frequency")

plt.subplot(132)
plt.hist(heart.select("FastingBS").rdd.flatMap(lambda x: x).collect(), bins=25, color="green")
plt.title('FastingBS')
plt.xlabel("FastingBS")
plt.ylabel("Frequency")

plt.subplot(133)
plt.hist(heart.select("Oldpeak").rdd.flatMap(lambda x: x).collect(), bins=25, color="blue")
plt.title('Oldpeak')
plt.xlabel("Oldpeak")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()
```

Here is my result



Step 33: Calculate the percentage count of each unique value in the feature

```
def percent_counts(df, feature):
    total_counts = df.groupBy(feature).agg(count("*").alias("Total"))
    total_records = df.count()
    percent_counts = total_counts.withColumn("Percentage", round(col("Total") / total_records * 100, 2))

    return percent_counts
```

Step 34: Show result

```
result = percent_counts(heart, "Cholesterol")

result.show()
```

```
+-----------+-----+----------+
|Cholesterol|Total|Percentage|
+-----------+-----+----------+
|      305.0|    4|      0.44|
|      299.0|    2|      0.22|
|      184.0|    4|      0.44|
|      147.0|    2|      0.22|
|      170.0|    2|      0.22|
|      160.0|    6|      0.65|
|      169.0|    2|      0.22|
|      311.0|    2|      0.22|
|      168.0|    2|      0.22|
|      206.0|    3|      0.33|
|      365.0|    1|      0.11|
|      249.0|    5|      0.55|
|      142.0|    1|      0.11|
|      329.0|    1|      0.11|
|      232.0|    3|      0.33|
|      303.0|    4|      0.44|
|      253.0|    4|      0.44|
|      331.0|    1|      0.11|
|      201.0|    6|      0.65|
|      235.0|    5|      0.55|
+-----------+-----+----------+
only showing top 20 rows
```

Step 35: Calculate the percentage count of each unique value in the feature

```
def percent_counts(df, feature):
    total_counts = df.groupBy(feature).agg(count("*").alias("Total"))
    total_records = df.count()
    percent_counts = total_counts.withColumn("Percentage", round(col("Total") / total_records * 100, 2))

    return percent_counts
```

Step 36: Get the percentage counts for the 'Sex' feature

```
result = percent_counts(heart, 'Sex')
result_pd = result.toPandas()

styled_result = result_pd.style.background_gradient(cmap='coolwarm').set_precision(2)

styled_result
```

```
/tmp/ipykernel_3101/1102230461.py:10: FutureWarning: this method is deprecated in favour of `Styler.format(precision=..)`
  styled_result = result_pd.style.background_gradient(cmap='coolwarm').set_precision(2)
```

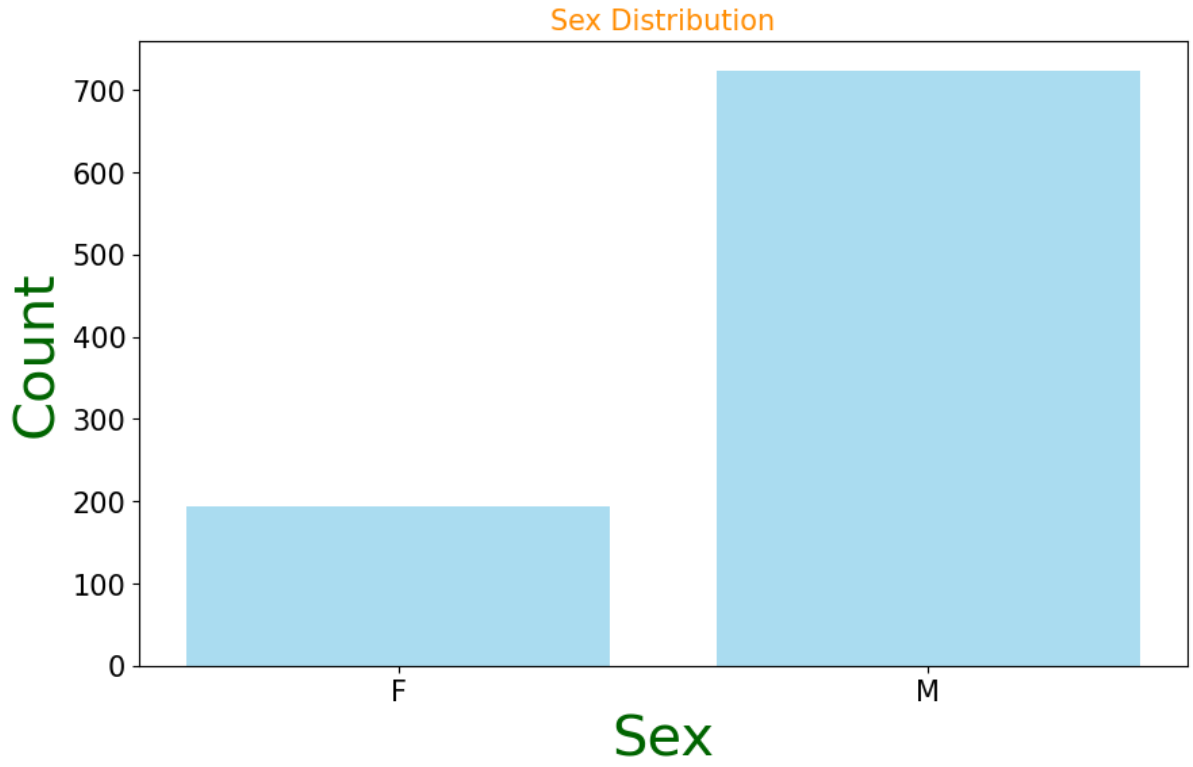
	Sex	Total	Percentage
0	F	193	21.05
1	M	724	78.95

Step 37: Create the count plot using matplotlib

```
sex_counts = heart.groupBy('Sex').agg(count('*').alias('Count'))
sex_counts_pd = sex_counts.toPandas()

plt.figure(figsize=(10, 6))
plt.bar(sex_counts_pd['Sex'], sex_counts_pd['Count'], color='skyblue', alpha=0.7)
plt.title('Sex Distribution', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.xlabel('Sex', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')
plt.ylabel('Count', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result

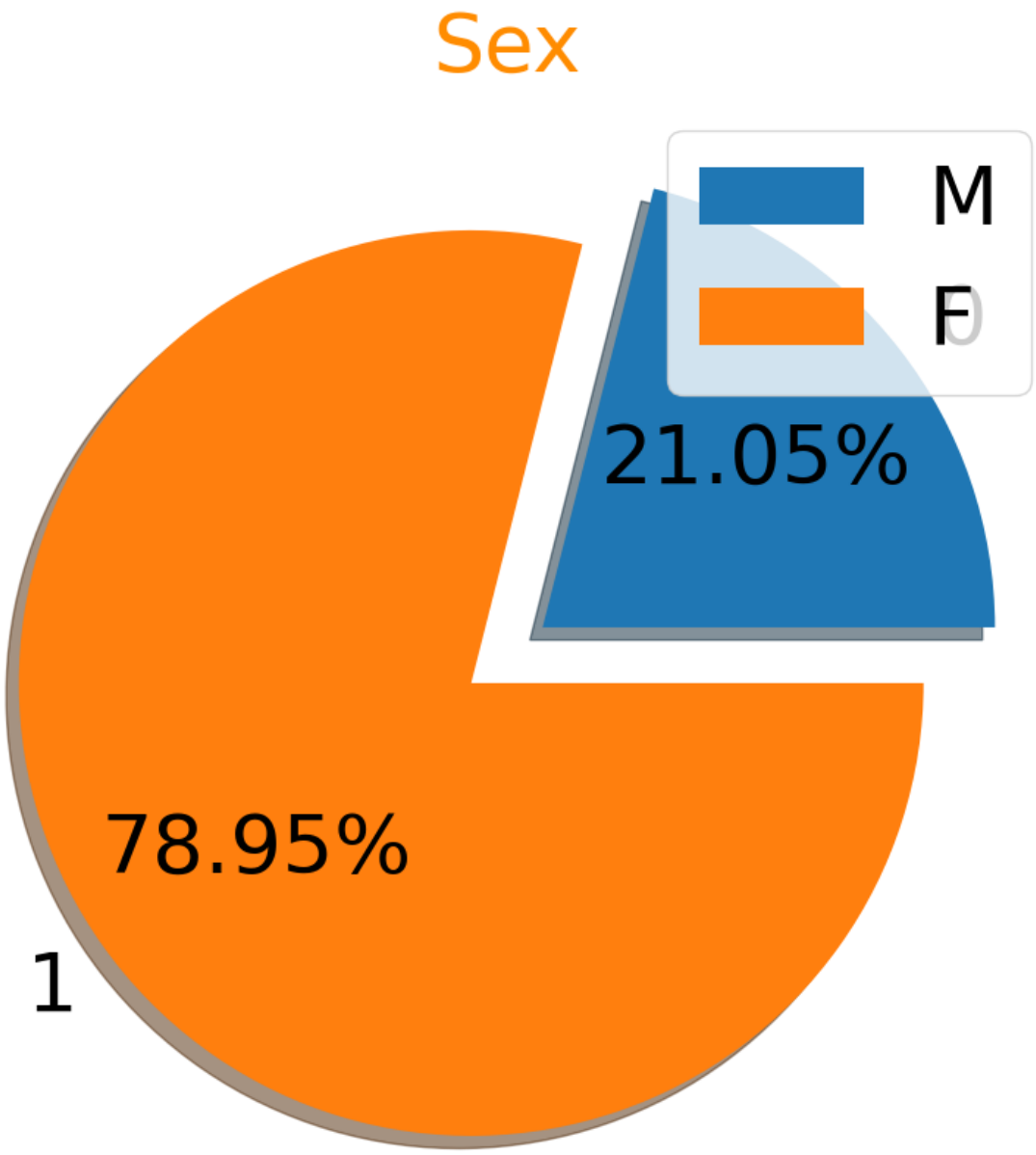


Step 38: Calculate the count of each value in the 'Sex' column

```
sex_counts = heart.groupBy('Sex').agg(count('*').alias('Count'))

sex_counts_pd = sex_counts.toPandas()

plt.rcParams.update({'font.size': 40})

plt.figure(figsize=(10, 10))
ax = sex_counts_pd['Count'].plot.pie(explode=[0.1, 0.1], autopct='%1.2f%%', shadow=True)
ax.set_title("Sex", fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.legend(labels=['M', 'F'])
plt.axis('off')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result



Step 39: Get the percentage counts for 'ChestPainType'

```python
def percent_counts(df, feature):
    total_counts = df.groupBy(feature).agg(count('*').alias('Total'))
    percent_counts = df.groupBy(feature).agg(round((count('*') / df.count()) * 100, 2).alias('Percentage'))

    result = total_counts.join(percent_counts, on=feature)
    result_pd = result.toPandas()

    return result_pd

chest_pain_percent_counts = percent_counts(heart, 'ChestPainType')

chest_pain_percent_counts.style.background_gradient(cmap='coolwarm').set_precision(2)
```

```
/tmp/ipykernel_3101/379936201.py:16: FutureWarning: this method is deprecated in favour of `Styler.format(precision=..)`
  chest_pain_percent_counts.style.background_gradient(cmap='coolwarm').set_precision(2)
```

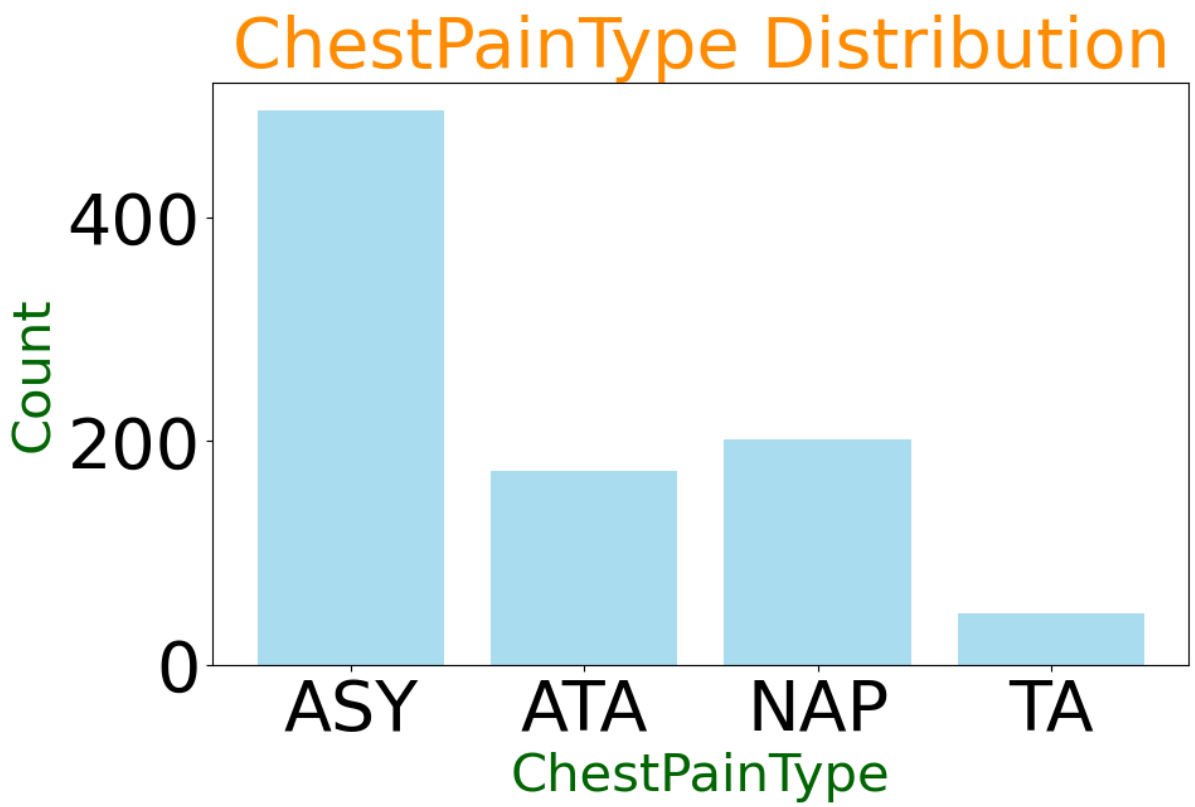
	ChestPainType	Total	Percentage
0	NAP	202	22.03
1	ATA	173	18.87
2	TA	46	5.02
3	ASY	496	54.09

Step 40: Plot the histogram using Matplotlib

```python
chest_pain_counts = heart.groupBy('ChestPainType').count().orderBy('ChestPainType')

chest_pain_types = chest_pain_counts.select('ChestPainType').rdd.flatMap(lambda x: x).collect()
chest_pain_counts = chest_pain_counts.select('count').rdd.flatMap(lambda x: x).collect()

plt.figure(figsize=(10, 6))
plt.bar(chest_pain_types, chest_pain_counts, color='skyblue', alpha=0.7)
plt.title('ChestPainType Distribution', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.xlabel('ChestPainType', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')
plt.ylabel('Count', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result

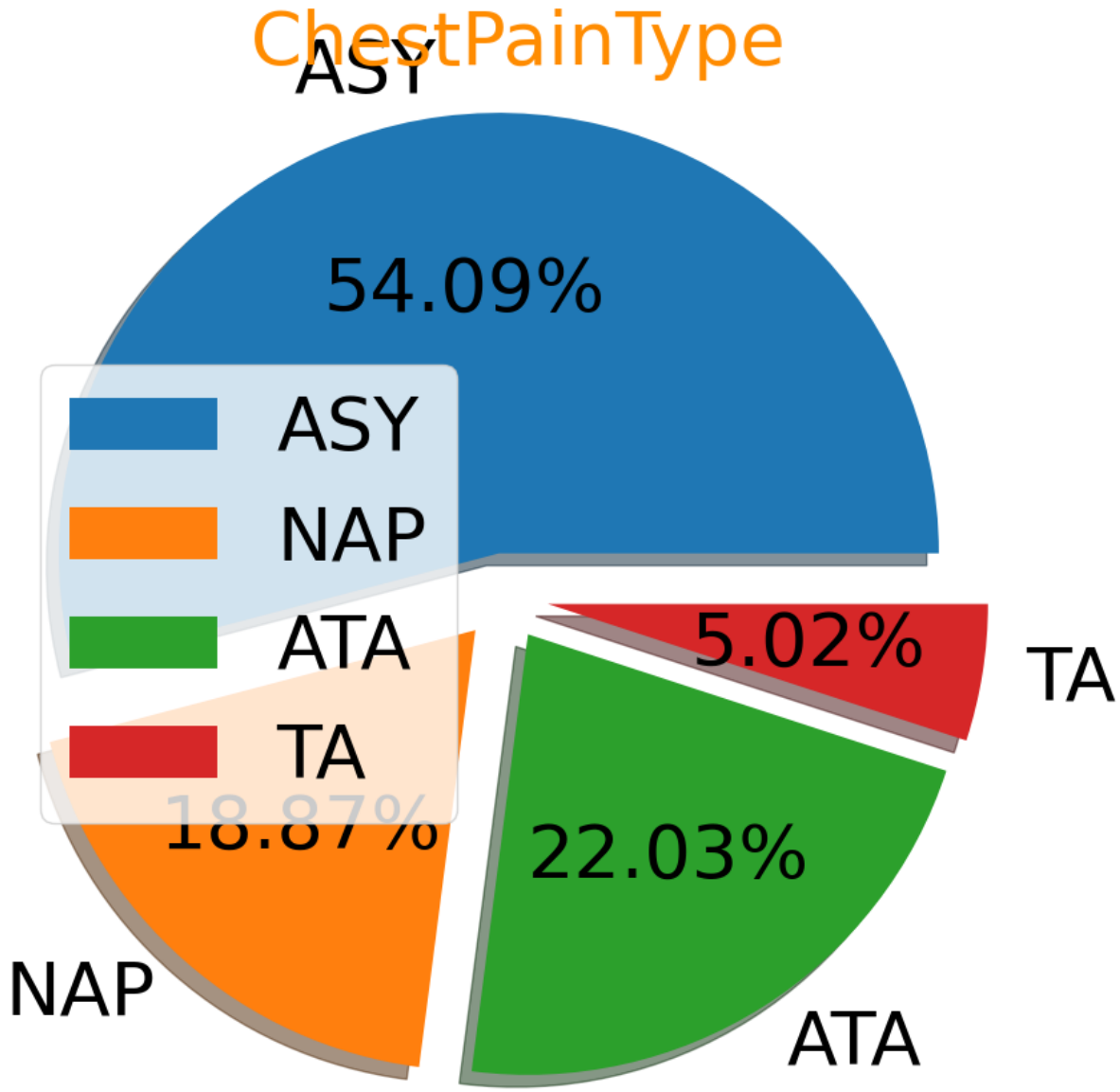


Step 41: Plot the pie chart using Matplotlib

```
chest_pain_counts = heart.groupBy('ChestPainType').agg(count('*').alias('Count')).orderBy('ChestPainType')

chest_pain_counts_pd = chest_pain_counts.toPandas()

plt.figure(figsize=(10, 10))
explode = [0.1, 0.1, 0.1, 0.1]
plt.pie(chest_pain_counts_pd['Count'], labels=['ASY', 'NAP', 'ATA', 'TA'], explode=explode, autopct='%1.2f%%', shadow=True)
plt.title('ChestPainType', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')

plt.axis('equal')
plt.legend(labels=['ASY', 'NAP', 'ATA', 'TA'])
plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result:



Step 42: Calculate total counts and percentages for the given feature

```python
from pyspark.sql.functions import col, count, round

def percent_counts(df, feature):

    total_counts = df.groupBy(feature).agg(count('*').alias('Total'))
    total_counts_pd = total_counts.toPandas()
    total_counts_pd.set_index(feature, inplace=True)

    total_counts_pd['Percentage'] = (total_counts_pd['Total'] / df.count()) * 100
    total_counts_pd['Percentage'] = total_counts_pd['Percentage'].round(2)

    total_counts_pd = total_counts_pd.sort_values(by='Total', ascending=False)

    return total_counts_pd
```

```python
resting_ecg_counts = percent_counts(heart, 'RestingECG')
resting_ecg_counts.style.background_gradient(cmap='coolwarm').set_precision(2)
```

```
/tmp/ipykernel_3101/3619607633.py:2: FutureWarning: this method is deprecated in favour of `Styler.format(precision=..)`
  resting_ecg_counts.style.background_gradient(cmap='coolwarm').set_precision(2)
```

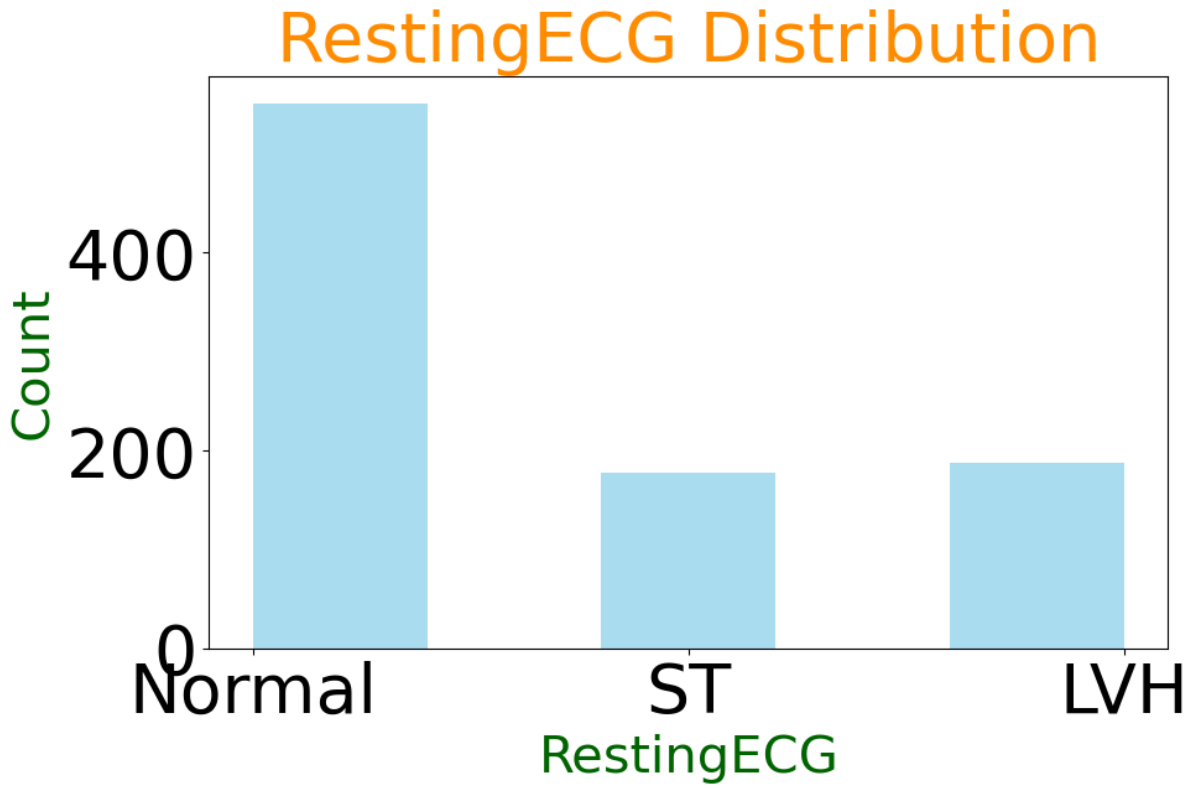
RestingECG	Total	Percentage
Normal	551	60.09
LVH	188	20.50
ST	178	19.41

Step 43: Create the histogram using matplotlib

```python
heart_pd = heart.select("RestingECG").toPandas()

plt.figure(figsize=(10, 6))
plt.hist(heart_pd['RestingECG'], bins=5, color='skyblue', alpha=0.7)
plt.title('RestingECG Distribution', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.xlabel('RestingECG', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')
plt.ylabel('Count', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result:

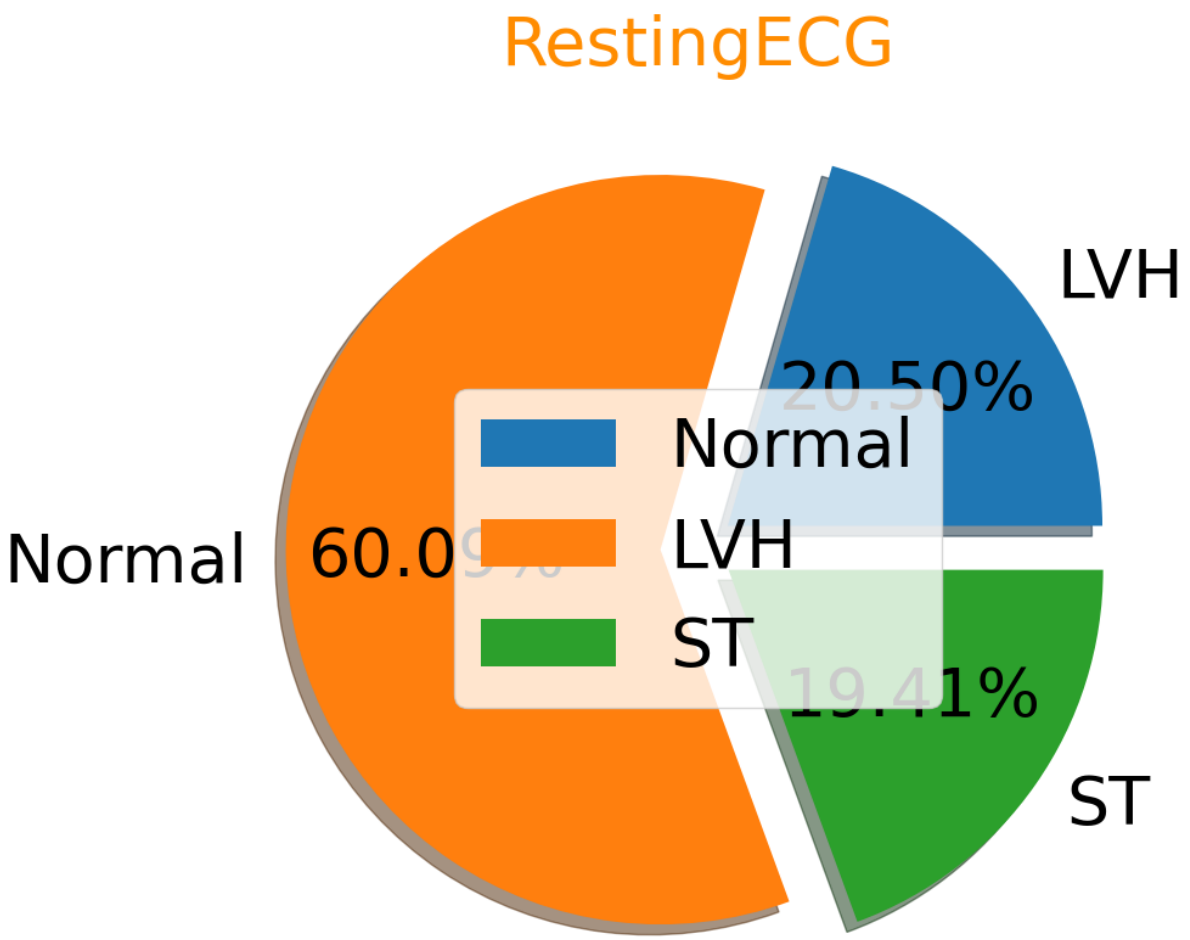


Step 44: Plot the pie chart using matplotlib

```
resting_ecg_counts = heart.groupBy('RestingECG').count().orderBy('RestingECG')

resting_ecg_counts_pd = resting_ecg_counts.toPandas()

plt.figure(figsize=(10, 10))
plt.pie(resting_ecg_counts_pd['count'], labels=resting_ecg_counts_pd['RestingECG'], explode=[0.1, 0.1, 0.1], autopct='%1.2f%%', shadow=True)
plt.title('RestingECG', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.legend(labels=['Normal', 'LVH', 'ST'])
plt.axis('off')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result:



Step 45: Calculate the counts for each unique value in 'ExerciseAngina'

```
exercise_angina_counts = heart.groupBy('ExerciseAngina').count()
total_rows = heart.count()
exercise_angina_counts = exercise_angina_counts.withColumn('Percentage', F.round(exercise_angina_counts['count'] / total_rows * 100, 2))

exercise_angina_counts_pd = exercise_angina_counts.toPandas()

exercise_angina_counts_pd.rename(columns={'ExerciseAngina': 'Value', 'count': 'Total', 'Percentage': 'Percentage'}, inplace=True)
exercise_angina_counts_pd['Percentage'] = exercise_angina_counts_pd['Percentage'].map('{:.2f}%'.format)
exercise_angina_counts_pd['Percentage'] = exercise_angina_counts_pd['Percentage'].str.replace('%', '').astype(float)  # Remove '%' and convert to

exercise_angina_counts_pd.style.background_gradient(cmap='coolwarm', subset=['Total', 'Percentage'], low=0, high=100)
```

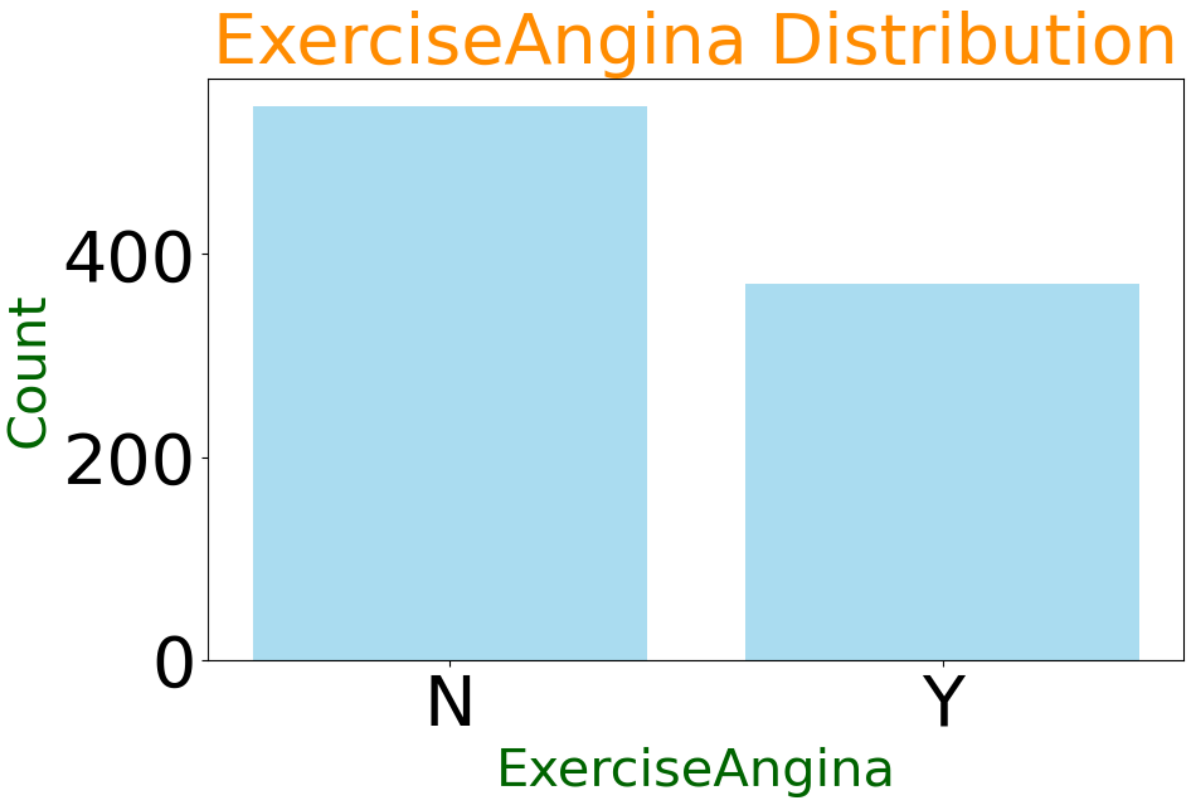
	Value	Total	Percentage
0	Y	371	40.460000
1	N	546	59.540000

Step 46: Plot the count plot using matplotlib

```python
exercise_angina_counts = heart.groupBy('ExerciseAngina').count().orderBy('ExerciseAngina')

exercise_angina_counts_pd = exercise_angina_counts.toPandas()

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(exercise_angina_counts_pd['ExerciseAngina'], exercise_angina_counts_pd['count'], color='skyblue', alpha=0.7)
plt.title('ExerciseAngina Distribution', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.xlabel('ExerciseAngina', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')
plt.ylabel('Count', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result

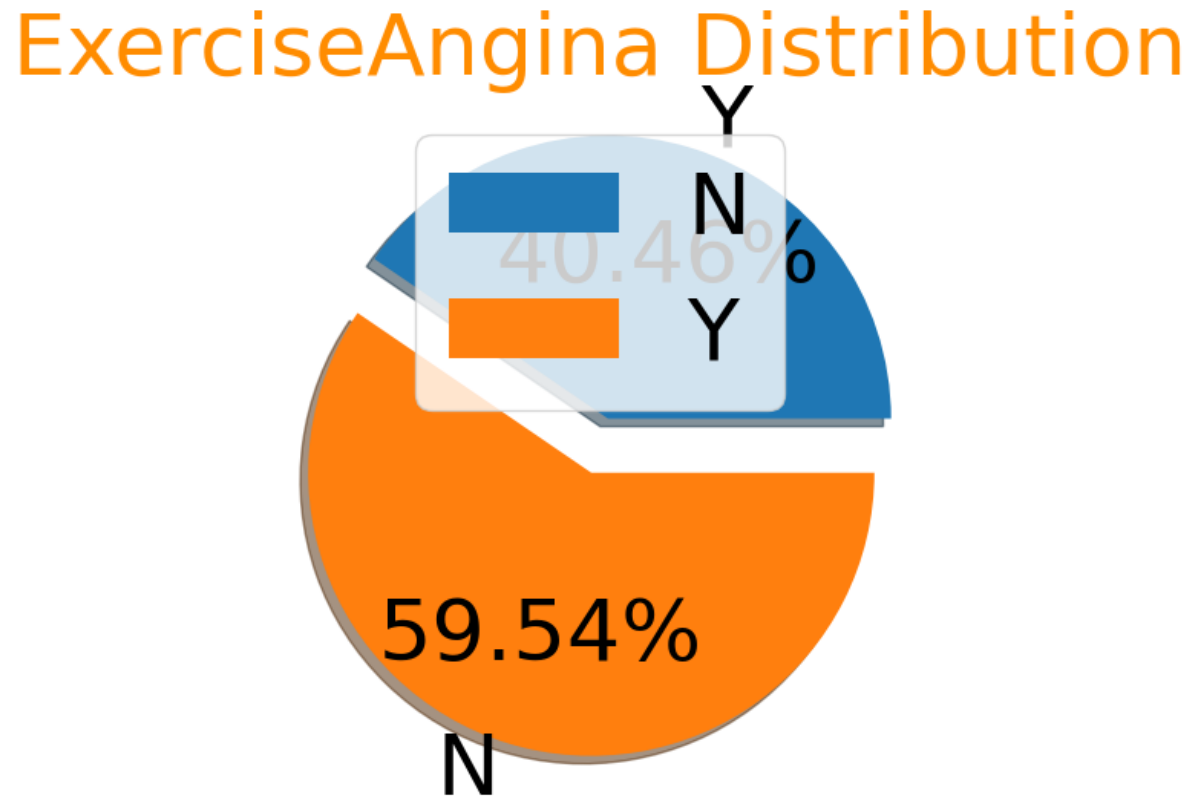


Step 47: Plot the pie chart using matplotlib

```
exercise_angina_counts = heart.groupBy('ExerciseAngina').count()

exercise_angina_counts_pd = exercise_angina_counts.toPandas()

plt.figure(figsize=(10, 6))
plt.pie(exercise_angina_counts_pd['count'], labels=exercise_angina_counts_pd['ExerciseAngina'], explode=[0.1, 0.1], autopct='%1.2f%%', shadow=True
plt.title('ExerciseAngina Distribution', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')

plt.axis('off')
plt.legend(labels=['N', 'Y'])
plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result:



Step 48: Calculate the counts for each unique value in 'ST_Slope'

```python
st_slope_counts = heart.groupBy('ST_Slope').count()
total_rows = heart.count()
st_slope_counts = st_slope_counts.withColumn('Percentage', F.round(st_slope_counts['count'] / total_rows * 100, 2))

st_slope_counts_pd = st_slope_counts.toPandas()

st_slope_counts_pd.rename(columns={'ST_Slope': 'Value'}, inplace=True)
st_slope_counts_pd['Percentage'] = st_slope_counts_pd['Percentage'].apply(lambda x: "{:.2f}%".format(x))
st_slope_counts_pd['Percentage'] = st_slope_counts_pd['Percentage'].str.replace('%', '').astype(float)  # Remove '%' and convert to float

display(st_slope_counts_pd.style.background_gradient(cmap='coolwarm', subset=['count', 'Percentage'], low=0, high=100))
```

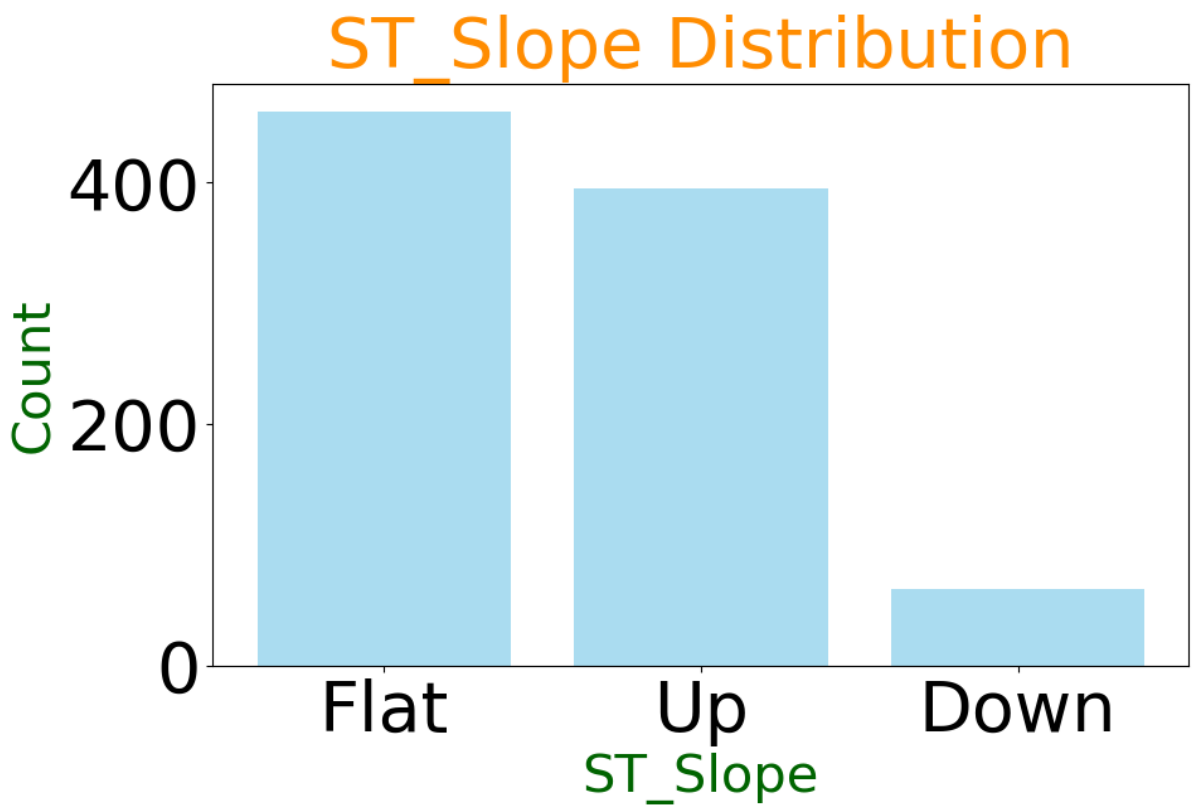
	Value	count	Percentage
0	Flat	459	50.050000
1	Up	395	43.080000
2	Down	63	6.870000

Step 49: Plot the count plot using matplotlib

```python
st_slope_counts = heart.groupBy('ST_Slope').count()

st_slope_counts_pd = st_slope_counts.toPandas()

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(st_slope_counts_pd['ST_Slope'], st_slope_counts_pd['count'], color='skyblue', alpha=0.7)
plt.title('ST_Slope Distribution', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.xlabel('ST_Slope', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')
plt.ylabel('Count', fontsize=30, color='DarkGreen', font='Lucida Calligraphy')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result

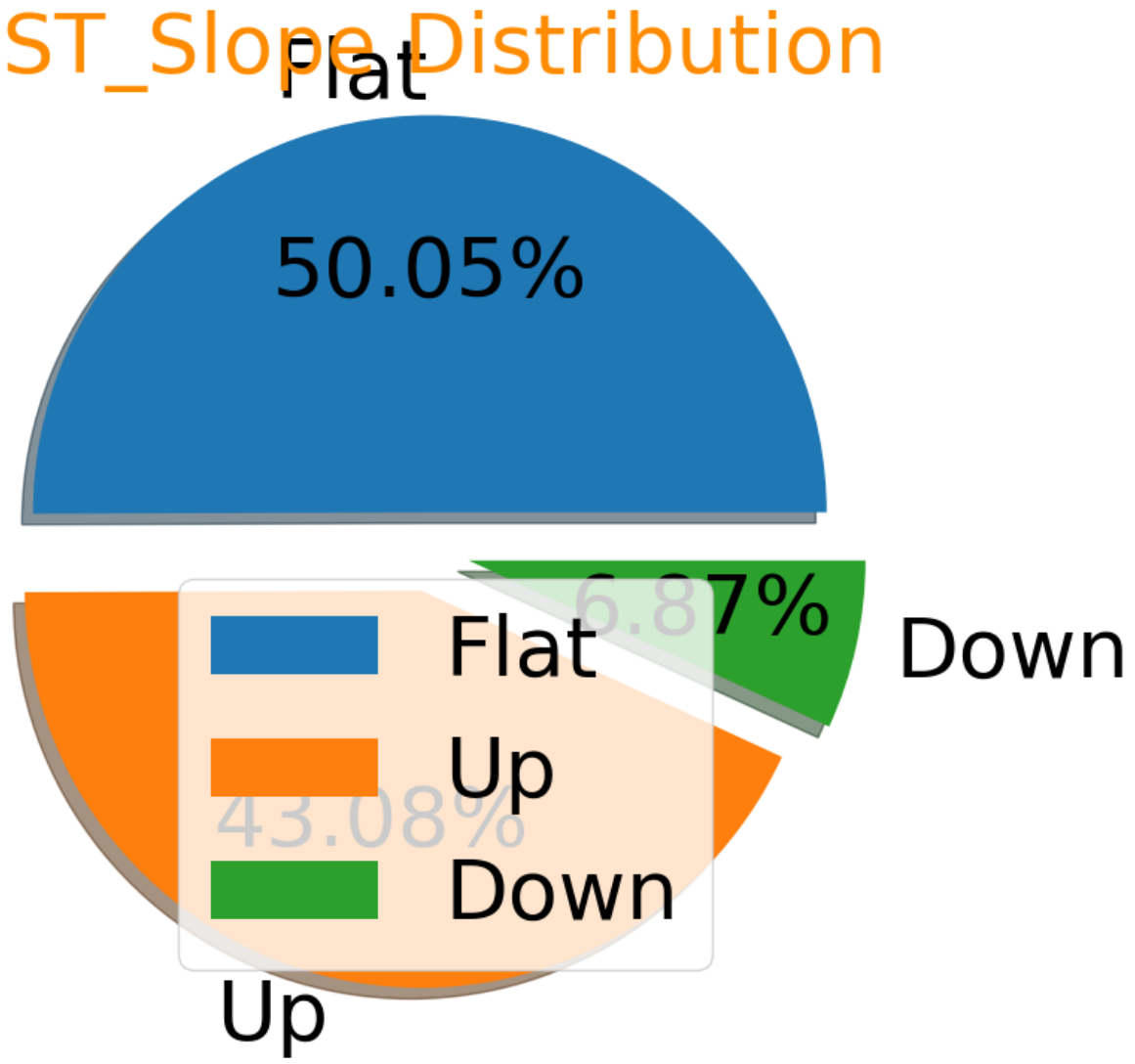


Step 50: Plot the pie chart

```
st_slope_counts = heart.groupBy('ST_Slope').count()

st_slope_counts_pd = st_slope_counts.toPandas()

plt.figure(figsize=(8, 8))
plt.pie(st_slope_counts_pd['count'], labels=st_slope_counts_pd['ST_Slope'], explode=[0.1, 0.1, 0.1], autopct='%1.2f%%', shadow=True)
plt.title('ST_Slope Distribution', fontsize=40, color='DarkOrange', font='Lucida Calligraphy')
plt.legend(labels=['Flat', 'Up', 'Down'])
plt.axis('equal')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Lucida Calligraphy' not found.
```

Here is my result



Step 51: Display the tables side by side using the multi_table function

```
def multi_table(table_list):
    for table in table_list:
        display(table)

variables = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']

df_groupby = {var: heart.groupBy(var, 'HeartDisease').count().alias('Count')
              for var in variables}

df_groupby_pd = {var: df_groupby[var].toPandas() for var in variables}


multi_table([df_groupby_pd['ChestPainType'], df_groupby_pd['Sex'], df_groupby_pd['RestingECG'],
             df_groupby_pd['ExerciseAngina'], df_groupby_pd['ST_Slope']])
```

Here is my result

	ChestPainType	HeartDisease	count
0	TA	1	20
1	ASY	1	392
2	NAP	1	71
3	TA	0	26
4	ASY	0	104
5	ATA	0	149
6	NAP	0	131
7	ATA	1	24

	Sex	HeartDisease	count
0	M	1	457
1	F	0	143
2	F	1	50
3	M	0	267

	RestingECG	HeartDisease	count
0	ST	0	61
1	ST	1	117
2	LVH	1	106
3	Normal	0	267
4	Normal	1	284
5	LVH	0	82

	ExerciseAngina	HeartDisease	count
0	N	1	191
1	Y	1	316
2	Y	0	55
3	N	0	355

	ST_Slope	HeartDisease	count
0	Flat	1	380
1	Down	1	49
2	Up	1	78
3	Down	0	14
4	Up	0	317
5	Flat	0	79

Step 52: show heart table

```
heart.show()
```

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
| 40|  M|          ATA|      140|      289.0|        0|    Normal|  172|             N|    0.0|      Up|           0|
| 49|  F|          NAP|      160|      180.0|        0|    Normal|  156|             N|    1.0|    Flat|           1|
| 37|  M|          ATA|      130|      283.0|        0|        ST|   98|             N|    0.0|      Up|           0|
| 48|  F|          ASY|      138|      214.0|        0|    Normal|  108|             Y|    1.5|    Flat|           1|
| 54|  M|          NAP|      150|      195.0|        0|    Normal|  122|             N|    0.0|      Up|           0|
| 39|  M|          NAP|      120|      339.0|        0|    Normal|  170|             N|    0.0|      Up|           0|
| 45|  F|          ATA|      130|      237.0|        0|    Normal|  170|             N|    0.0|      Up|           0|
| 54|  M|          ATA|      110|      208.0|        0|    Normal|  142|             N|    0.0|      Up|           0|
| 37|  M|          ASY|      140|      207.0|        0|    Normal|  130|             Y|    1.5|    Flat|           1|
| 48|  F|          ATA|      120|      284.0|        0|    Normal|  120|             N|    0.0|      Up|           0|
| 37|  F|          NAP|      130|      211.0|        0|    Normal|  142|             N|    0.0|      Up|           0|
| 58|  M|          ATA|      136|      164.0|        0|        ST|   99|             Y|    2.0|    Flat|           1|
| 39|  M|          ATA|      120|      204.0|        0|    Normal|  145|             N|    0.0|      Up|           0|
| 49|  M|          ASY|      140|      234.0|        0|    Normal|  140|             Y|    1.0|    Flat|           1|
| 42|  F|          NAP|      115|      211.0|        0|        ST|  137|             N|    0.0|      Up|           0|
| 54|  F|          ATA|      120|      273.0|        0|    Normal|  150|             N|    1.5|    Flat|           0|
| 38|  M|          ASY|      110|      196.0|        0|    Normal|  166|             N|    0.0|    Flat|           1|
| 43|  F|          ATA|      120|      201.0|        0|    Normal|  165|             N|    0.0|      Up|           0|
| 60|  M|          ASY|      100|      248.0|        0|    Normal|  125|             N|    1.0|    Flat|           1|
| 36|  M|          ATA|      120|      267.0|        0|    Normal|  160|             N|    3.0|    Flat|           1|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+
only showing top 20 rows
```

Step 53: Show the first few rows of the encoded PySpark DataFrame

```
indexers = [StringIndexer(inputCol=col, outputCol=col + "_index", handleInvalid="keep") for col in ['Sex', 'ChestPainType', 'RestingECG', 'Exerci
indexers_model = [indexer.fit(heart) for indexer in indexers]
heart_indexed = heart
for model in indexers_model:
    heart_indexed = model.transform(heart_indexed)

encoder = OneHotEncoder(inputCols=[col + "_index" for col in ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']],
                        outputCols=[col + "_encoded" for col in ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']])
heart_encoded = encoder.fit(heart_indexed).transform(heart_indexed)

heart_encoded.show()
```

Here is my result:

```
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+---------+-------------------+-
|Age|Sex|ChestPainType|RestingBP|Cholesterol|FastingBS|RestingECG|MaxHR|ExerciseAngina|Oldpeak|ST_Slope|HeartDisease|Sex_index|ChestPainType_index|R
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+---------+-------------------+-
| 40|  M|          ATA|      140|      289.0|        0|    Normal|  172|             N|    0.0|      Up|           0|      0.0|                2.0|
| 49|  F|          NAP|      160|      180.0|        0|    Normal|  156|             N|    1.0|    Flat|           1|      1.0|                1.0|
| 37|  M|          ATA|      130|      283.0|        0|        ST|   98|             N|    0.0|      Up|           0|      0.0|                2.0|
| 48|  F|          ASY|      138|      214.0|        0|    Normal|  108|             Y|    1.5|    Flat|           1|      1.0|                0.0|
| 54|  M|          NAP|      150|      195.0|        0|    Normal|  122|             N|    0.0|      Up|           0|      0.0|                1.0|
| 39|  M|          NAP|      120|      339.0|        0|    Normal|  170|             N|    0.0|      Up|           0|      0.0|                1.0|
| 45|  F|          ATA|      130|      237.0|        0|    Normal|  170|             N|    0.0|      Up|           0|      1.0|                2.0|
| 54|  M|          ATA|      110|      208.0|        0|    Normal|  142|             N|    0.0|      Up|           0|      0.0|                2.0|
| 37|  M|          ASY|      140|      207.0|        0|    Normal|  130|             Y|    1.5|    Flat|           1|      0.0|                0.0|
| 48|  F|          ATA|      120|      284.0|        0|    Normal|  120|             N|    0.0|      Up|           0|      1.0|                2.0|
| 37|  F|          NAP|      130|      211.0|        0|    Normal|  142|             N|    0.0|      Up|           0|      1.0|                1.0|
| 58|  M|          ATA|      136|      164.0|        0|        ST|   99|             Y|    2.0|    Flat|           1|      0.0|                2.0|
| 39|  M|          ATA|      120|      204.0|        0|    Normal|  145|             N|    0.0|      Up|           0|      0.0|                2.0|
| 49|  M|          ASY|      140|      234.0|        0|    Normal|  140|             Y|    1.0|    Flat|           1|      0.0|                0.0|
| 42|  F|          NAP|      115|      211.0|        0|        ST|  137|             N|    0.0|      Up|           0|      1.0|                1.0|
| 54|  F|          ATA|      120|      273.0|        0|    Normal|  150|             N|    1.5|    Flat|           0|      1.0|                2.0|
| 38|  M|          ASY|      110|      196.0|        0|    Normal|  166|             N|    0.0|    Flat|           1|      0.0|                0.0|
| 43|  F|          ATA|      120|      201.0|        0|    Normal|  165|             N|    0.0|      Up|           0|      1.0|                2.0|
| 60|  M|          ASY|      100|      248.0|        0|    Normal|  125|             N|    1.0|    Flat|           1|      0.0|                0.0|
| 36|  M|          ATA|      120|      267.0|        0|    Normal|  160|             N|    3.0|    Flat|           1|      0.0|                2.0|
+---+---+-------------+---------+-----------+---------+----------+-----+--------------+-------+--------+------------+---------+-------------------+-
only showing top 20 rows
```

Step 54: Check the structure of the DataFrame

```
heart.printSchema()
```

```
root
 |-- Age: integer (nullable = true)
 |-- Sex: string (nullable = true)
 |-- ChestPainType: string (nullable = true)
 |-- RestingBP: integer (nullable = true)
 |-- Cholesterol: double (nullable = true)
 |-- FastingBS: integer (nullable = true)
 |-- RestingECG: string (nullable = true)
 |-- MaxHR: integer (nullable = true)
 |-- ExerciseAngina: string (nullable = true)
 |-- Oldpeak: double (nullable = true)
 |-- ST_Slope: string (nullable = true)
 |-- HeartDisease: integer (nullable = true)
```

Step 55: Split the data into training and testing sets

```
categorical_cols = ["Sex", "ChestPainType", "RestingECG", "ExerciseAngina", "ST_Slope"]

existing_indexed_cols = [col for col in heart.columns if col.endswith("_indexed")]

indexers = [StringIndexer(inputCol=col, outputCol=col + "_indexed") for col in categorical_cols if col + "_indexed" not in existing_indexed_cols]
for indexer in indexers:
    heart = indexer.fit(heart).transform(heart)

feature_cols = ["Age", "Sex_indexed", "ChestPainType_indexed", "RestingBP", "Cholesterol", "FastingBS", "RestingECG_indexed", "MaxHR", "ExerciseAng
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
heart = assembler.transform(heart)

train_data, test_data = heart.randomSplit([0.8, 0.2], seed=42)
```

Step 56: Create and fit the logistic regression model

```
lr = LogisticRegression(featuresCol="features", labelCol="HeartDisease")
lr_model = lr.fit(train_data)

predictions = lr_model.transform(test_data)

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="accuracy")
log_accuracy = evaluator.evaluate(predictions)

evaluator_f1 = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="f1")
log_f1 = evaluator_f1.evaluate(predictions)

print("Model Accuracy Score :", log_accuracy * 100, "%")
print("F1 Score             :", log_f1 * 100, "%")
print("Confusion Matrix: \n", predictions.groupBy("HeartDisease", "prediction").count().show())
print("Classification_Report: \n")
predictions.select("HeartDisease", "prediction").show()
```

Here is my result:

```
Model Accuracy Score : 84.45945945945947 %
F1 Score             : 84.22210452381043 %
+------------+----------+-----+
|HeartDisease|prediction|count|
+------------+----------+-----+
|           1|       0.0|    5|
|           0|       0.0|   50|
|           1|       1.0|   75|
|           0|       1.0|   18|
+------------+----------+-----+

Confusion Matrix:
 None
Classification_Report:

+------------+----------+
|HeartDisease|prediction|
+------------+----------+
|           0|       0.0|
|           1|       1.0|
|           1|       1.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
...
|           0|       1.0|
+------------+----------+
only showing top 20 rows
```

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

Step 57: Get the average AUC from the cross-validated model

```
logreg = LogisticRegression(featuresCol="features", labelCol="HeartDisease")

param_grid = ParamGridBuilder() \
    .addGrid(logreg.regParam, [0.01, 0.1, 1.0]) \
    .addGrid(logreg.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()

evaluator = BinaryClassificationEvaluator(labelCol="HeartDisease")

crossval = CrossValidator(estimator=logreg, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=10)

cv_model = crossval.fit(train_data)

mean_auc = cv_model.avgMetrics[0]

print("Model Average AUC: {:.2f} %".format(mean_auc * 100))
```

```
Model Average AUC: 91.35 %
```

Step 58: Create the SVM model

```
svc = LinearSVC(labelCol="HeartDisease", featuresCol="features")

param_grid = ParamGridBuilder() \
    .addGrid(svc.maxIter, [10, 50, 100]) \
    .addGrid(svc.regParam, [0.01, 0.1, 1.0]) \
    .build()

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", metricName="accuracy")

crossval = CrossValidator(estimator=svc, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=10)

cv_model = crossval.fit(train_data)

best_model = cv_model.bestModel

predictions = best_model.transform(test_data)

accuracy = evaluator.evaluate(predictions)

print("Model Accuracy Score  : {:.2f} %".format(accuracy * 100))
print("Confusion Matrix: \n")
predictions.groupBy("HeartDisease", "prediction").count().show()
print("Classification_Report: \n")
predictions.select("HeartDisease", "prediction").show()
```

Here is my result:

```
Model Accuracy Score  : 84.46 %
Confusion Matrix:

+------------+----------+-----+
|HeartDisease|prediction|count|
+------------+----------+-----+
|           1|       0.0|    5|
|           0|       0.0|   50|
|           1|       1.0|   75|
|           0|       1.0|   18|
+------------+----------+-----+

Classification_Report:

+------------+----------+
|HeartDisease|prediction|
+------------+----------+
|           0|       0.0|
|           1|       1.0|
|           1|       1.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       1.0|
...
|           0|       1.0|
+------------+----------+
only showing top 20 rows
```

Step 59: Get the average accuracy from the cross-validated model

```
svc = LinearSVC(labelCol="HeartDisease", featuresCol="features")

param_grid = ParamGridBuilder() \
    .addGrid(svc.maxIter, [10, 50, 100]) \
    .addGrid(svc.regParam, [0.01, 0.1, 1.0]) \
    .build()

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", metricName="accuracy")

crossval = CrossValidator(estimator=svc, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=10)

cv_model = crossval.fit(train_data)

mean_accuracy = cv_model.avgMetrics[0]

print("Model Average Accuracy: {:.2f} %".format(mean_accuracy * 100))
```

```
Model Average Accuracy: 85.19 %
```

Step 60: Create the Random Forest Classifier

```
random_forest = RandomForestClassifier(featuresCol="features", labelCol="HeartDisease", numTrees=100)

model = random_forest.fit(train_data)

predictions = model.transform(test_data)

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions) * 100

print("Model Accuracy Score  : {:.2f} %".format(accuracy))
print("Confusion Matrix: \n", predictions.groupBy("HeartDisease", "prediction").count().show())
print("Classification_Report: \n")
predictions.select("HeartDisease", "prediction").show()
```

Here is my result:

```
Model Accuracy Score  : 83.78 %
+------------+----------+-----+
|HeartDisease|prediction|count|
+------------+----------+-----+
|           1|       0.0|    4|
|           0|       0.0|   48|
|           1|       1.0|   76|
|           0|       1.0|   20|
+------------+----------+-----+

Confusion Matrix:
 None
Classification_Report:

+------------+----------+
|HeartDisease|prediction|
+------------+----------+
|           0|       0.0|
|           1|       1.0|
|           1|       1.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       1.0|
...
|           0|       1.0|
+------------+----------+
only showing top 20 rows
```

Step 61: Evaluate the model's performance

```
    random_forest = RandomForestClassifier(featuresCol="features", labelCol="HeartDisease", numTrees=100)

    paramGrid = ParamGridBuilder()\
        .addGrid(random_forest.maxDepth, [5, 10, 15])\
        .addGrid(random_forest.minInstancesPerNode, [1, 5, 10])\
        .build()

    evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="accuracy")
    crossval = CrossValidator(estimator=random_forest,
                              estimatorParamMaps=paramGrid,
                              evaluator=evaluator,
                              numFolds=10)

    cv_model = crossval.fit(train_data)

    best_model = cv_model.bestModel
    predictions = best_model.transform(test_data)

    accuracy = evaluator.evaluate(predictions) * 100

    print("Model Accuracy Score  : {:.2f} %".format(accuracy))
```

```
Model Accuracy Score  : 85.14 %
```

Step 62: Create the Multilayer Perceptron Classifier

```
mlp = MultilayerPerceptronClassifier(featuresCol="features", labelCol="HeartDisease", layers=[len(feature_cols), 10, 2])

paramGrid = ParamGridBuilder()\
    .addGrid(mlp.maxIter, [50, 100, 150])\
    .addGrid(mlp.blockSize, [32, 64, 128])\
    .build()

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="accuracy")
crossval = CrossValidator(estimator=mlp,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=10)

cv_model = crossval.fit(train_data)

best_model = cv_model.bestModel
predictions = best_model.transform(test_data)

accuracy = evaluator.evaluate(predictions) * 100

print("Model Accuracy Score  : {:.2f} %".format(accuracy))
print("Confusion Matrix: \n", predictions.groupBy("HeartDisease", "prediction").count().show())
print("Classification_Report: \n")
predictions.select("HeartDisease", "prediction").show()
```

Here is my result

```
Model Accuracy Score   : 81.08 %
+------------+----------+-----+
|HeartDisease|prediction|count|
+------------+----------+-----+
|           1|       0.0|    7|
|           0|       0.0|   47|
|           1|       1.0|   73|
|           0|       1.0|   21|
+------------+----------+-----+

Confusion Matrix:
 None
Classification_Report:

+------------+----------+
|HeartDisease|prediction|
+------------+----------+
|           0|       0.0|
|           1|       1.0|
|           1|       1.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       1.0|
...
|           0|       1.0|
+------------+----------+
only showing top 20 rows
```

Step 63: Evaluate the model's performance

```
mlp = MultilayerPerceptronClassifier(featuresCol="features", labelCol="HeartDisease", layers=[len(feature_cols), 10, 2])

paramGrid = ParamGridBuilder()\
    .addGrid(mlp.maxIter, [50, 100, 150])\
    .addGrid(mlp.blockSize, [32, 64, 128])\
    .build()

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="accuracy")
crossval = CrossValidator(estimator=mlp,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=10)

cv_model = crossval.fit(train_data)

val_score = cv_model.avgMetrics

print("Model Accuracy Score: {:.2f} %".format(sum(val_score) * 100 / len(val_score)))
print("Std. Dev: {:.2f} %".format((sum((x - (sum(val_score) / len(val_score))) ** 2 for x in val_score) / len(val_score)) ** 0.5 * 100))
```

```
Model Accuracy Score: 79.48 %
Std. Dev: 5.27 %
```

Step 64: Create the RandomForest

```
decision_tree = DecisionTreeClassifier(featuresCol="features", labelCol="HeartDisease")

model = decision_tree.fit(train_data)

predictions = model.transform(test_data)

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions) * 100

print("Model Accuracy Score  : {:.2f} %".format(accuracy))
print("Confusion Matrix: \n", predictions.groupBy("HeartDisease", "prediction").count().show())
print("Classification_Report: \n")
predictions.select("HeartDisease", "prediction").show()
```

Here is my result:

```
Model Accuracy Score  : 81.08 %
+------------+----------+-----+
|HeartDisease|prediction|count|
+------------+----------+-----+
|           1|       0.0|   10|
|           0|       0.0|   50|
|           1|       1.0|   70|
|           0|       1.0|   18|
+------------+----------+-----+

Confusion Matrix:
 None
Classification_Report:

+------------+----------+
|HeartDisease|prediction|
+------------+----------+
|           0|       0.0|
|           1|       1.0|
|           1|       1.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
|           0|       0.0|
...
|           0|       1.0|
+------------+----------+
only showing top 20 rows
```

Step 65: Get the cross-validated results

```
decision_tree = DecisionTreeClassifier(featuresCol="features", labelCol="HeartDisease")

paramGrid = ParamGridBuilder()\
    .addGrid(decision_tree.maxDepth, [5, 10, 15])\
    .addGrid(decision_tree.maxBins, [32, 64, 128])\
    .build()

evaluator = MulticlassClassificationEvaluator(labelCol="HeartDisease", predictionCol="prediction", metricName="accuracy")
crossval = CrossValidator(estimator=decision_tree,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=10)

cv_model = crossval.fit(train_data)


val_score = cv_model.avgMetrics

print("Model Accuracy Score: {:.2f} %".format(sum(val_score) * 100 / len(val_score)))
print("Std. Dev: {:.2f} %".format((sum((x - (sum(val_score) / len(val_score))) ** 2 for x in val_score) / len(val_score)) ** 0.5 * 100))
```

```
Model Accuracy Score: 81.16 %
Std. Dev: 1.56 %
```

Result of Prediction

Model	Train Accuracy Score	Average Accuracy Score
Logistic Regression Model	84.45	91.35
Support Vector Machines Model	84.46	85.19
Random Forest Classifier Model	83.78	85.14
Multilayer Perceptron Classifier Model	81.08	79.48
Decision Tree Model	81.08	81.16

Conclusion

Look at result of prediction, we can see, Logistic Regression Model has best Score Accuracy. Therefore, it is the most suitable model for this dataset

Reference

[1] Link dataset: https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction