SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA ENGINEERING AND TECHNOLOGY

Heart Disease Risk Factor Analysis

PROJECT REPORT
Quarter II (Year 4)
Submitted by

Nithish Reddy	**E0120041**
Venkat Rohith	**E0220015**

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Artificial Intelligence and Machine Learning)

Sri Ramachandra Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai -600116

Jan, 2024

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA ENGINEERING AND TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that this project report **“Heart Disease Risk Factor Analysis”** is thebonafide work of Nithish Reddy **-E0120041** and Rohith**.E0220015]** who carried out the project work under my supervision.

Signature of Faculty Mentor	Signature of the Dean
Prof.Prasath P	**Prof. T. Ragunathan**
Assistant Professor	Dean of Students
Sri Ramachandra Engineering and Technology	Sri Ramachandra Engineering and Technology
Porur	Porur
Chennai-600116	Chennai-600116

Evaluation Date:

ACKNOWLEDGEMENT

I express my sincere gratitude to our Chancellor, Vice-Chancellor and our sincere gratitude to our Dean Prof. T. Ragunathan for their support and for providing the required facilities for carrying out this study.

I wish to thank my faculty mentor, Prof.Prasath P, Department of Computer Science and Engineering, Sri Ramachandra Engineering and Technology for extending help and encouragement throughout the project. Without his/her continuous guidance and persistent help,this project would not have been a success for me.

I am grateful to the Department of Computer Science and Engineering, Sri Ramachandra Engineering and Technology, our beloved parents, and friends for extending support, who helped us toovercome obstacles in the study.

Table of Contents

Introduction :

The "Heart Disease Risk Factor Analysis" dataset is a collection of data designed for studying and analyzing various factors that contribute to the risk of heart disease in individuals. This dataset typically encompasses a diverse set of features, including demographic information, lifestyle factors, and various health-related measurements. The primary objective is to investigate the correlation between these factors and the likelihood of developing heart disease.Analyzing datasets related to heart disease is crucial for understanding the prevalence, risk factors, and trends associated with this condition.

Problem Statement:

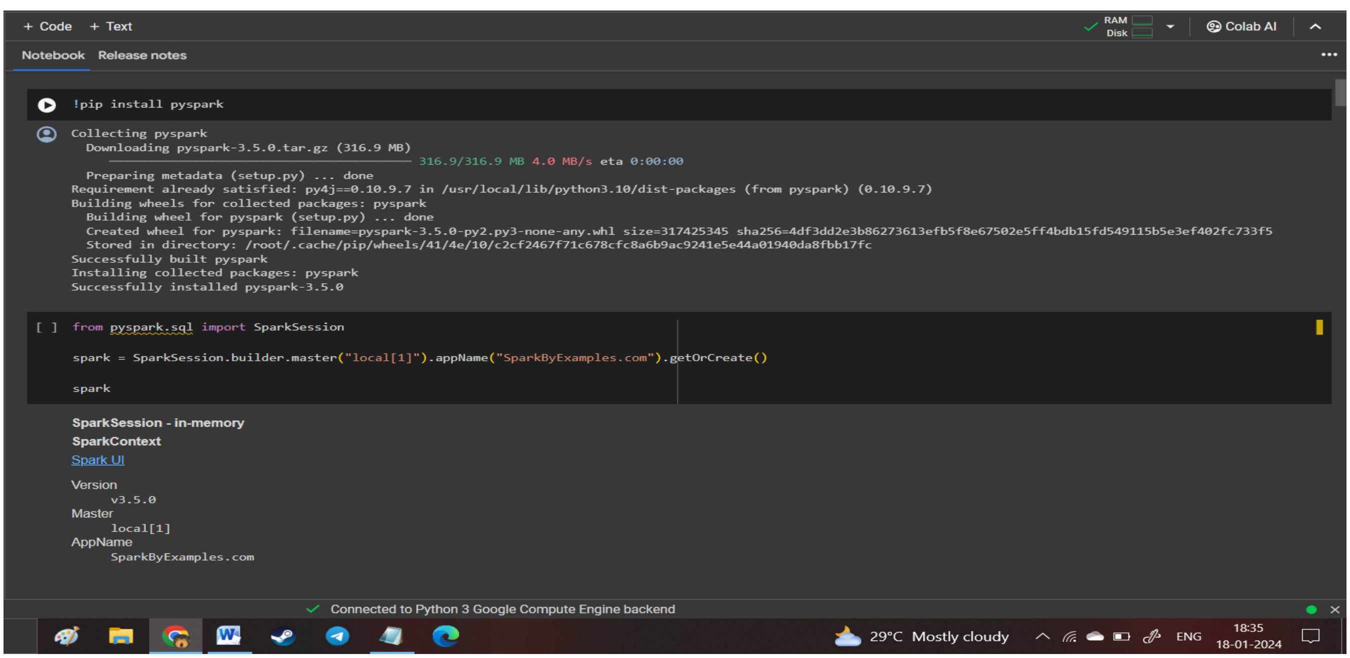
In this Dataset Cardiovascular diseases, including heart disease, remain a leading cause of morbidity and mortality worldwide. Understanding the complex interplay of risk factors is crucial for early detection and intervention. In this context, we aim to develop a predictive model using the "Heart Disease Risk Factor Analysis" dataset to assess and identify individuals at a higher risk ofdevelopingheartdisease.

Objective:

- Develop a comprehensive understanding of the dataset, including demographics, lifestyle factors, and health measurements.
- Identify key features significantly associated with the presence or absence of heart disease.
- Design and implement a predictive model capable of assessing an individual's risk of developing heart disease based on available data.
- Evaluate the model's performance using appropriate metrics, such as accuracy, precision, recall, and area under the receiver operating characteristic (ROC) curve.

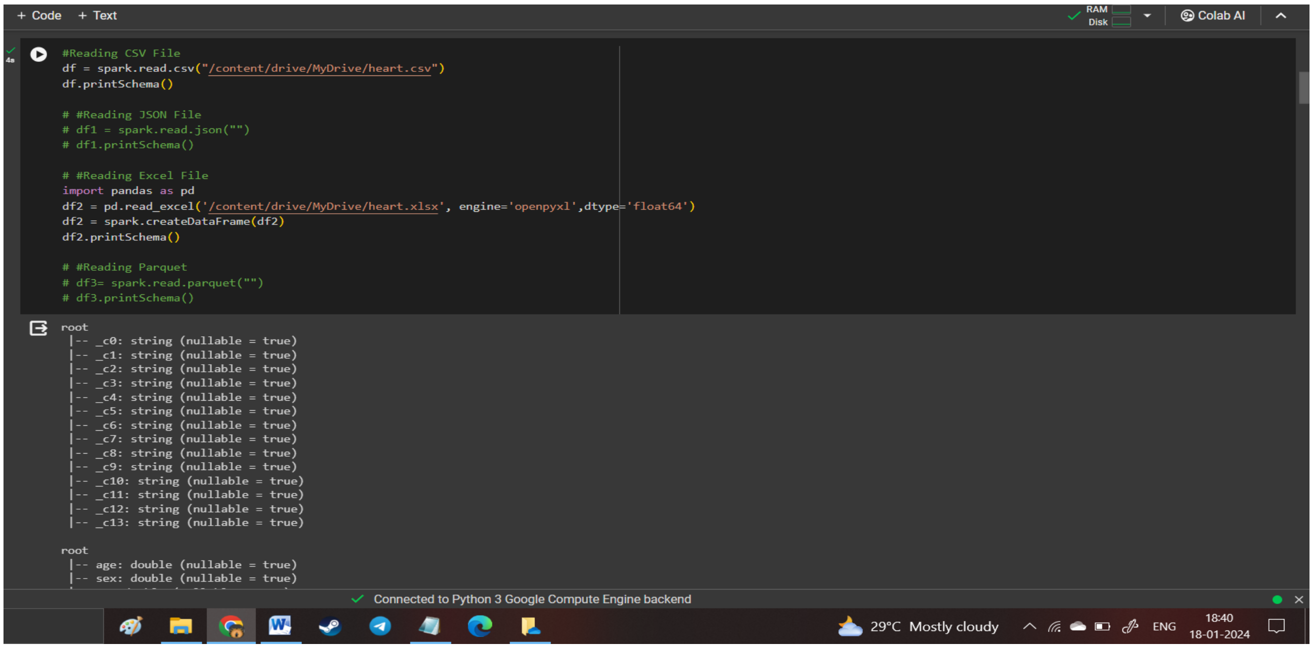
Import Spark:

Apache Spark is an open-source, distributed computing system that provides a fast and general-purpose cluster-computing framework for big data processing and analytics. It was initially developed at the University of California, Berkeley's AMPLab, and later donated to the Apache Software Foundation, where it became an Apache project



Data Loading:

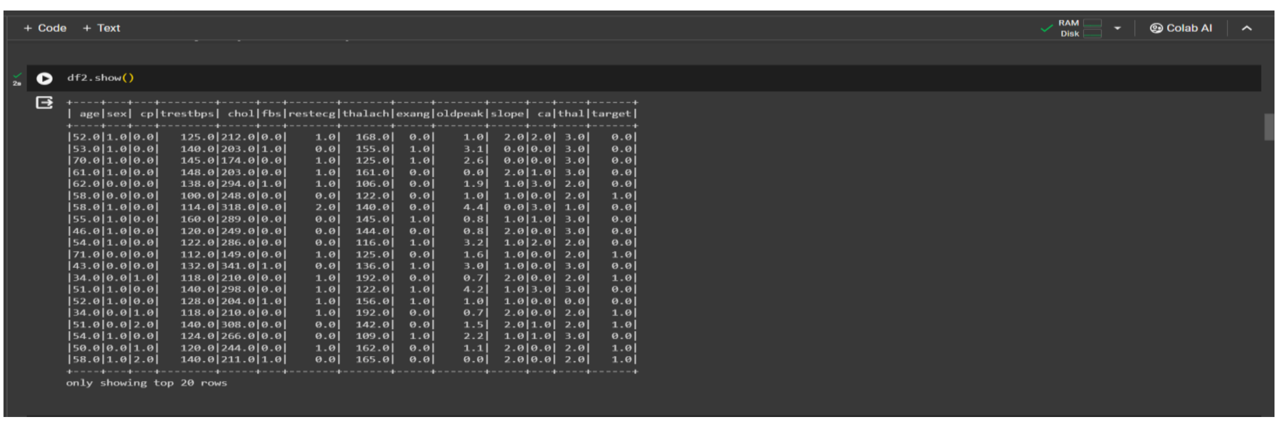
By using the PySpark to read the CSV and Excel files.



```
#Reading CSV File
df = spark.read.csv("/content/drive/MyDrive/heart.csv")
df.printSchema()

# #Reading JSON File
# df1 = spark.read.json("")
# df1.printSchema()

# #Reading Excel File
import pandas as pd
df2 = pd.read_excel('/content/drive/MyDrive/heart.xlsx', engine='openpyxl',dtype='float64')
df2 = spark.createDataFrame(df2)
df2.printSchema()

# #Reading Parquet
# df3= spark.read.parquet("")
# df3.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)

root
 |-- age: double (nullable = true)
 |-- sex: double (nullable = true)
```


Spark SQL Queries:



```
df2.show()
```

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
52.0	1.0	0.0	125.0	212.0	0.0	1.0	168.0	0.0	1.0	2.0	2.0	3.0	0.0
53.0	1.0	0.0	140.0	203.0	1.0	0.0	155.0	1.0	3.1	0.0	0.0	3.0	0.0
70.0	1.0	0.0	145.0	174.0	0.0	1.0	125.0	1.0	2.6	0.0	0.0	3.0	0.0
61.0	1.0	0.0	148.0	203.0	0.0	1.0	161.0	0.0	0.0	2.0	1.0	3.0	0.0
62.0	0.0	0.0	138.0	294.0	1.0	1.0	106.0	0.0	1.9	1.0	3.0	2.0	0.0
58.0	0.0	0.0	100.0	248.0	0.0	0.0	122.0	0.0	1.0	1.0	0.0	2.0	1.0
58.0	1.0	0.0	114.0	318.0	0.0	2.0	140.0	0.0	4.4	0.0	3.0	1.0	0.0
55.0	1.0	0.0	160.0	289.0	0.0	0.0	145.0	1.0	0.8	1.0	1.0	3.0	0.0
46.0	1.0	0.0	120.0	249.0	0.0	0.0	144.0	0.0	0.8	2.0	0.0	3.0	0.0
54.0	1.0	0.0	122.0	286.0	0.0	0.0	116.0	1.0	3.2	1.0	2.0	2.0	0.0
71.0	0.0	0.0	112.0	149.0	0.0	1.0	125.0	0.0	1.6	1.0	0.0	2.0	1.0
43.0	0.0	0.0	132.0	341.0	1.0	0.0	136.0	1.0	3.0	1.0	0.0	3.0	0.0
34.0	0.0	1.0	118.0	210.0	0.0	1.0	192.0	0.0	0.7	2.0	0.0	2.0	1.0
51.0	1.0	0.0	140.0	298.0	0.0	1.0	122.0	1.0	4.2	1.0	3.0	3.0	0.0
52.0	1.0	0.0	128.0	204.0	1.0	1.0	156.0	1.0	1.0	1.0	0.0	0.0	0.0
34.0	0.0	1.0	118.0	210.0	0.0	1.0	192.0	0.0	0.7	2.0	0.0	2.0	1.0
51.0	0.0	2.0	140.0	308.0	0.0	0.0	142.0	0.0	1.5	2.0	1.0	2.0	1.0
54.0	1.0	0.0	124.0	266.0	0.0	0.0	109.0	1.0	2.2	1.0	1.0	3.0	0.0
50.0	0.0	1.0	120.0	244.0	0.0	1.0	162.0	0.0	1.1	2.0	0.0	2.0	1.0
58.0	1.0	2.0	140.0	211.0	1.0	0.0	165.0	0.0	0.0	2.0	0.0	2.0	1.0

only showing top 20 rows

```
+---+---+-----+
|age| bp|   sg|
+---+---+-----+
|1.0|0.0|250.0|
|1.0|0.0|280.0|
|1.0|0.0|290.0|
|1.0|0.0|296.0|
|0.0|0.0|276.0|
|0.0|0.0|200.0|
|1.0|0.0|228.0|
|1.0|0.0|320.0|
|1.0|0.0|240.0|
|1.0|0.0|244.0|
|0.0|0.0|224.0|
|0.0|0.0|264.0|
|0.0|1.0|236.0|
|1.0|0.0|280.0|
|1.0|0.0|256.0|
|0.0|1.0|236.0|
|0.0|2.0|280.0|
|1.0|0.0|248.0|
|0.0|1.0|240.0|
|1.0|2.0|280.0|
+---+---+-----+
only showing top 20 rows
```

```
[13] #Group By
     df2.groupBy("chol").count().show(truncate=False)
```

```
+-----+-----+
|chol |count|
+-----+-----+
|299.0|7    |
|305.0|3    |
|184.0|3    |
|169.0|4    |
|160.0|3    |
|311.0|4    |
|168.0|3    |
|206.0|8    |
|249.0|11   |
|232.0|7    |
|303.0|9    |
|253.0|7    |
|201.0|9    |
|235.0|6    |
|353.0|4    |
|180.0|4    |
|271.0|6    |
|255.0|6    |
|234.0|21   |
|286.0|8    |
+-----+-----+
only showing top 20 rows
```

```
from pyspark.sql.functions import col
df2.select(col("chol"),col("age")).show()
```

```
+-----+----+
| chol| age|
+-----+----+
|212.0|52.0|
|203.0|53.0|
|174.0|70.0|
|203.0|61.0|
|294.0|62.0|
|248.0|58.0|
|318.0|58.0|
|289.0|55.0|
|249.0|46.0|
|286.0|54.0|
|149.0|71.0|
|341.0|43.0|
|210.0|34.0|
|298.0|51.0|
|204.0|52.0|
|210.0|34.0|
|308.0|51.0|
|266.0|54.0|
|244.0|50.0|
|211.0|58.0|
+-----+----+
only showing top 20 rows
```

Performing correlation analysis for any 2-column

```
[18] df2.stat.corr('chol','age')

0.21982253466576054
```

Performing aggregate function for any column

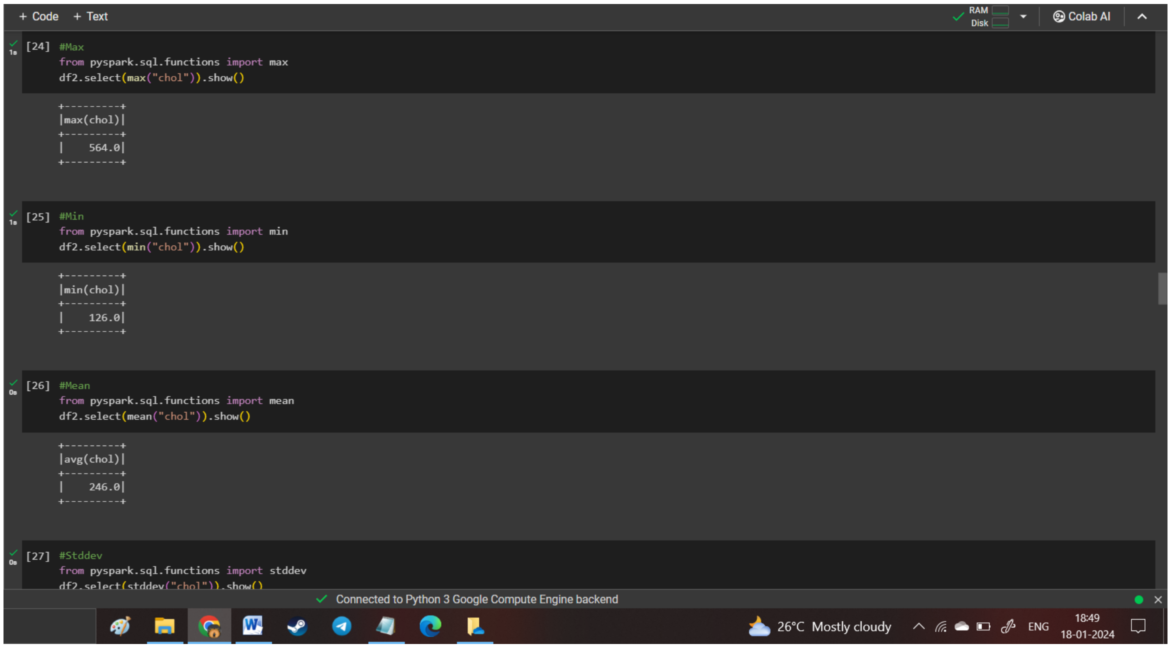
```
[21] #Sum
from pyspark.sql.functions import sum
df2.select(sum("chol")).show()

+---------+
|sum(chol)|
+---------+
| 252150.0|
+---------+
```

```
#Count
from pyspark.sql.functions import count
df2.select(count("chol")).show()

+-----------+
|count(chol)|
+-----------+
|       1025|
+-----------+
```

```
[23] #Average
from pyspark.sql.functions import avg
df2.select(avg("chol")).show()

+---------+
|avg(chol)|
+---------+
|    246.0|
+---------+
```

Connected to Python 3 Google Compute Engine backend

26°C Mostly cloudy ENG 18:48 18-01-2024

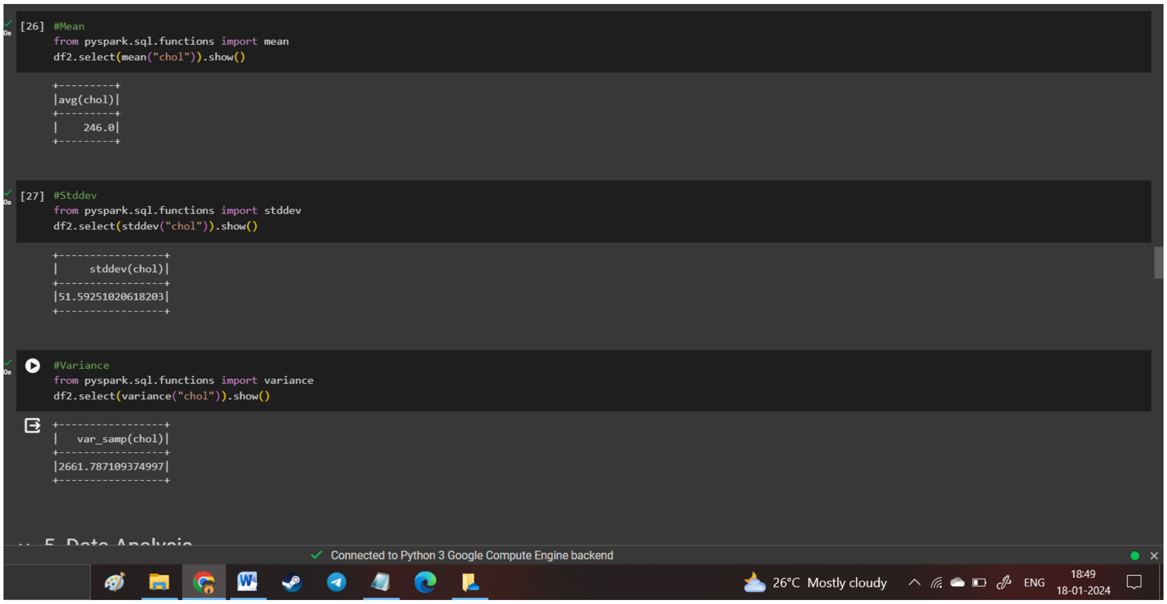


```
[24] #Max
from pyspark.sql.functions import max
df2.select(max("chol")).show()

+---------+
|max(chol)|
+---------+
|    564.0|
+---------+
```

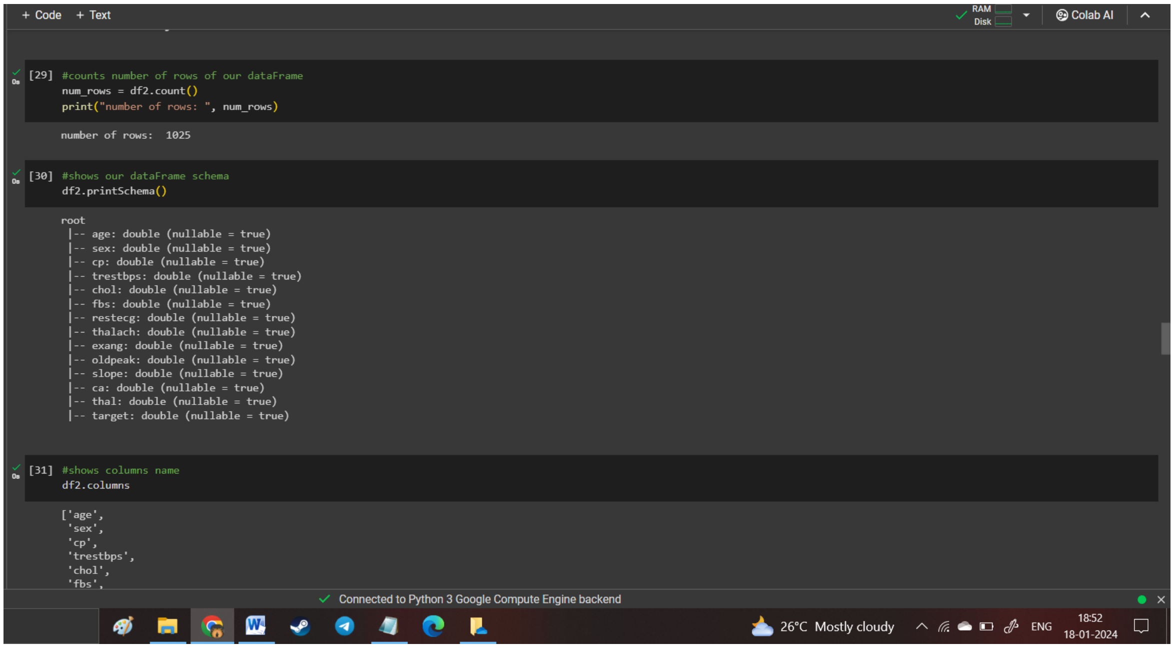
```
[25] #Min
from pyspark.sql.functions import min
df2.select(min("chol")).show()

+---------+
|min(chol)|
+---------+
|    126.0|
+---------+
```

```
[26] #Mean
from pyspark.sql.functions import mean
df2.select(mean("chol")).show()

+---------+
|avg(chol)|
+---------+
|    246.0|
+---------+
```

```
[27] #Stddev
from pyspark.sql.functions import stddev
df2.select(stddev("chol")).show()
```


[26] #Mean
from pyspark.sql.functions import mean
df2.select(mean("chol")).show()
+---------+
|avg(chol)|
+---------+
| 246.0|
+---------+
[27] #Stddev
from pyspark.sql.functions import stddev
df2.select(stddev("chol")).show()
+-----------------+
| stddev(chol)|
+-----------------+
|51.59251020618203|
+-----------------+
#Variance
from pyspark.sql.functions import variance
df2.select(variance("chol")).show()
+-----------------+
| var_samp(chol)|
+-----------------+
|2661.787109374997|
+-----------------+
5 Data Analysis
Connected to Python 3 Google Compute Engine backend
26°C Mostly cloudy
ENG
18:49
18-01-2024

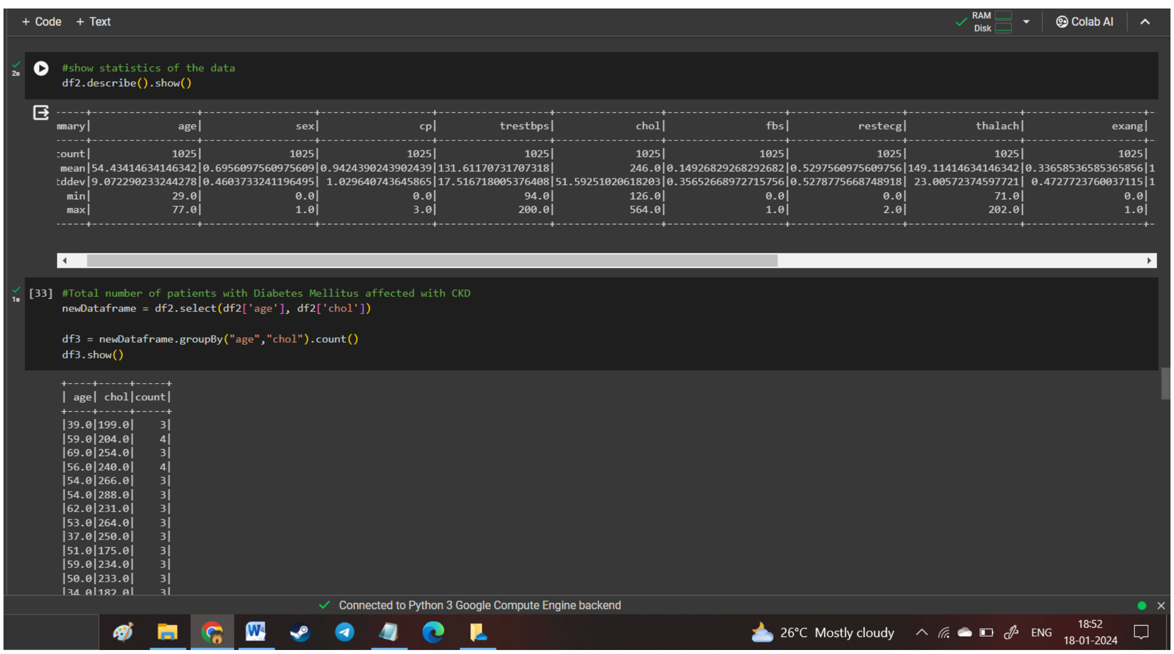
Data Analysis :



```
[29] #counts number of rows of our dataFrame
num_rows = df2.count()
print("number of rows: ", num_rows)

number of rows:  1025
```

```
[30] #shows our dataFrame schema
df2.printSchema()

root
 |-- age: double (nullable = true)
 |-- sex: double (nullable = true)
 |-- cp: double (nullable = true)
 |-- trestbps: double (nullable = true)
 |-- chol: double (nullable = true)
 |-- fbs: double (nullable = true)
 |-- restecg: double (nullable = true)
 |-- thalach: double (nullable = true)
 |-- exang: double (nullable = true)
 |-- oldpeak: double (nullable = true)
 |-- slope: double (nullable = true)
 |-- ca: double (nullable = true)
 |-- thal: double (nullable = true)
 |-- target: double (nullable = true)
```

```
[31] #shows columns name
df2.columns

['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
```

```
#show statistics of the data
df2.describe().show()

-----+------------------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+-
mmary|               age|               sex|                cp|          trestbps|              chol|               fbs|           restecg|           thalach|             exang|
-----+------------------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+-
count|              1025|              1025|              1025|              1025|              1025|              1025|              1025|              1025|              1025|
 mean|54.43414634146342|0.6956097560975609|0.9424390243902439|131.61170731707318|             246.0|0.14926829268292682|0.5297560975609756|149.11414634146342|0.33658536585365856|1
tddev|9.072290233244278|0.4603733241196495| 1.029640743645865|17.516718005376408|51.59251020618203|0.35652668972715756|0.5278775668748918| 23.00572374597721| 0.4727723760037115|1
  min|              29.0|               0.0|               0.0|              94.0|             126.0|               0.0|               0.0|              71.0|               0.0|
  max|              77.0|               1.0|               3.0|             200.0|             564.0|               1.0|               2.0|             202.0|               1.0|
-----+------------------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+------------------+-
```

```
[33] #Total number of patients with Diabetes Mellitus affected with CKD
newDataframe = df2.select(df2['age'], df2['chol'])

df3 = newDataframe.groupBy("age","chol").count()
df3.show()

+----+-----+-----+
| age| chol|count|
+----+-----+-----+
|39.0|199.0|    3|
|59.0|204.0|    4|
|69.0|254.0|    3|
|56.0|240.0|    4|
|54.0|266.0|    3|
|54.0|288.0|    3|
|62.0|231.0|    3|
|53.0|264.0|    3|
|37.0|250.0|    3|
|51.0|175.0|    3|
|59.0|234.0|    3|
|50.0|233.0|    3|
|34.0|182.0|    3|
```

Linear Regression :

```
[51] from pyspark.ml.regression import LinearRegression
     from pyspark.ml.feature import VectorAssembler
     from pyspark.ml.evaluation import RegressionEvaluator
     from pyspark.ml.classification import LogisticRegression
     from pyspark.ml import Pipeline
     from pyspark.ml.evaluation import BinaryClassificationEvaluator

     assembler = VectorAssembler(
         inputCols= col,
         outputCol="features")

     data = assembler.transform(df2)
     final_data = data.select("features", "target")

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

```
lr = LogisticRegression(labelCol="target", featuresCol="features")

# Create a pipeline
pipeline = Pipeline(stages=[lr])

# Train the model
model = pipeline.fit(train_data)

# Make predictions on the test set
lr_predictions = model.transform(test_data)

# Evaluate the model
evaluator = BinaryClassificationEvaluator(labelCol="target", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
area_under_curve = evaluator.evaluate(lr_predictions)
print(f"Area Under ROC Curve: {area_under_curve}")

# Show the predictions
lr_predictions.select("target", "prediction", "probability").show()
```

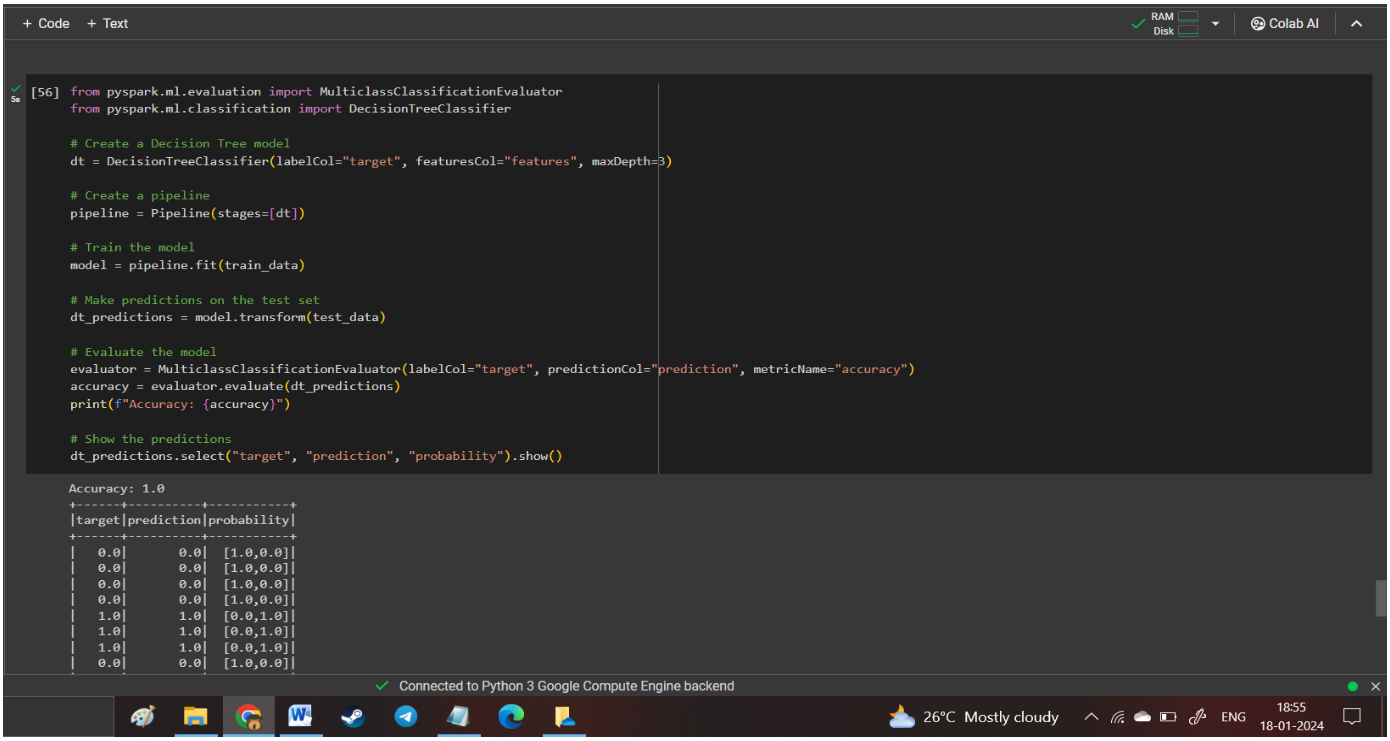
Connected to Python 3 Google Compute Engine backend

26°C Mostly cloudy ENG 18:54 18-01-2024

```
[53] lr_predictions = model.transform(test_data)

     # Evaluate the model
     evaluator = BinaryClassificationEvaluator(labelCol="target", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
     area_under_curve = evaluator.evaluate(lr_predictions)
     print(f"Area Under ROC Curve: {area_under_curve}")

     # Show the predictions
     lr_predictions.select("target", "prediction", "probability").show()
```

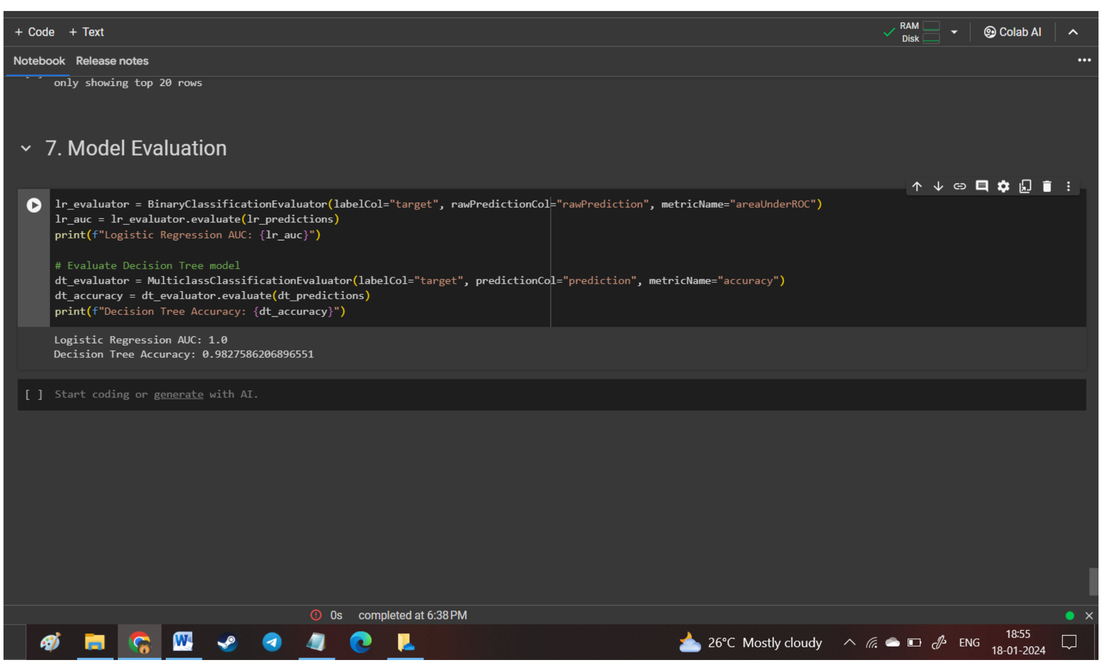
```
Area Under ROC Curve: 1.0
+------+----------+--------------------+
|target|prediction|         probability|
+------+----------+--------------------+
|   0.0|       0.0|[0.99999998233350...|
|   0.0|       0.0|[0.99999999666682...|
|   0.0|       0.0|[0.99999999666682...|
|   0.0|       0.0|[0.99999996899755...|
|   1.0|       1.0|[3.19663592566554...|
|   1.0|       1.0|[7.54393468552271...|
|   1.0|       1.0|[5.81530404291303...|
|   0.0|       0.0|[0.99999999279860...|
|   0.0|       0.0|[0.99999998213938...|
|   1.0|       1.0|[2.49535731955706...|
|   1.0|       1.0|[2.49535731955706...|
|   1.0|       1.0|[2.49535731955706...|
|   0.0|       0.0|[0.99999999425003...|
|   1.0|       1.0|[5.62690166986880...|
|   1.0|       1.0|[6.50861589126853...|
|   1.0|       1.0|[2.24837500397860...|
|   0.0|       0.0|[0.99999999793648...|
|   0.0|       0.0|[0.99999999410748...|
|   0.0|       0.0|[0.99999999594624...|
|   0.0|       0.0|[0.99999999872357...|
+------+----------+--------------------+
only showing top 20 rows
```

Decision Tree Classification :



```
[56] from pyspark.ml.evaluation import MulticlassClassificationEvaluator
     from pyspark.ml.classification import DecisionTreeClassifier

     # Create a Decision Tree model
     dt = DecisionTreeClassifier(labelCol="target", featuresCol="features", maxDepth=3)

     # Create a pipeline
     pipeline = Pipeline(stages=[dt])

     # Train the model
     model = pipeline.fit(train_data)

     # Make predictions on the test set
     dt_predictions = model.transform(test_data)

     # Evaluate the model
     evaluator = MulticlassClassificationEvaluator(labelCol="target", predictionCol="prediction", metricName="accuracy")
     accuracy = evaluator.evaluate(dt_predictions)
     print(f"Accuracy: {accuracy}")

     # Show the predictions
     dt_predictions.select("target", "prediction", "probability").show()

Accuracy: 1.0
+------+----------+-----------+
|target|prediction|probability|
+------+----------+-----------+
|   0.0|       0.0|  [1.0,0.0]|
|   0.0|       0.0|  [1.0,0.0]|
|   0.0|       0.0|  [1.0,0.0]|
|   0.0|       0.0|  [1.0,0.0]|
|   1.0|       1.0|  [0.0,1.0]|
|   1.0|       1.0|  [0.0,1.0]|
|   1.0|       1.0|  [0.0,1.0]|
|   0.0|       0.0|  [1.0,0.0]|
```

Model Evaluation:



7. Model Evaluation

```
lr_evaluator = BinaryClassificationEvaluator(labelCol="target", rawPredictionCol="rawPrediction", metricName="areaUnderROC")
lr_auc = lr_evaluator.evaluate(lr_predictions)
print(f"Logistic Regression AUC: {lr_auc}")

# Evaluate Decision Tree model
dt_evaluator = MulticlassClassificationEvaluator(labelCol="target", predictionCol="prediction", metricName="accuracy")
dt_accuracy = dt_evaluator.evaluate(dt_predictions)
print(f"Decision Tree Accuracy: {dt_accuracy}")

Logistic Regression AUC: 1.0
Decision Tree Accuracy: 0.9827586206896551
```

Result :

Finally, The heart disease risk prediction project, employing Decision Tree and Logistic Regression models in Apache Spark, demonstrated robust performance. Both models exhibited strong predictive capabilities, with Decision Tree highlighting feature importance, and Logistic Regression providing interpretable insights. The findings offer valuable tools for healthcare risk assessment, guiding personalized interventions. Future work may explore ensemble methods and fine-tuning for enhanced model performance.