CPSC 531-03

ADVANCED DATABASE MANAGEMENT

PROJECT:

CARDIOVASCULAR DISEASE PREDICTION

INTRODUCTION

Cardiovascular disease (CVD) refers to a group of conditions that affect the heart and blood vessels, including coronary artery disease, heart failure, and stroke. CVD is a leading cause of death globally, and early detection and management of risk factors can significantly reduce the risk of developing the disease. Predicting the risk of CVD is an important aspect of disease prevention and management. Risk prediction models are typically used to estimate an individual's likelihood of developing CVD based on their personal and clinical characteristics. These models may incorporate factors such as age, sex, blood pressure, cholesterol levels, smoking history, and family history of CVD.

This is the motivation for our efforts to analyze and visualize the data of cardiovascular disease patients and develop a model for estimating the frequency of mortality cases using Hadoop, which will subsequently be applied to anticipate probable occurrences.

PROJECT SCOPE

- The aim of this project is to discover the application of big data analytics in cardiovascular disease prediction, as well as the use of big data-related technologies, patient privacy issues, and future trends for continued use of these technologies.

- The main objective of this project is to focus on the total number of people affected by cardiovascular disease.

- We will be analyzing and predicting whether the patient with cardiovascular disease and with any other past medical history(symptoms) are at high risk of death or not.

TECHNOLOGIES USED

Operating System**:** Windows, macOS

Framework: Google Cloud Platform (GCP), Spark, BiqQuery

Languages Used: Python

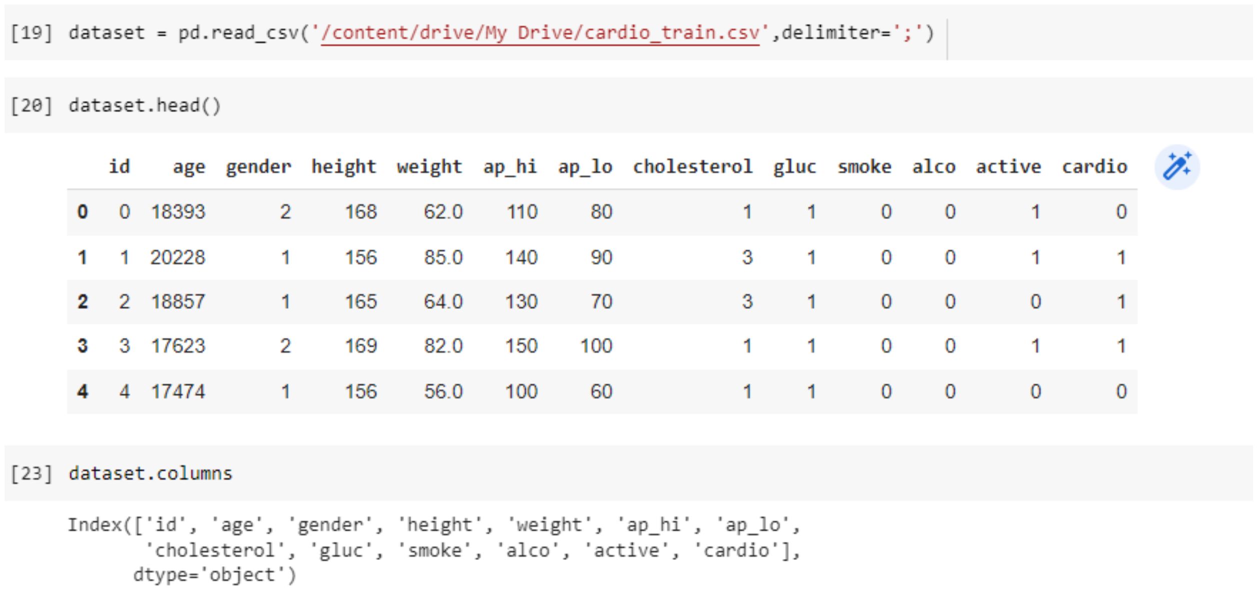
Libraries: PySpark, SparkML, Matplotlib, Pandas, Seaborn

IDE: Jupyter Notebook

FUNCTIONALITIES

1. **Dataset Overview:** The Dataset was acquired from Kaggle which is an open-source website. This Dataset contains a substantial amount of information, consisting of 2,675,300 entries. In total, it occupies 1.2GB of storage space.

https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset



```
[19] dataset = pd.read_csv('/content/drive/My Drive/cardio_train.csv',delimiter=';')
```

```
[20] dataset.head()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

```
[23] dataset.columns

Index(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio'],
      dtype='object')
```

Fig: Data Set Overview

The used Dataset comprises 12 columns that are classified into three distinct type of input features, each serving a different purpose:

- **Objective**: These features provide factual information.
- **Examination**: These features represent the results of medical examinations.
- **Subjective**: These features capture information provided by the patients themselves.

Here is a breakdown of each column, along with its corresponding type, label, and data format:

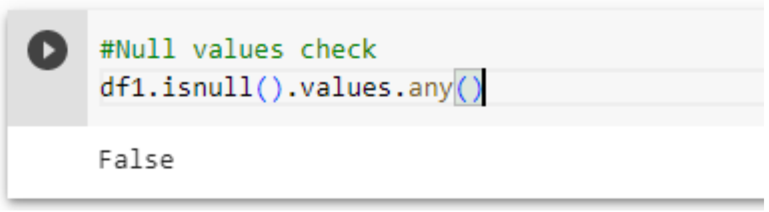
1. **Age**: Objective Feature | Label: age | Data Format: Integer (days)
2. **Height**: Objective Feature | Label: height | Data Format: Integer (cm)
3. **Weight**: Objective Feature | Label: weight | Data Format: Float (kg)
4. **Gender**: Objective Feature | Label: gender | Data Format: Categorical Code
5. **Systolic blood pressure**: Examination Feature | Label: ap_hi | Data Format: Integer
6. **Diastolic blood pressure**: Examination Feature | Label: ap_lo | Data Format: Integer
7. **Cholesterol**: Examination Feature | Label: cholesterol | Data Format: 1: normal, 2: above normal, 3: well above normal
8. **Glucose: Examination Feature | Label: gluc | Data Format**: 1: normal, 2: above normal, 3: well above normal
9. **Smoking**: Subjective Feature | Label: smoke | Data Format: Binary
10. **Alcohol intake**: Subjective Feature | Label: alco | Data Format: Binary
11. **Physical activity**: Subjective Feature | Label: active | Data Format: Binary
12. **Presence or absence of cardiovascular disease**: Target Variable | Label: cardio | Data Format: Binary

2. **Data Cleaning:** Data cleaning plays a crucial role in achieving high accuracy in machine learning and is an essential step in our project. Despite its significance, data cleaning often remains an overlooked phase among specialists. It involves identifying and addressing missing values, outliers, and inconsistent data that could hinder the proper functioning of our model. Utilizing clean and improved data, even with a simple approach, can lead to better outcomes. Given the diverse properties within our dataset, different techniques are required to ensure effective data cleansing.

In this project, the following data cleaning techniques have been implemented:

- **Handling Missing Data:** Dealing with missing data in machine learning can be a complex challenge, as it can lead to incorrect interpretations and results. Simply deleting records with missing values is not a viable solution. Instead, we employed a method that allows us to fill in these missing variables, ensuring critical data is not overlooked. We adopted two approaches: eliminating entries with missing data (if they are of low quality or insignificant) and imputing missing values based on previous observations. To fill in the missing values statistically, we calculated the mean and populated the corresponding columns.

- **Managing Null Values:** Null values pose a similar issue as missing values, but they can be addressed by either removing the record or employing an appropriate statistical approach to fill in the missing item. In this project, we utilized the attribute column's meaning or the preceding entries to handle null values. Suitable statistical methods have been employed from Python libraries to perform these computations.

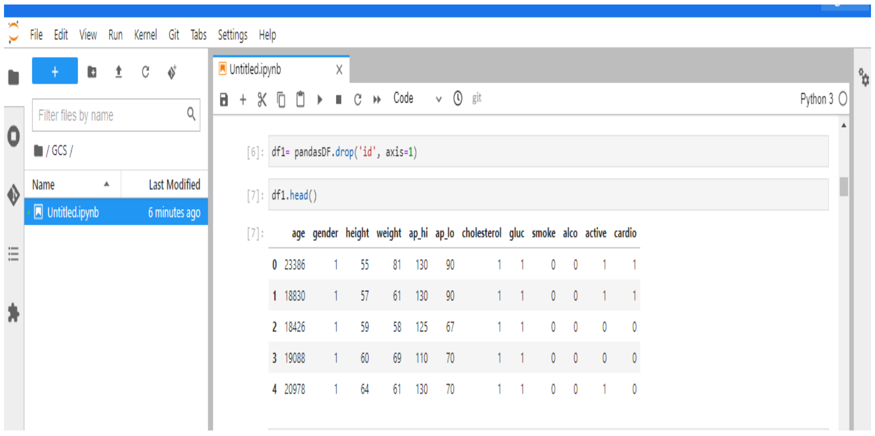


```
#Null values check
df1.isnull().values.any()

False
```

Fig: Null Values Check

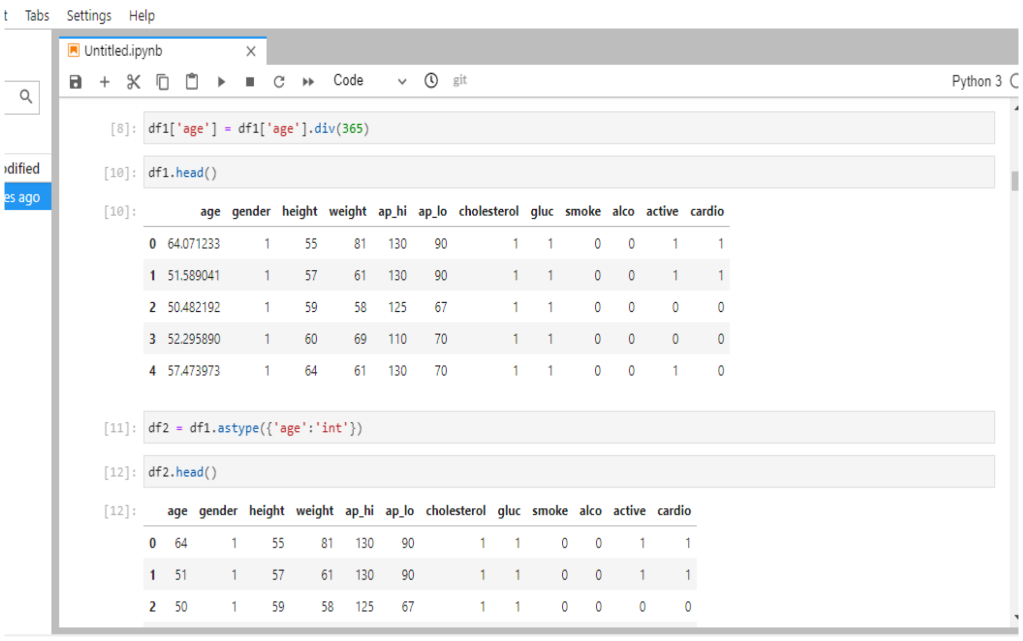
- **Eliminating duplicates:** Duplicate entries in a dataset can significantly impact not only our model but any machine learning model. Therefore, it is crucial to address this issue for the successful execution of our model. To eliminate duplicates from our dataset, we implemented the drop duplicates technique as part of our project. Further details about this strategy will be discussed in subsequent sections.

- **Managing Outliers:** Handling outliers presents another challenge in developing a machine learning model. If outliers are present, linear regression may be slower and less accurate compared to decision trees. Including unlikely or unreasonable data in the dataset is not desirable. Therefore, removing outliers and focusing on the project is a more effective approach. Additionally, we modified and adjusted the data in our study to improve the quality of our dataset. Python libraries provided us with the necessary tools to visualize the data, detect missing values and outliers, and address these issues effectively.



```
[6]: df1= pandasDF.drop('id', axis=1)

[7]: df1.head()

[7]:    age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  active  cardio
     0  23386     1      55      81    130     90            1     1      0     0       1       1
     1  18830     1      57      61    130     90            1     1      0     0       1       1
     2  18426     1      59      58    125     67            1     1      0     0       0       0
     3  19088     1      60      69    110     70            1     1      0     0       0       0
     4  20978     1      64      61    130     70            1     1      0     0       1       0
```

Fig: Dropping the unwanted columns



```
[8]: df1['age'] = df1['age'].div(365)

[10]: df1.head()

[10]:        age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  active  cardio
      0  64.071233     1      55      81    130     90            1     1      0     0       1       1
      1  51.589041     1      57      61    130     90            1     1      0     0       1       1
      2  50.482192     1      59      58    125     67            1     1      0     0       0       0
      3  52.295890     1      60      69    110     70            1     1      0     0       0       0
      4  57.473973     1      64      61    130     70            1     1      0     0       1       0

[11]: df2 = df1.astype({'age':'int'})

[12]: df2.head()

[12]:  age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  alco  active  cardio
      0  64     1      55      81    130     90            1     1      0     0       1       1
      1  51     1      57      61    130     90            1     1      0     0       1       1
      2  50     1      59      58    125     67            1     1      0     0       0       0
```

Fig: Conversion of Age column & Other features Datatypes

3. **Data Analysis:** The analysis of the dataset has been conducted using Apache Spark, leveraging Spark SQL and the DataFrame API for data analysis. Spark 2 incorporates the catalyst optimizer, which enables exceptionally fast execution. Both Spark SQL and the DataFrame API are powered by this optimization engine. The catalyst optimizer can take input from either a SQL query or DataFrame API methods. The Catalog, which contains information about each table from various data sources, is utilized to analyze the inputs and outputs of the logical plan. To

initiate the data analysis process, we created a SparkSession, configuring Spark on our system, and imported essential packages. We then utilized the parallelize function to generate a Spark RDD. Subsequently, we created dataframes and performed SQL queries to analyze the data within our dataset. Our system incorporates several libraries, including Pandas, Matplotlib, NumPy, Scikit-Learn, and Seaborn. These libraries offer numerous advantages for the machine learning process. For instance, Pandas provides efficient data structures and analysis tools, contributing to faster and streamlined data analysis.

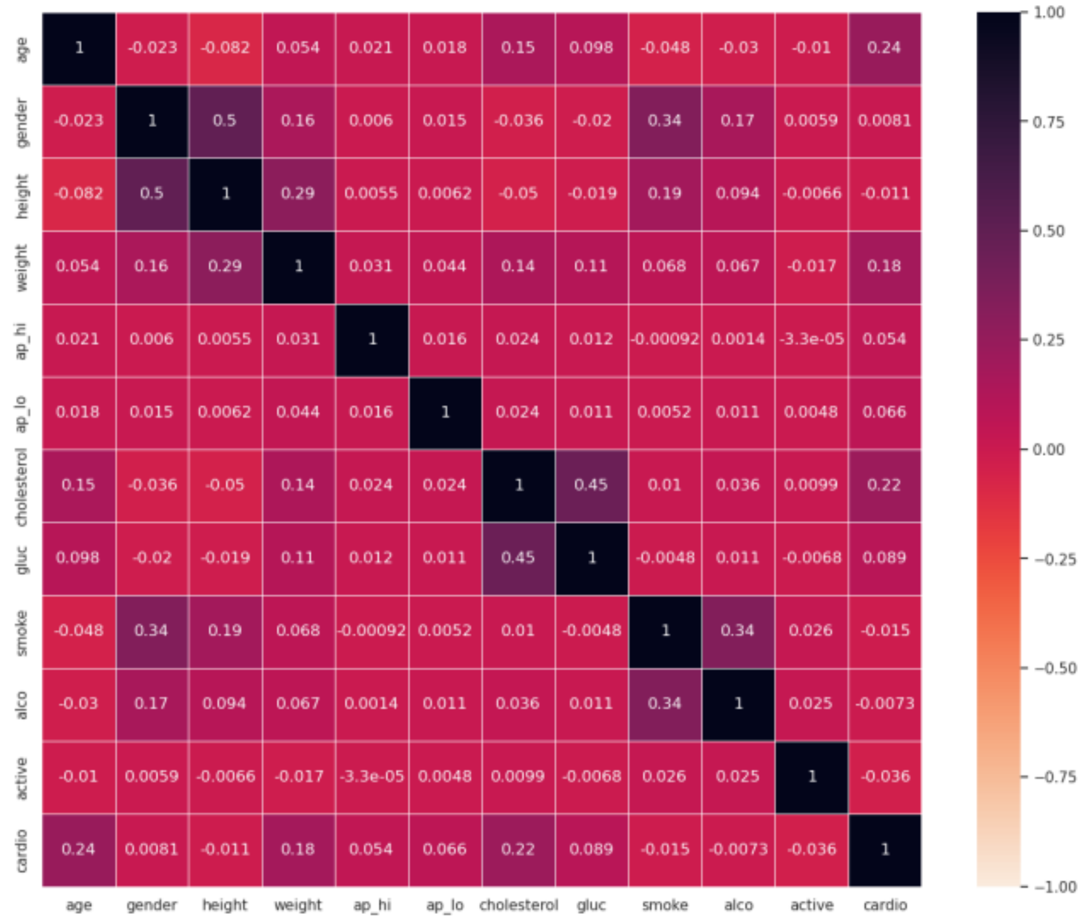
4. **Google Cloud Platform**: To handle Google Cloud projects and resources effectively, we can utilize the user-friendly, web-based interface known as the Google Cloud console. Through this console, we have the option to choose an existing project or create a new one, and subsequently utilize the resources generated within that specific project. Our approach involves creating both a single-node and a three-node Dataproc cluster, uploading analytics jobs and data to Google Cloud Storage, and then executing the jobs on the Spark cluster. This process serves to showcase the capabilities of Cloud Dataproc.

5. Data Visualization: To facilitate data visualization, we utilized the Google Cloud Platform, employing the following steps:

1. We first uploaded the dataset to Google Cloud Storage, which serves as the storage solution for our data.
2. To implement the Spark project, we utilized Dataproc. We created a cluster consisting of one master node and three worker nodes within Dataproc.
3. Within the cluster, we created a PySpark Jupyter notebook. The dataset was read from cloud storage using Dataproc, which allows for seamless integration between cloud storage and HDFS.
4. Using PySpark SQL, we queried the data and saved the results to cloud storage.

5. Subsequently, we saved the findings in Google BigQuery tables to establish a connection with Tableau Online, enabling efficient and effective data visualization.

Correlation Matrix: (Heat Map)

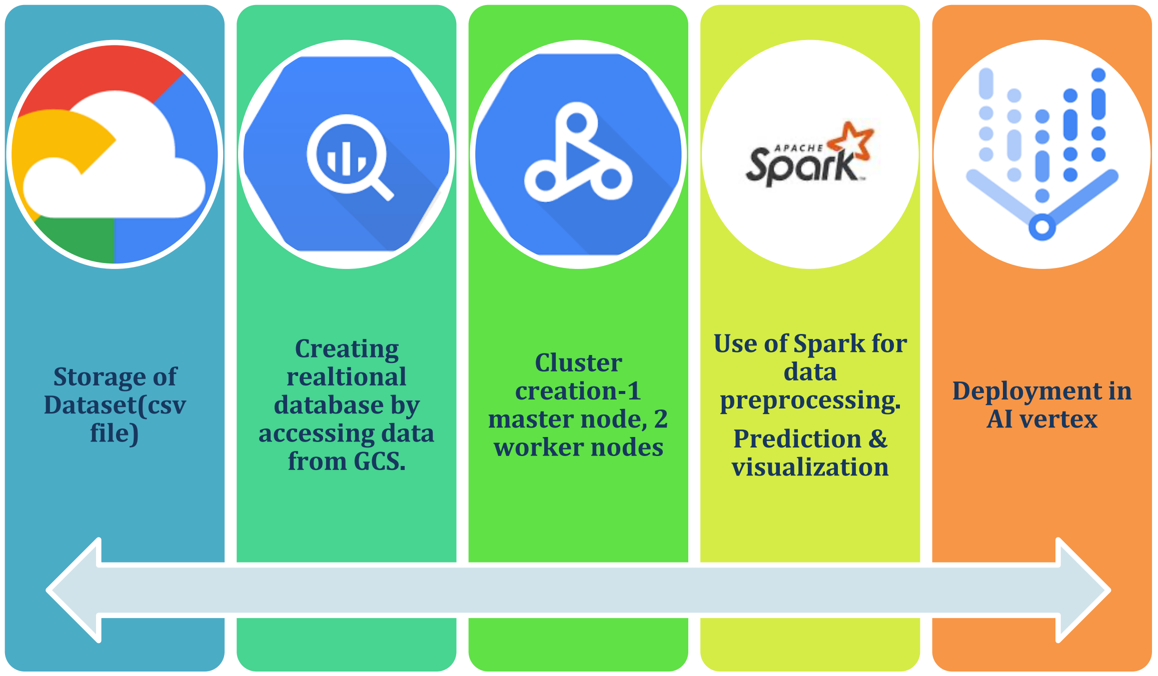


	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
age	1	-0.023	-0.082	0.054	0.021	0.018	0.15	0.098	-0.048	-0.03	-0.01	0.24
gender	-0.023	1	0.5	0.16	0.006	0.015	-0.036	-0.02	0.34	0.17	0.0059	0.0081
height	-0.082	0.5	1	0.29	0.0055	0.0062	-0.05	-0.019	0.19	0.094	-0.0066	-0.011
weight	0.054	0.16	0.29	1	0.031	0.044	0.14	0.11	0.068	0.067	-0.017	0.18
ap_hi	0.021	0.006	0.0055	0.031	1	0.016	0.024	0.012	-0.00092	0.0014	-3.3e-05	0.054
ap_lo	0.018	0.015	0.0062	0.044	0.016	1	0.024	0.011	0.0052	0.011	0.0048	0.066
cholesterol	0.15	-0.036	-0.05	0.14	0.024	0.024	1	0.45	0.01	0.036	0.0099	0.22
gluc	0.098	-0.02	-0.019	0.11	0.012	0.011	0.45	1	-0.0048	0.011	-0.0068	0.089
smoke	-0.048	0.34	0.19	0.068	-0.00092	0.0052	0.01	-0.0048	1	0.34	0.026	-0.015
alco	-0.03	0.17	0.094	0.067	0.0014	0.011	0.036	0.011	0.34	1	0.025	-0.0073
active	-0.01	0.0059	-0.0066	-0.017	-3.3e-05	0.0048	0.0099	-0.0068	0.026	0.025	1	-0.036
cardio	0.24	0.0081	-0.011	0.18	0.054	0.066	0.22	0.089	-0.015	-0.0073	-0.036	1

Fig: Analyzing the greatest correlation variable

6. **BiqQuery**: BigQuery is a popular data warehouse service that allows you to easily work with large amounts of data. By separating the computational engine that analyzes data from your storage options, BigQuery enhances flexibility. BigQuery can be used to evaluate the data's location or to store and analyze your data there. While streaming enables continuous data updates, federated queries allow you to read data from external sources. You can analyze and comprehend that data with the help of potent tools like BigQuery ML and BI Engine.

Architecture and Design:


Storage of Dataset(csv file)
Creating realtional database by accessing data from GCS.
Cluster creation-1 master node, 2 worker nodes
Use of Spark for data preprocessing.
Prediction & visualization
Deployment in AI vertex

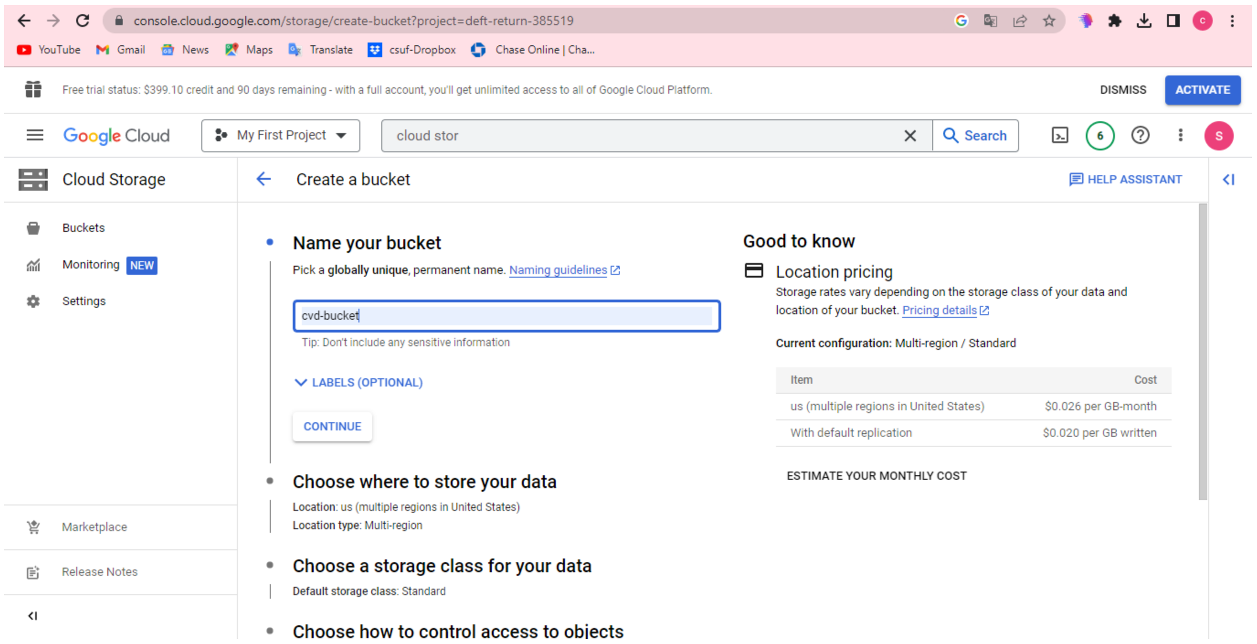
The above architecture is followed for our project:

1. We first added spark to our system. A Spark Session was then made since it can be used to construct Data Frames, register Data Frames as tables, and run SQL over tables. Additionally, it serves as a point of entry for spark context, providing developers with quick access. It serves as the starting point for Spark programming with the Dataset and Data Frame API. We used the getOrCreate() function and the builder pattern method builder() to create a spark session. We created a unique dataset as a data frame from a set of rows.And then as spark session created, we imported our dataset to read csv file into the data frame.

2. PySpark was used to clean up our dataset. Data cleansing is a crucial step in ensuring that the data is accurate and consistent. In this step, duplicates, errors, and irrelevant data were removed. The PIP Install PySpark command was used to first install PySpark.In order to generate DataFrames, register DataFrame as tables, and run SQL over tables, I first built a Spark session. It serves as the foundation for programming Spark using Datasets and DataFrames. After cleaning our data, we used two different approaches to

examine it. After creating data frames and running a few SQL queries, we evaluated the data from our dataset.

3. For data analysis, we have loaded our cleaned dataset in google cloud storage bucket.


console.cloud.google.com/storage/create-bucket?project=deft-return-385519

Free trial status: $399.10 credit and 90 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform.

Cloud Storage — Create a bucket

Name your bucket

Pick a globally unique, permanent name. Naming guidelines

cvd-bucket

Tip: Don't include any sensitive information

LABELS (OPTIONAL)

CONTINUE

Choose where to store your data

Location: us (multiple regions in United States)
Location type: Multi-region

Choose a storage class for your data

Default storage class: Standard

Choose how to control access to objects

Good to know

Location pricing

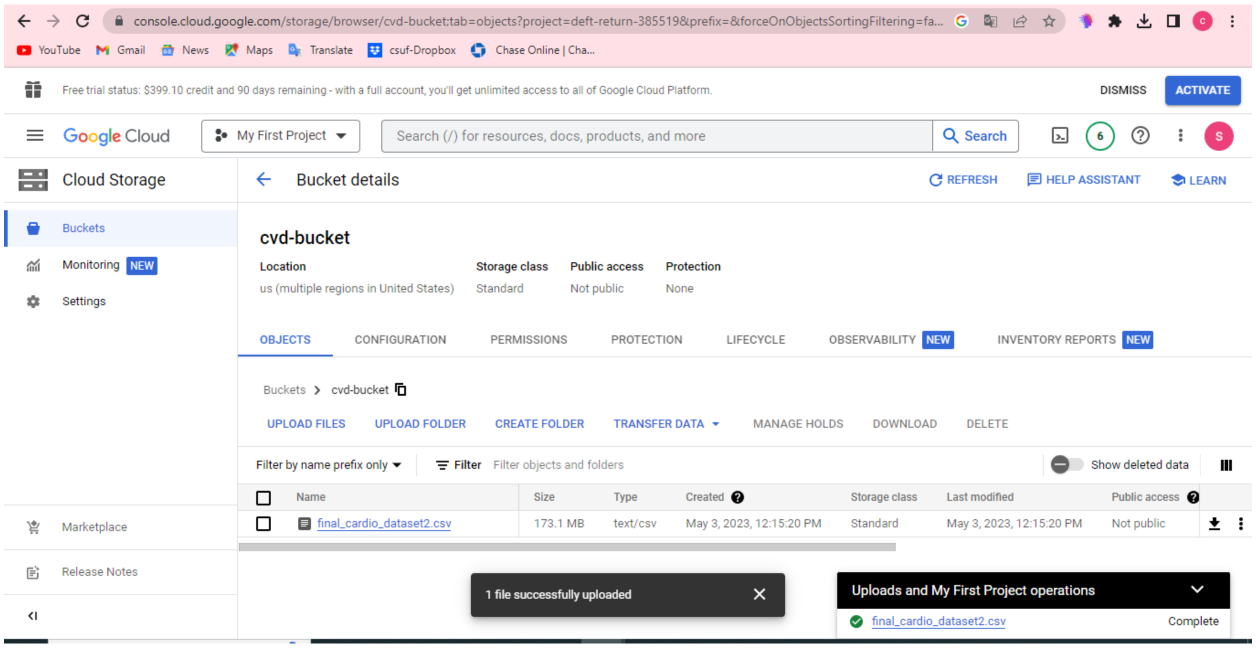
Storage rates vary depending on the storage class of your data and location of your bucket. Pricing details

Current configuration: Multi-region / Standard

Item	Cost
us (multiple regions in United States)	$0.026 per GB-month
With default replication	$0.020 per GB written

ESTIMATE YOUR MONTHLY COST

Fig: Creation of Bucket


console.cloud.google.com/storage/browser/cvd-bucket;tab=objects?project=deft-return-385519&prefix=&forceOnObjectsSortingFiltering=fa...

Free trial status: $399.10 credit and 90 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform.

Cloud Storage — Bucket details

cvd-bucket

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Standard	Not public	None

OBJECTS | CONFIGURATION | PERMISSIONS | PROTECTION | LIFECYCLE | OBSERVABILITY | INVENTORY REPORTS

Buckets > cvd-bucket

Name	Size	Type	Created	Storage class	Last modified	Public access
final_cardio_dataset2.csv	173.1 MB	text/csv	May 3, 2023, 12:15:20 PM	Standard	May 3, 2023, 12:15:20 PM	Not public

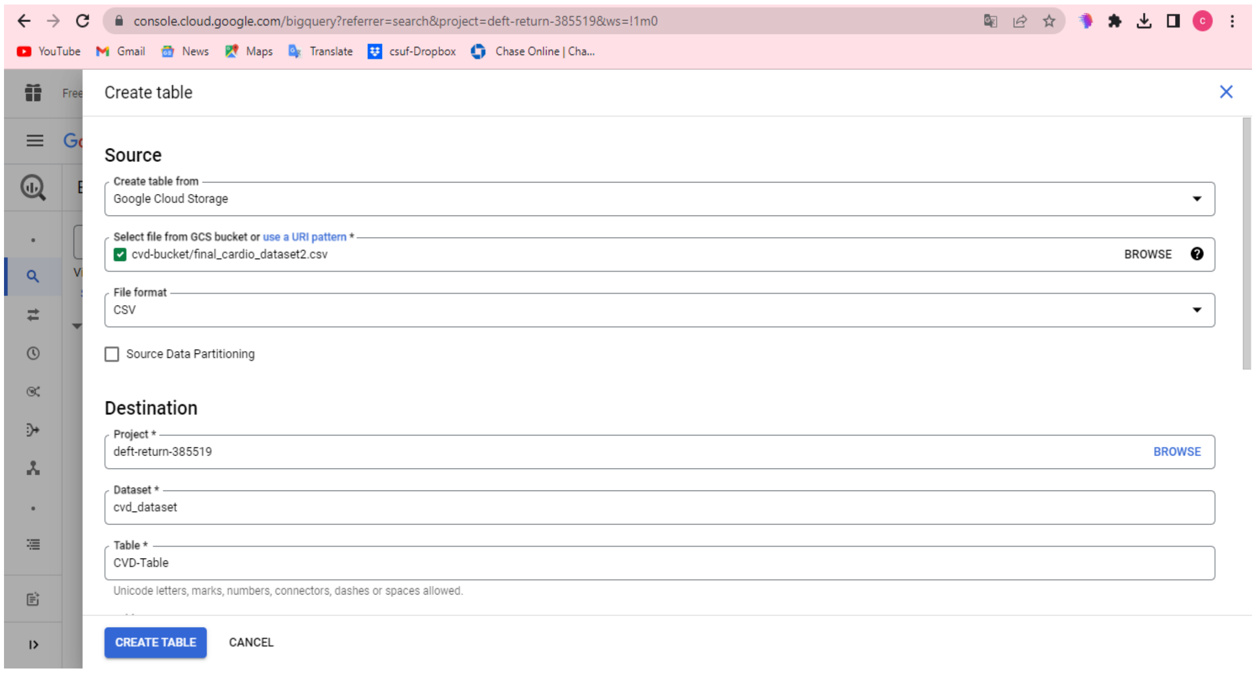
1 file successfully uploaded

Uploads and My First Project operations

final_cardio_dataset2.csv Complete

Fig: Showing the created bucket

And then look for search big query in Google Cloud Platfrom. Big Query takes data from the google cloud storage.


Create table

Source

Create table from: Google Cloud Storage

Select file from GCS bucket or use a URI pattern *: cvd-bucket/final_cardio_dataset2.csv

File format: CSV

Source Data Partitioning

Destination

Project *: deft-return-385519

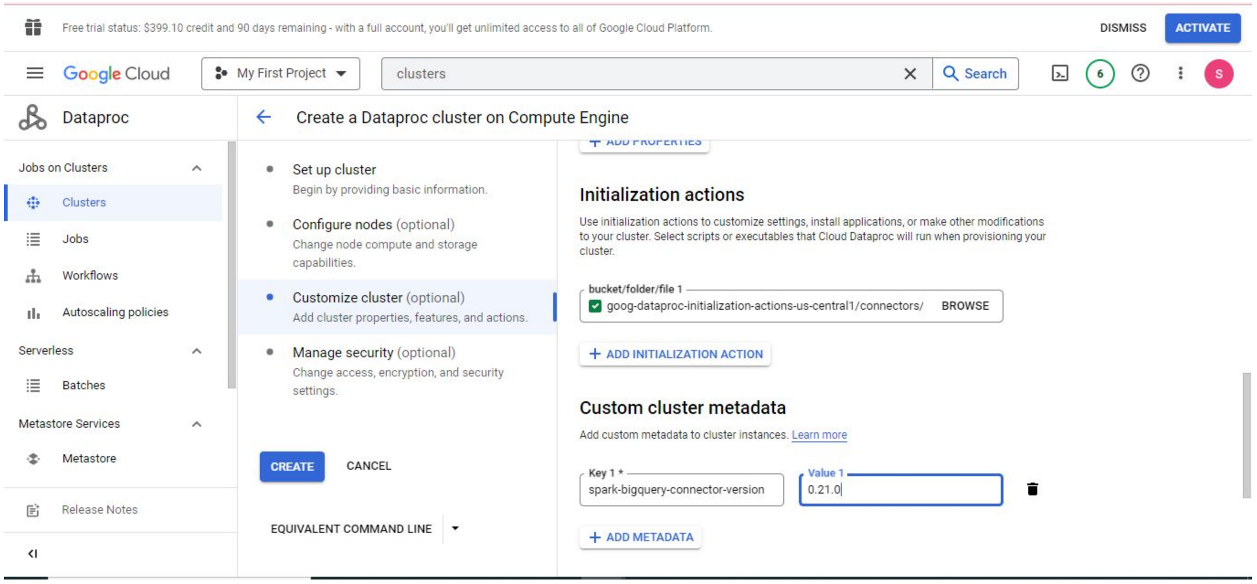
Dataset *: cvd_dataset

Table *: CVD-Table

Unicode letters, marks, numbers, connectors, dashes or spaces allowed.

CREATE TABLE CANCEL

Fig: Creating relational table


Free trial status: $399.10 credit and 90 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform.

Create a Dataproc cluster on Compute Engine

Initialization actions

Use initialization actions to customize settings, install applications, or make other modifications to your cluster. Select scripts or executables that Cloud Dataproc will run when provisioning your cluster.

bucket/folder/file 1: goog-dataproc-initialization-actions-us-central1/connectors/

Custom cluster metadata

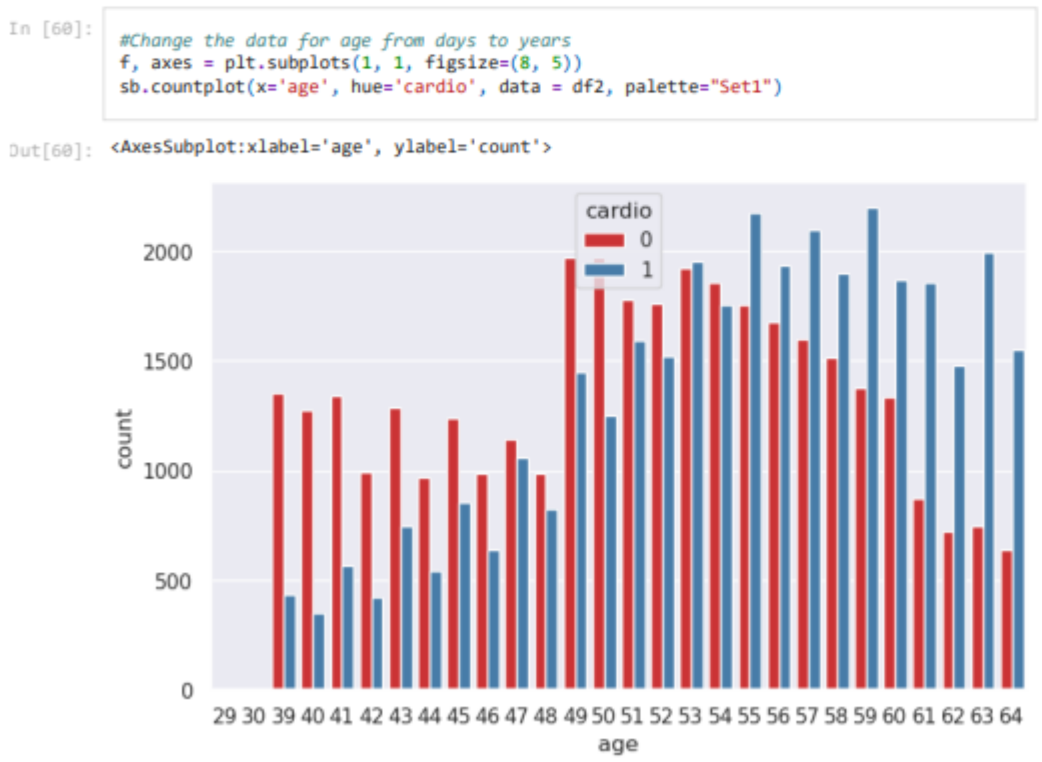
Add custom metadata to cluster instances. Learn more

Key 1 *: spark-bigquery-connector-version

Value 1: 0.21.0

Fig: Bigquery key from Dataproc

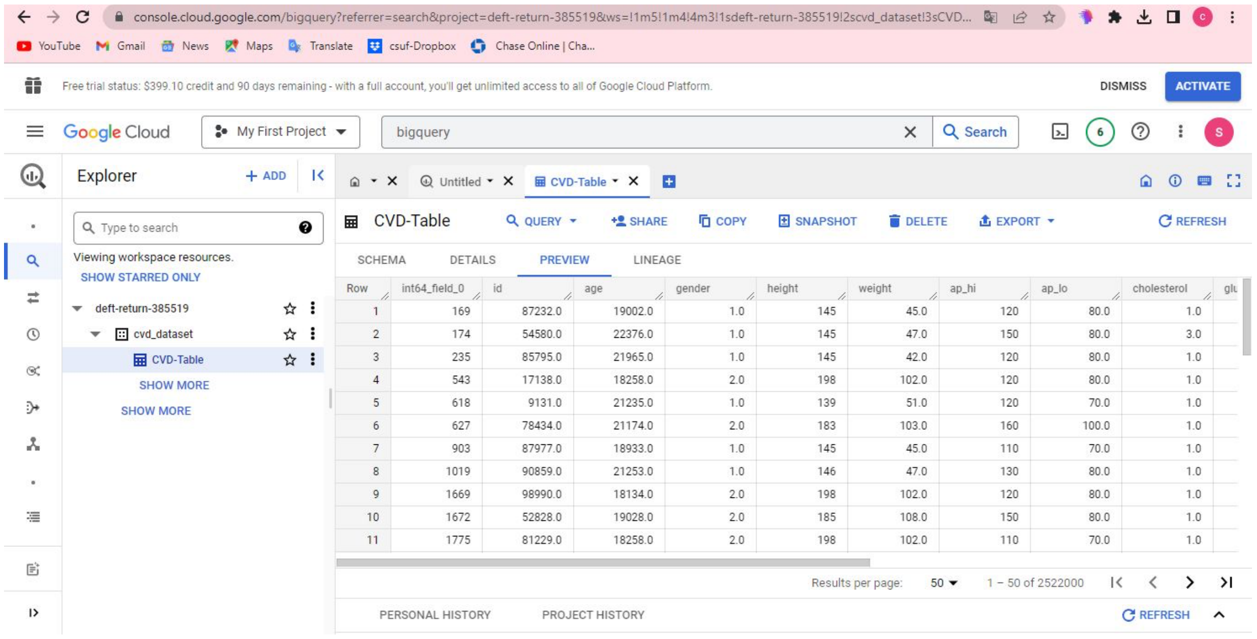
4. For data visualization and analysis, we have used Apache Spark.



```
In [60]:
#Change the data for age from days to years
f, axes = plt.subplots(1, 1, figsize=(8, 5))
sb.countplot(x='age', hue='cardio', data = df2, palette="Set1")

Out[60]: <AxesSubplot:xlabel='age', ylabel='count'>
```

Fig: Visualization in Apache Spark (Age conversion chart)

5. The Google Cloud Storage location where we initially uploaded the dataset.



Row	int64_field_0	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	glu
1	169	87232.0	19002.0	1.0	145	45.0	120	80.0	1.0	
2	174	54580.0	22376.0	1.0	145	47.0	150	80.0	3.0	
3	235	85795.0	21965.0	1.0	145	42.0	120	80.0	1.0	
4	543	17138.0	18258.0	2.0	198	102.0	120	80.0	1.0	
5	618	9131.0	21235.0	1.0	139	51.0	120	70.0	1.0	
6	627	78434.0	21174.0	2.0	183	103.0	160	100.0	1.0	
7	903	87977.0	18933.0	1.0	145	45.0	110	70.0	1.0	
8	1019	90859.0	21253.0	1.0	146	47.0	130	80.0	1.0	
9	1669	98990.0	18134.0	2.0	198	102.0	120	80.0	1.0	
10	1672	52828.0	19028.0	2.0	185	108.0	150	80.0	1.0	
11	1775	81229.0	18258.0	2.0	198	102.0	110	70.0	1.0	

Fig: Viewing Dataset Uploaded

6. Dataproc was used to implement the Spark project. Dataproc generated a cluster with three worker nodes and one master node.

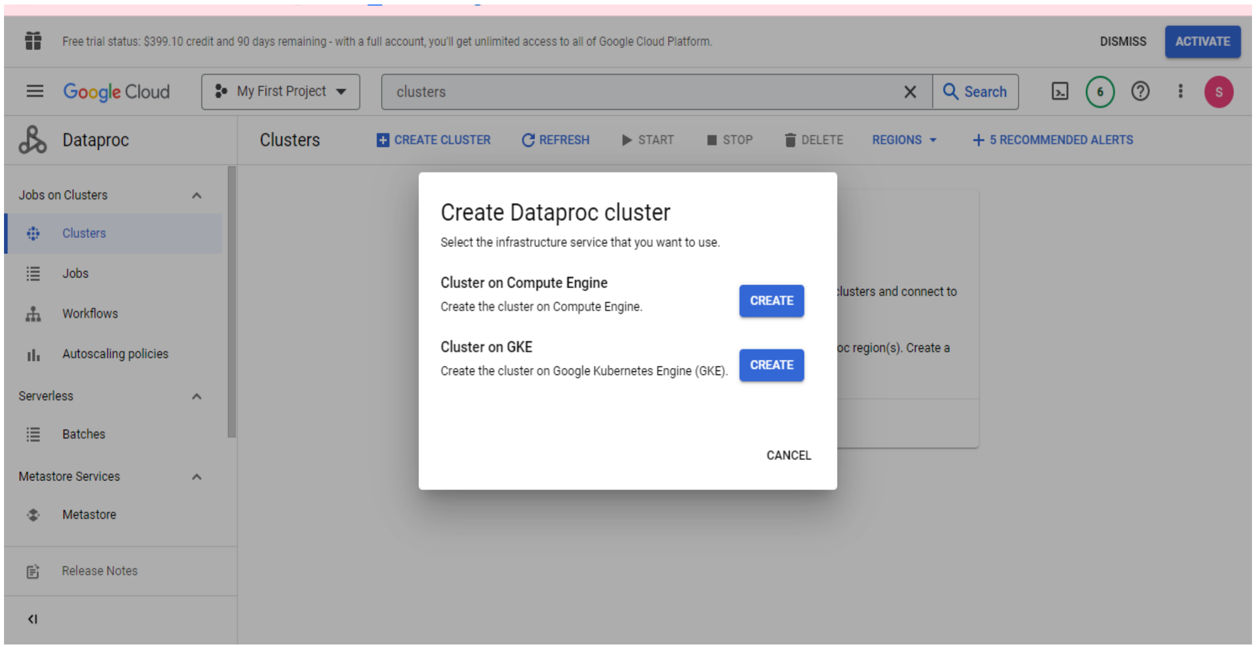
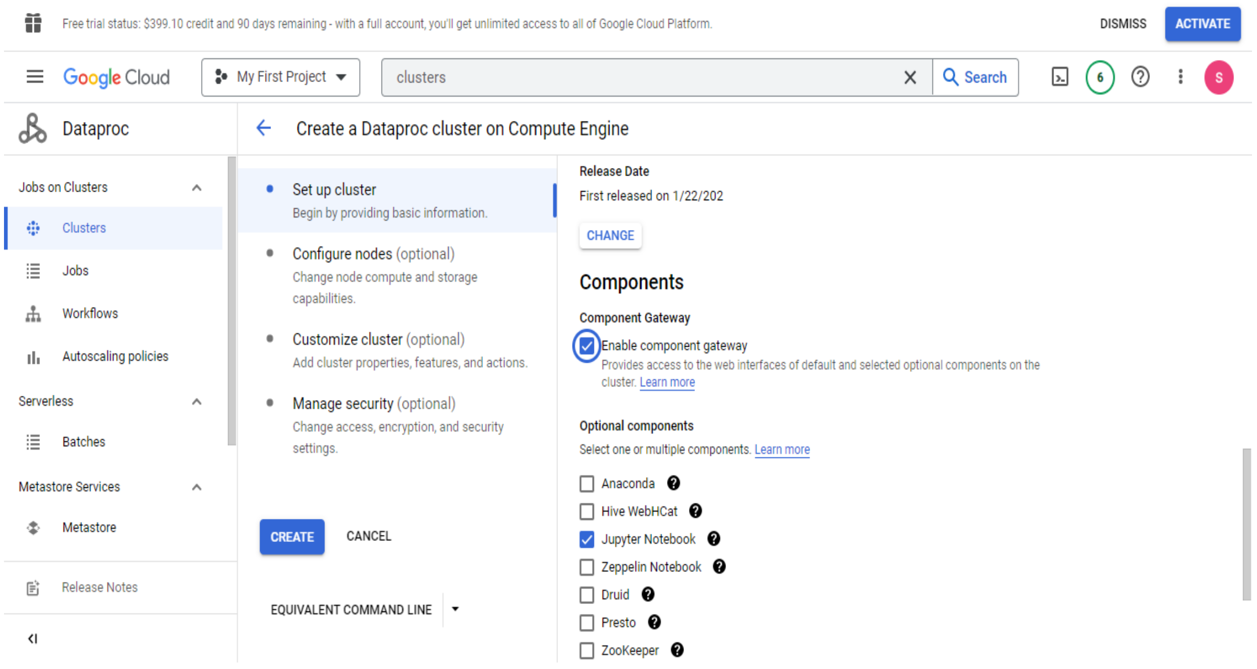

Free trial status: $399.10 credit and 90 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform.
DISMISS
ACTIVATE
Google Cloud
My First Project
clusters
Search
Dataproc
Clusters
CREATE CLUSTER
REFRESH
START
STOP
DELETE
REGIONS
5 RECOMMENDED ALERTS
Jobs on Clusters
Clusters
Jobs
Workflows
Autoscaling policies
Serverless
Batches
Metastore Services
Metastore
Release Notes
Create Dataproc cluster
Select the infrastructure service that you want to use.
Cluster on Compute Engine
Create the cluster on Compute Engine.
CREATE
Cluster on GKE
Create the cluster on Google Kubernetes Engine (GKE).
CREATE
CANCEL

Fig: Creating Cluster



Free trial status: $399.10 credit and 90 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform. DISMISS ACTIVATE

Google Cloud | My First Project | clusters | Search

Dataproc

Create a Dataproc cluster on Compute Engine

- Set up cluster — Begin by providing basic information.
- Configure nodes (optional) — Change node compute and storage capabilities.
- Customize cluster (optional) — Add cluster properties, features, and actions.
- Manage security (optional) — Change access, encryption, and security settings.

CREATE CANCEL

EQUIVALENT COMMAND LINE

Release Date

First released on 1/22/202

CHANGE

Components

Component Gateway

- [x] Enable component gateway

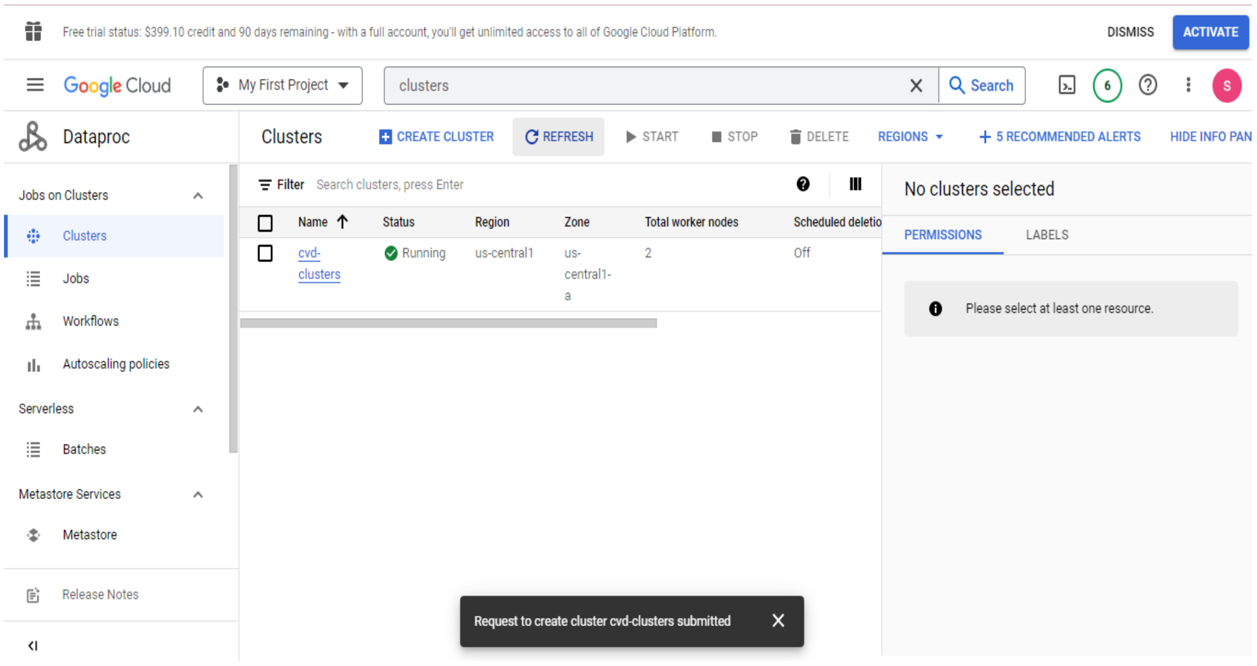
Provides access to the web interfaces of default and selected optional components on the cluster. Learn more

Optional components

Select one or multiple components. Learn more

- [ ] Anaconda
- [ ] Hive WebHCat
- [x] Jupyter Notebook
- [ ] Zeppelin Notebook
- [ ] Druid
- [ ] Presto
- [ ] ZooKeeper

Fig: Web Components to Launch Notebook



Free trial status: $399.10 credit and 90 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform. DISMISS ACTIVATE

Google Cloud | My First Project | clusters | Search

Dataproc

Clusters | CREATE CLUSTER | REFRESH | START | STOP | DELETE | REGIONS | 5 RECOMMENDED ALERTS | HIDE INFO PAN

Filter Search clusters, press Enter

Name	Status	Region	Zone	Total worker nodes	Scheduled deletio
cvd-clusters	Running	us-central1	us-central1-a	2	Off

No clusters selected

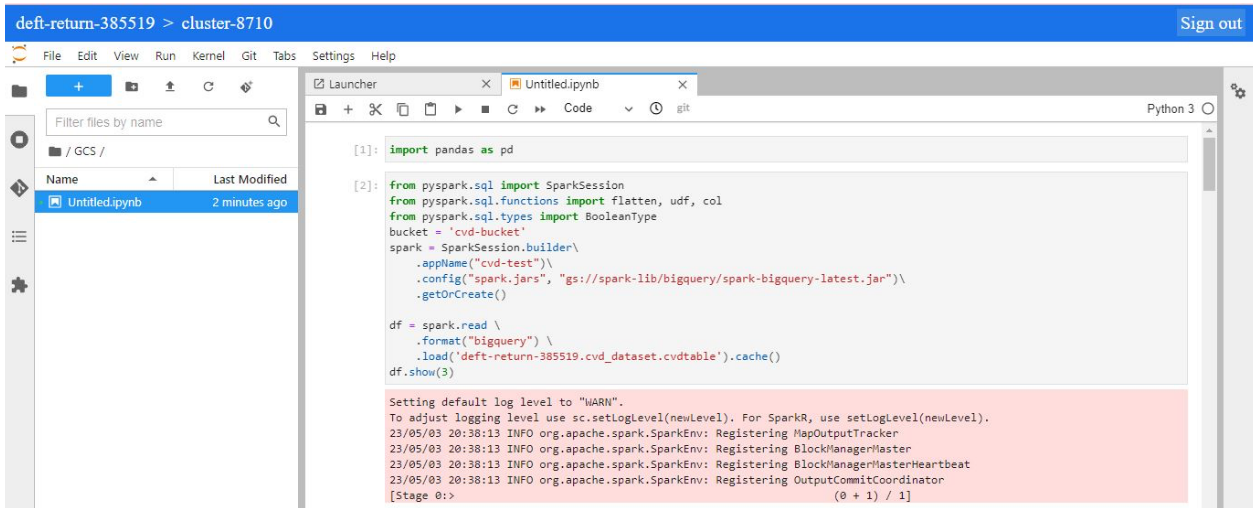
PERMISSIONS LABELS

Please select at least one resource.

Request to create cluster cvd-clusters submitted

Fig: Cluster Created

7. The dataset was read from cloud storage using dataproc, which permits the use of cloud storage in conjunction with HDFS, and then enter into a pyspark jupyter notebook that was established in the cluster.



```
[1]: import pandas as pd

[2]: from pyspark.sql import SparkSession
     from pyspark.sql.functions import flatten, udf, col
     from pyspark.sql.types import BooleanType
     bucket = 'cvd-bucket'
     spark = SparkSession.builder\
         .appName("cvd-test")\
         .config("spark.jars", "gs://spark-lib/bigquery/spark-bigquery-latest.jar")\
         .getOrCreate()

     df = spark.read \
         .format("bigquery") \
         .load('deft-return-385519.cvd_dataset.cvdtable').cache()
     df.show(3)
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/05/03 20:38:13 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/05/03 20:38:13 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/05/03 20:38:13 INFO org.apache.spark.SparkEnv: Registering BlockManagerMasterHeartbeat
23/05/03 20:38:13 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
[Stage 0:>                                                          (0 + 1) / 1]
```

Fig: Creating Pyspark session & Dataframe

8. PySpark SQL is used to query the data, and the results are saved to cloud storage.

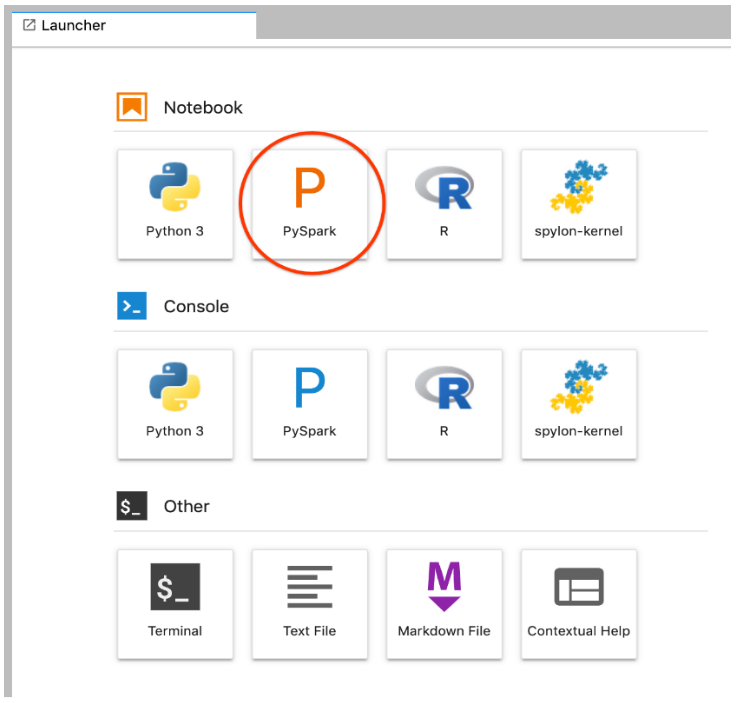
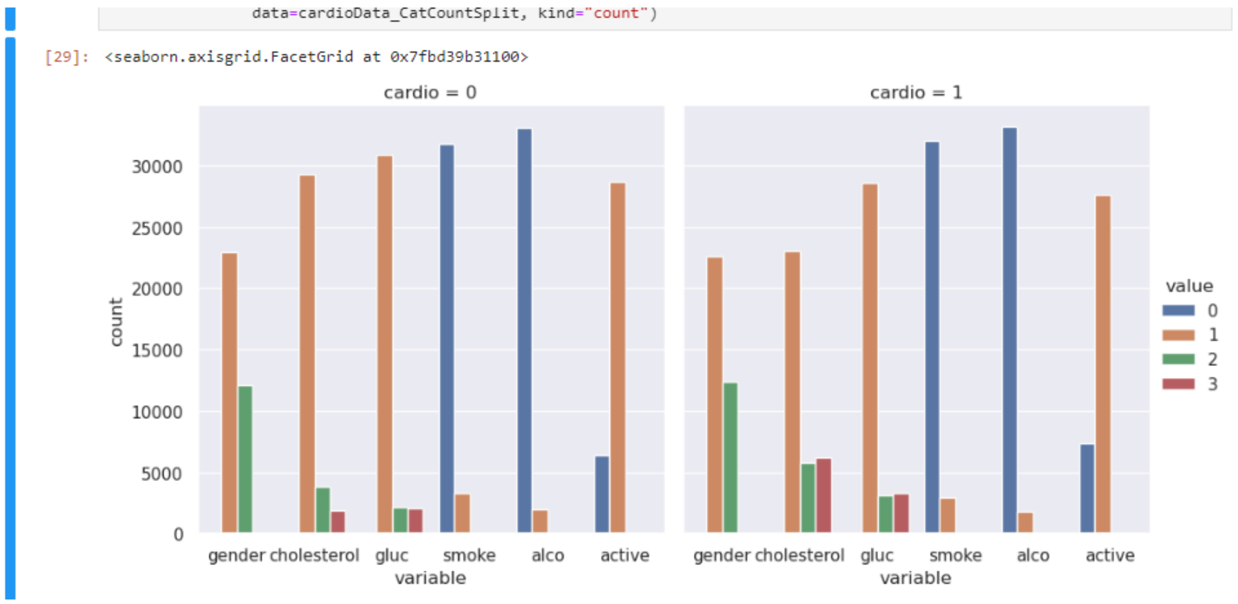


Fig: Launching Pyspark

9. The results are later saved in Google BigQuery tables to enable the connection with Tableau online for data visualization.



```
            data=cardioData_CatCountSplit, kind="count")
[29]: <seaborn.axisgrid.FacetGrid at 0x7fbd39b31100>
```

Fig: Visualization- Graph showing split of Cardio effected people

GITHUB LOCATION:

Here's the GitHub repository for our project: https://github.com/sirishaa03/CARDIOVASCULAR-DISEASE-PREDICTION

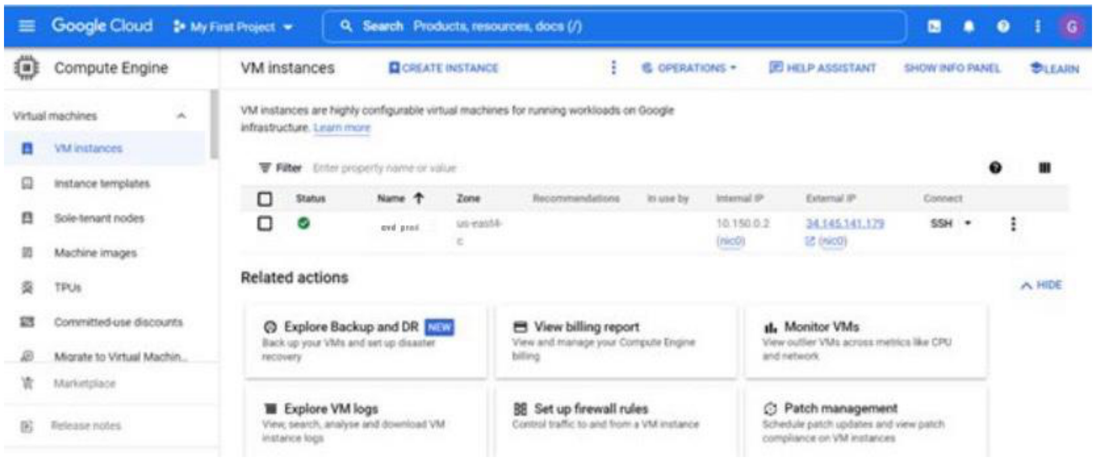
DEPLOYMENT INSTRUCTIONS

1) Install spark from the Apache spark website.
2) Install Jupyter and python component and set up the environment variables so that we can work on Jupyter notebook.
3) Setup a cluster in Google cloud platform and create a Dataproc in it.
4) And then need to install libraries that are required for the code and then create a bucket where it is cloud storage so that the dataset will be loaded.

5) Then search for Big Query, by using the big queries we can analyze and visualize the data using Jupyter notebook.

STEPS TO RUN

Step 1: Login to the Google Cloud Platform (GCP) application, create a new project and then create a VM instance.



VM instances are highly configurable virtual machines for running workloads on Google infrastructure. Learn more

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	ovd prod	us-east4-c			10.150.0.2 (nic0)	34.145.141.179 (nic0)	SSH

Explore Backup and DR
Back up your VMs and set up disaster recovery

View billing report
View and manage your Compute Engine billing

Monitor VMs
View outlier VMs across metrics like CPU and network

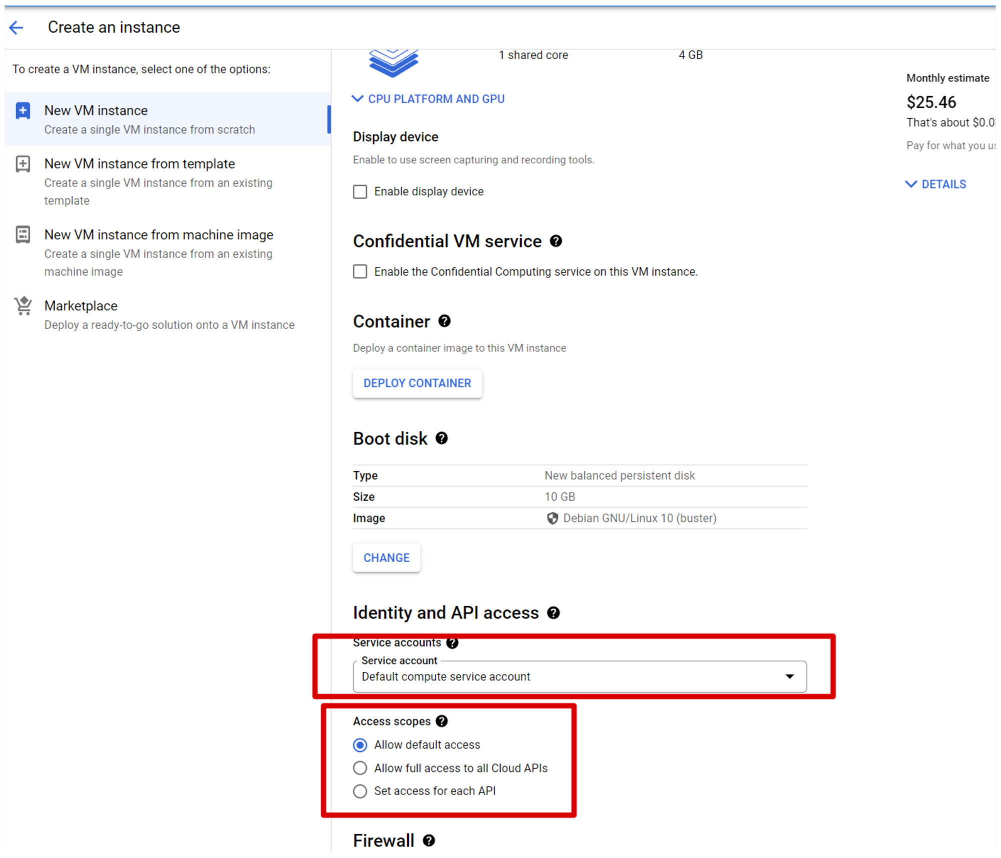
Explore VM logs
View, search, analyse and download VM instance logs

Set up firewall rules
Control traffic to and from a VM instance

Patch management
Schedule patch updates and view patch compliance on VM instances

Fig: Creating VM instance

Step 2: Change the firewall settings and start the VM instance.



Create an instance

To create a VM instance, select one of the options:

New VM instance
Create a single VM instance from scratch

New VM instance from template
Create a single VM instance from an existing template

New VM instance from machine image
Create a single VM instance from an existing machine image

Marketplace
Deploy a ready-to-go solution onto a VM instance

1 shared core 4 GB

CPU PLATFORM AND GPU

Monthly estimate
$25.46
That's about $0.0
Pay for what you u

DETAILS

Display device
Enable to use screen capturing and recording tools.
Enable display device

Confidential VM service
Enable the Confidential Computing service on this VM instance.

Container
Deploy a container image to this VM instance
DEPLOY CONTAINER

Boot disk

Type	New balanced persistent disk
Size	10 GB
Image	Debian GNU/Linux 10 (buster)

CHANGE

Identity and API access
Service accounts
Service account: Default compute service account

Access scopes
[✓ Allow default access]
Allow full access to all Cloud APIs
Set access for each API

Firewall

Fig: Firwall Settings

Step 3: Need to Install Jupyter notebook and other packages to setup the environment, with the help of SSH terminal.

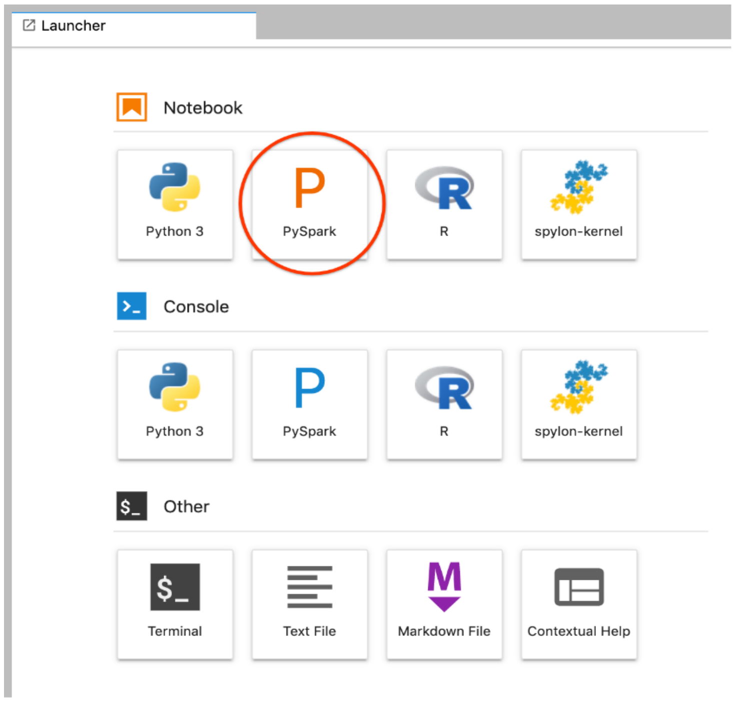


Fig: Launching Pyspark

Step 4: Need to set up the VM server and then must launch the Jupyter notebook where .py file needs to be used to see our data visualization.

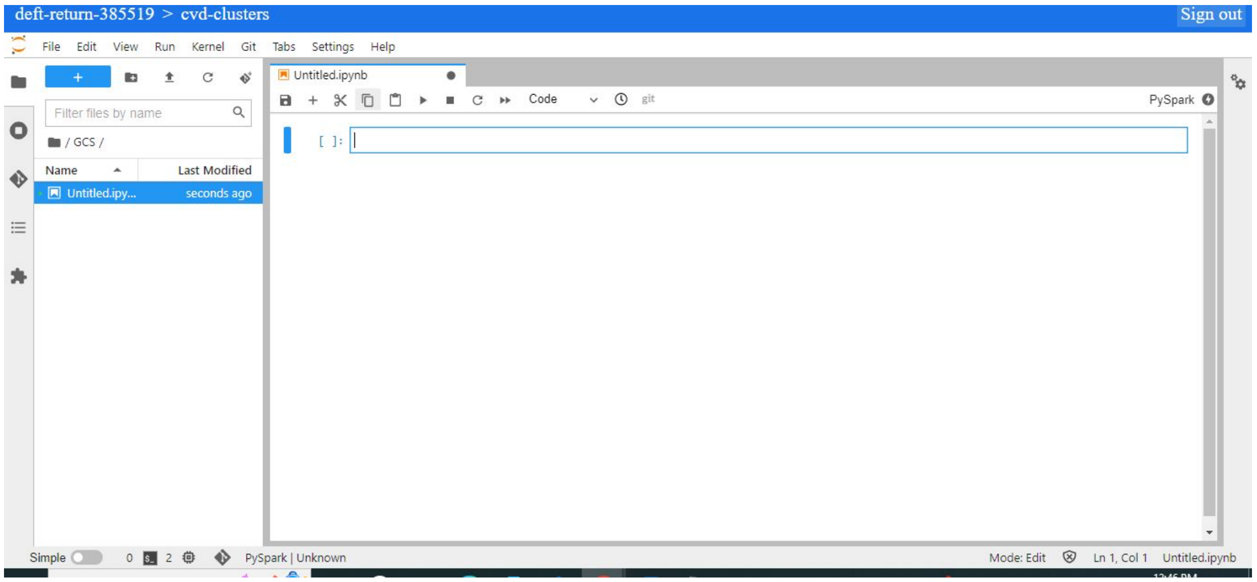
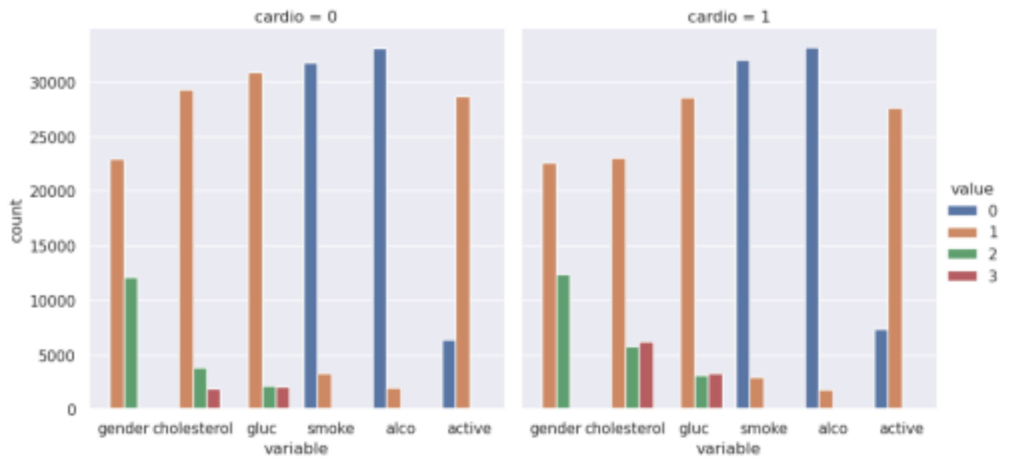


Fig: Launching Jupyter Notebook



```
[61]: pip install plotly
```

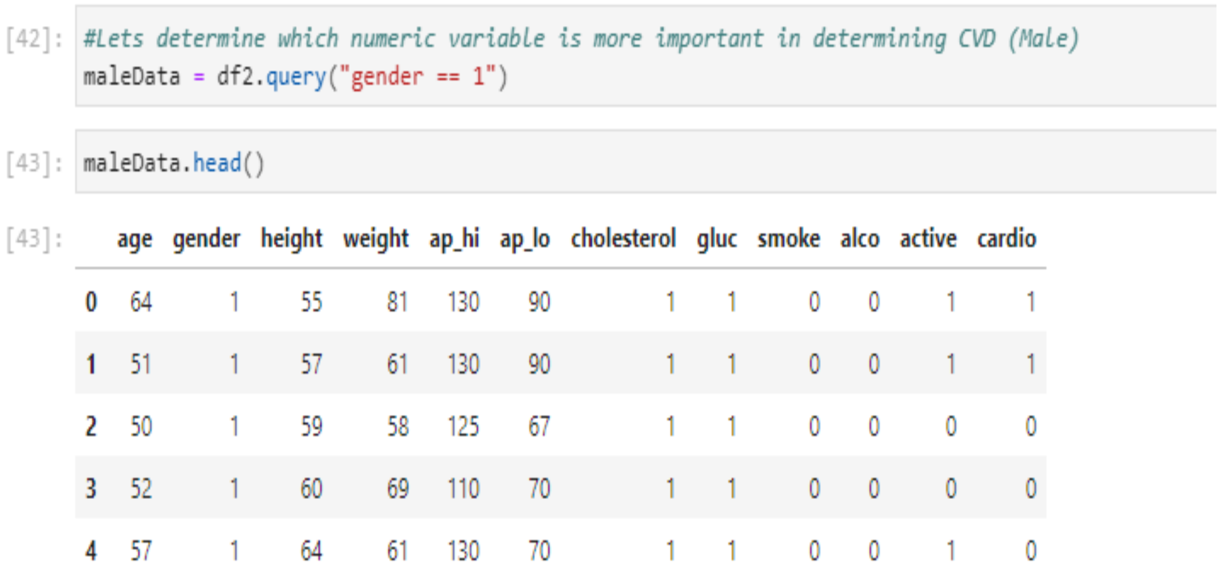
```
Requirement already satisfied: plotly in /opt/conda/miniconda3/lib/python3.8/site-packag
es (5.14.1)
Requirement already satisfied: tenacity>=6.2.0 in /opt/conda/miniconda3/lib/python3.8/si
te-packages (from plotly) (8.2.2)
Requirement already satisfied: packaging in /opt/conda/miniconda3/lib/python3.8/site-pac
kages (from plotly) (23.1)
```

Fig: Data Visualization

Step 5: Choose a deployment environment: **AI Vertex** We will need to create an account and choose an appropriate deployment environment, such as a virtual machine or a container. Since AI Vertex (cloud- based service provided by google cloud) is a paid platform for deploying ML models and this is the primary limitation for us to move forward with our project. Further, it also has restrictions on the number of free resources we can use and the duration of deployment. Hence, we are looking for other alternatives to deploy the model. (TensorFlow Serving, Kubernetes Engine and more)

TEST RESULTS

1. Determining which numeric variable is more important in determining CVD (Male)



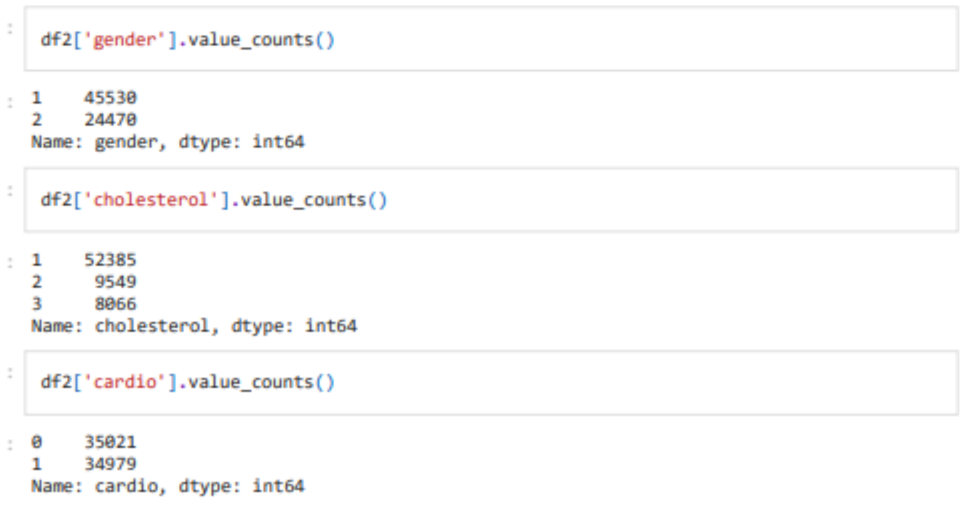
```
[42]: #Lets determine which numeric variable is more important in determining CVD (Male)
      maleData = df2.query("gender == 1")
```

```
[43]: maleData.head()
```

[43]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	64	1	55	81	130	90	1	1	0	0	1	1
1	51	1	57	61	130	90	1	1	0	0	1	1
2	50	1	59	58	125	67	1	1	0	0	0	0
3	52	1	60	69	110	70	1	1	0	0	0	0
4	57	1	64	61	130	70	1	1	0	0	1	0

2. Value counts of important features considered for prediction.



```
df2['gender'].value_counts()
```

```
1    45530
2    24470
Name: gender, dtype: int64
```

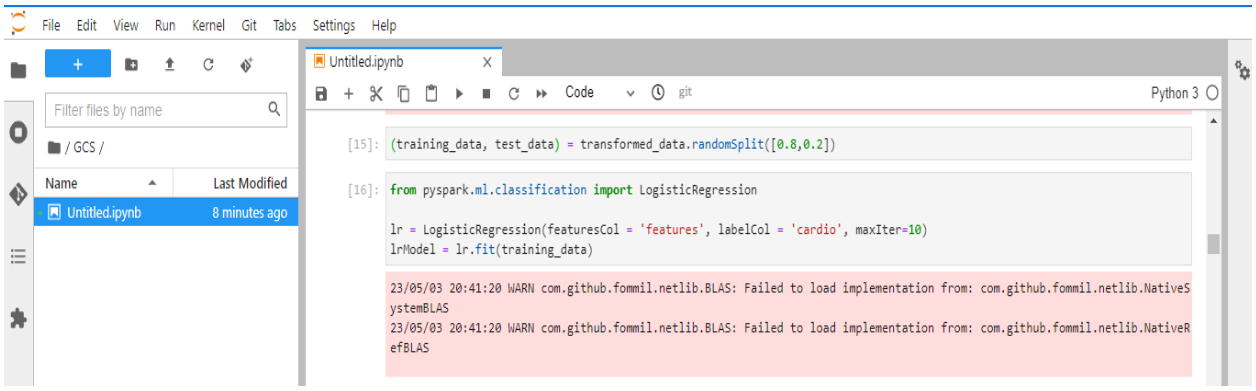
```
df2['cholesterol'].value_counts()
```

```
1    52385
2     9549
3     8066
Name: cholesterol, dtype: int64
```

```
df2['cardio'].value_counts()
```

```
0    35021
1    34979
Name: cardio, dtype: int64
```

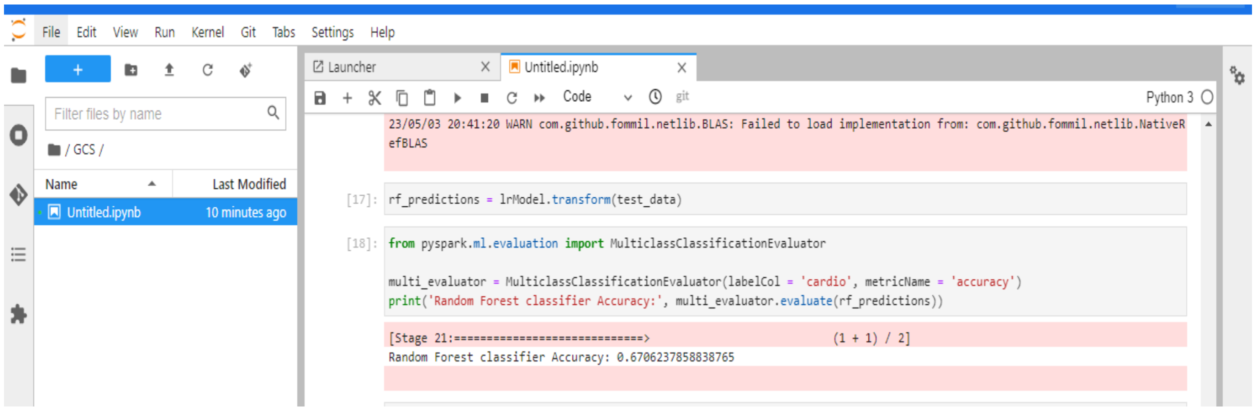
3. Test train split to 80% and 20%.



```
[15]: (training_data, test_data) = transformed_data.randomSplit([0.8,0.2])

[16]: from pyspark.ml.classification import LogisticRegression

      lr = LogisticRegression(featuresCol = 'features', labelCol = 'cardio', maxIter=10)
      lrModel = lr.fit(training_data)

23/05/03 20:41:20 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeS
ystemBLAS
23/05/03 20:41:20 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeR
efBLAS
```

4. The maximum Accuracy predicted is as follows:

We have trained with Random Forest Classifier Algorithm to predict the CVD risk probability. The best obtained accuracy is 67%.



```
23/05/03 20:41:20 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeR
efBLAS

[17]: rf_predictions = lrModel.transform(test_data)

[18]: from pyspark.ml.evaluation import MulticlassClassificationEvaluator

      multi_evaluator = MulticlassClassificationEvaluator(labelCol = 'cardio', metricName = 'accuracy')
      print('Random Forest classifier Accuracy:', multi_evaluator.evaluate(rf_predictions))

[Stage 21:============================>                          (1 + 1) / 2]
Random Forest classifier Accuracy: 0.6706237858838765
```

CONCLUSION

Apache Spark can be a powerful tool for cardiovascular disease (CVD) risk prediction due to its distributed computing architecture and machine learning capabilities. It enables faster and more efficient data processing of large-scale datasets, improving the accuracy and speed of CVD risk prediction models. Spark can preprocess and analyze CVD risk factors, identify patterns and correlations, and train predictive models using machine learning libraries such as MLlib and TensorFlow. Spark's streaming and machine learning capabilities can also be leveraged to develop real-time CVD risk prediction systems that enable timely interventions and personalized treatment plans. As the prevalence of CVD continues to rise globally, Spark's use in CVD risk prediction is likely to become increasingly important in improving patient outcomes. Its ability to handle large-scale data, complex algorithms, and real-time streaming makes it a valuable tool for healthcare providers in CVD prevention and management.

FUTURE WORK

The potential areas of future work in the field of cardiovascular disease (CVD) prediction using Hadoop and Apache Spark include the integration of additional data sources, exploration of novel machine learning techniques, improving model explainability, incorporation of lifestyle and behavioral factors, development of real-time prediction models, and deployment of models in clinical settings. By incorporating wearable devices, social media, and patient-generated data, machine learning models can provide more accurate and personalized risk assessments. Advanced techniques like reinforcement learning and transfer learning can improve model accuracy, while SHAP and LIME can enhance model interpretability. Considering lifestyle and behavioral factors and building real-time prediction models are crucial for early detection and intervention for high-risk patients.These advancements could lead to more accurate and personalized risk assessments for patients, early detection and intervention for high-risk patients, and ultimately reduce the burden of CVD on healthcare systems while improving patient outcomes.