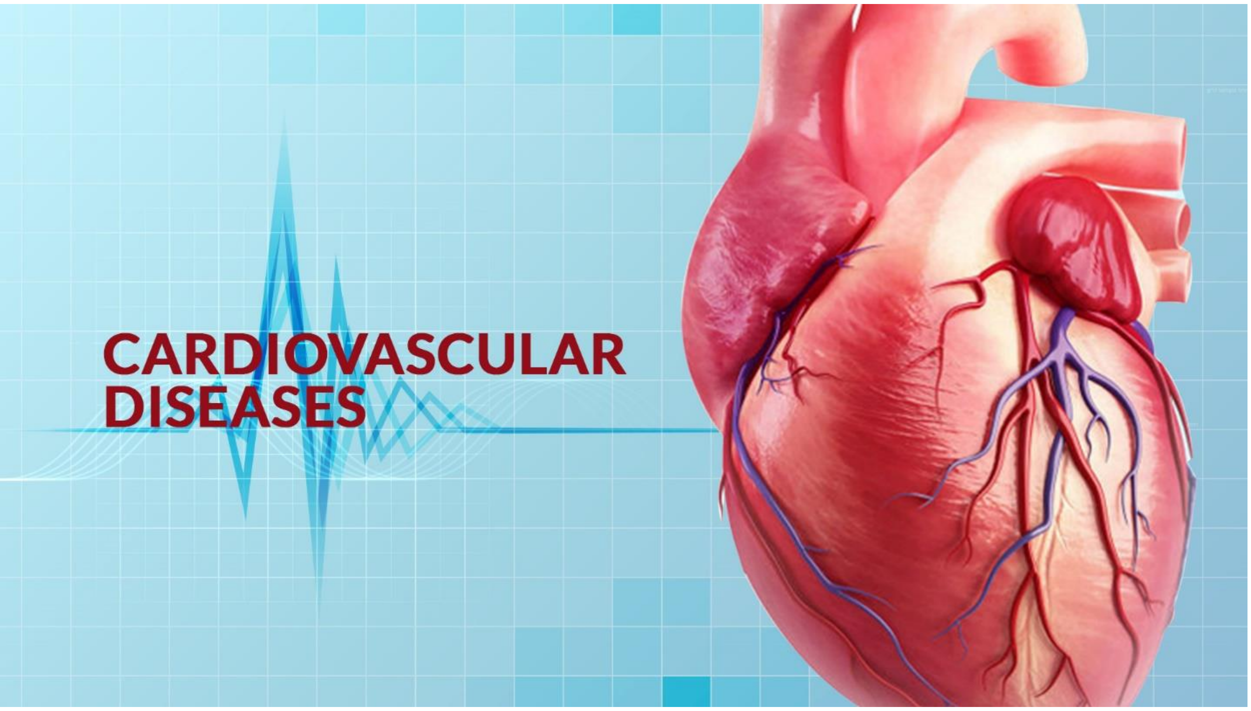
Context:

Cardiovascular disease is becoming increasingly prevalent. As a data science expert, you have been tasked with analyzing patient records to identify the factors that contribute to this problem. You will use IBM Cloud analysis tools to speed up the process.



Assignment:

- Firstly, you should study to understand dataset. The data is provided for you as a csv file.
- After that, you will revise how to upload data to IBM DB2 and do basic queries. Next, you will make connection to db2 using ibm_db API in python. Ibm_db API will enable you to do the same queries with the IBM DB2 console commands.
- Next, you will download data from DB2 to run offline analysis.
- Provided code show step by step how to study about data and how to build a system to classify the problem.
- After that, you should do all steps again on Watson studio and upload results to github.

Cardiovascular Disease dataset

About Dataset

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

Data description:

There are 3 types of input features:

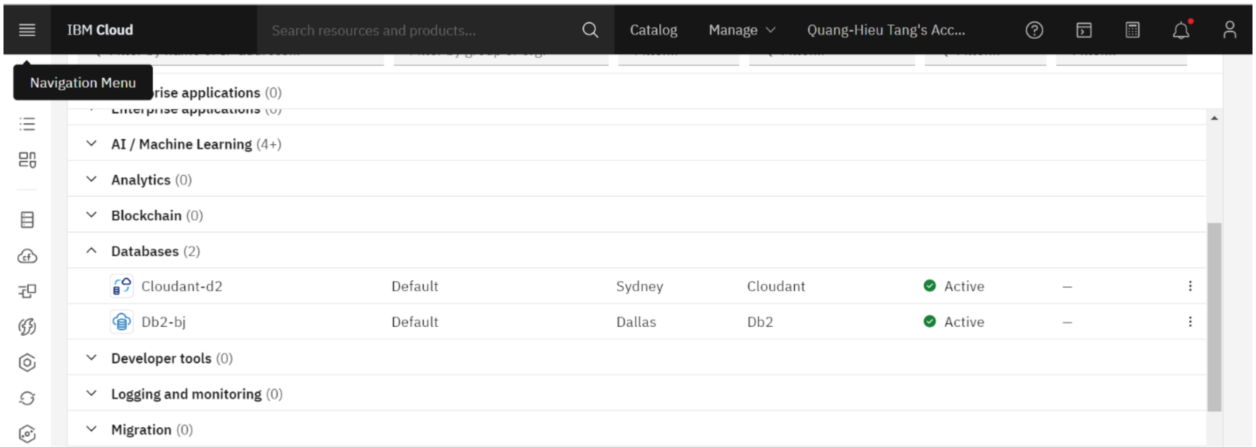
- *Objective*: factual information;
- *Examination*: results of medical examination;
- *Subjective*: information given by the patient.

Features:

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

IBM DB2:

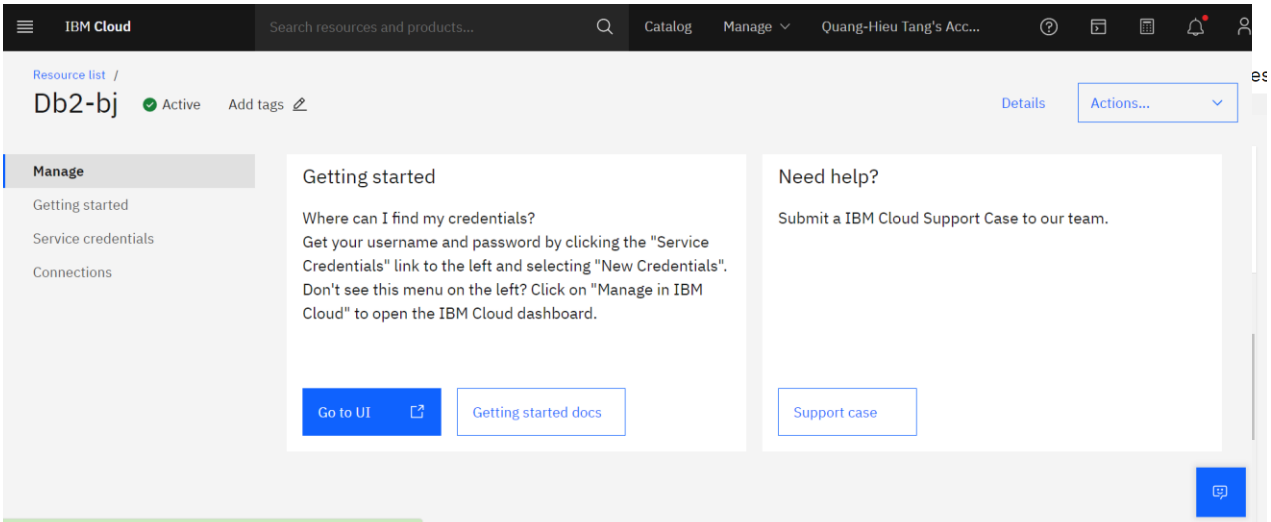
Upload the file into IBM DB2 cloud. You can do like this:



- Enterprise applications (0)
- AI / Machine Learning (4+)
- Analytics (0)
- Blockchain (0)
- Databases (2)

Cloudant-d2	Default	Sydney	Cloudant	Active	–
Db2-bj	Default	Dallas	Db2	Active	–

- Developer tools (0)
- Logging and monitoring (0)
- Migration (0)



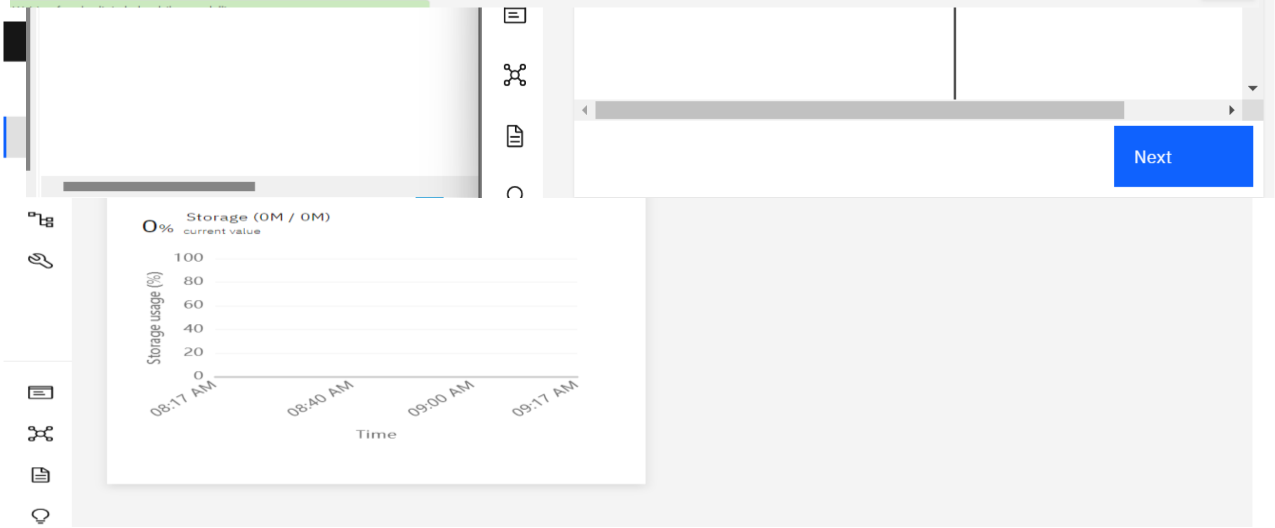
Db2-bj Active Add tags

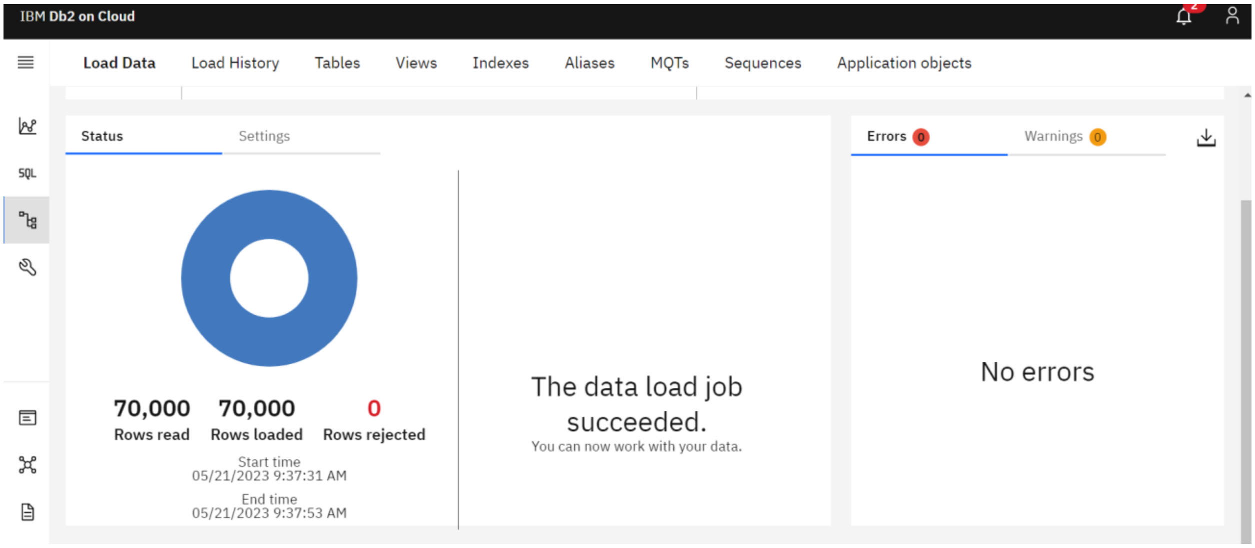
Getting started

Where can I find my credentials?
Get your username and password by clicking the "Service Credentials" link to the left and selecting "New Credentials". Don't see this menu on the left? Click on "Manage in IBM Cloud" to open the IBM Cloud dashboard.

Need help?

Submit a IBM Cloud Support Case to our team.





70,000 Rows read

70,000 Rows loaded

0 Rows rejected

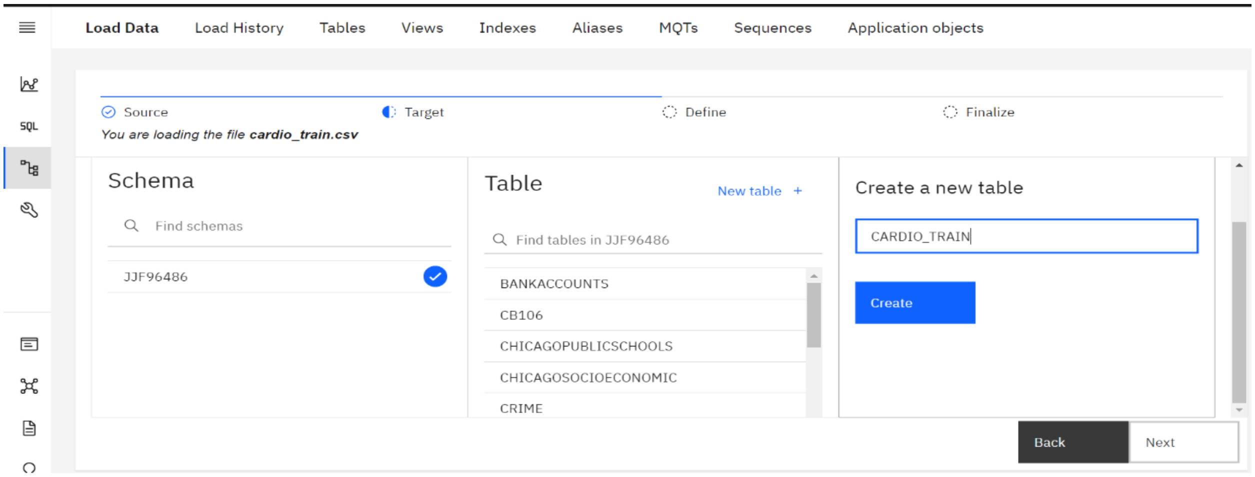
Start time 05/21/2023 9:37:31 AM

End time 05/21/2023 9:37:53 AM

The data load job succeeded.

You can now work with your data.

No errors

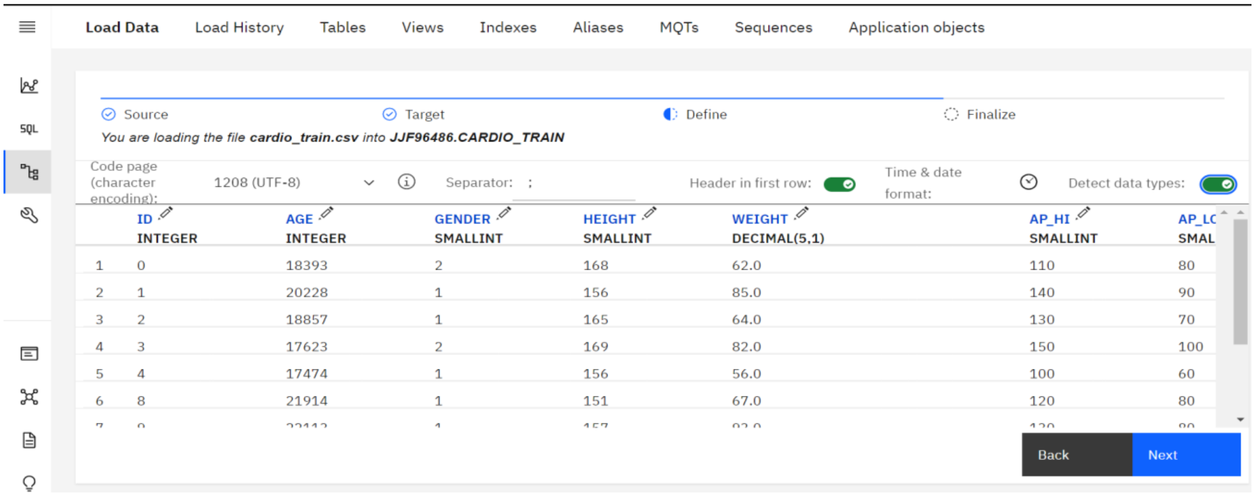


You are loading the file *cardio_train.csv*

Schema: JJF96486 ✓

Tables: BANKACCOUNTS, CB106, CHICAGOPUBLICSCHOOLS, CHICAGOSOCIOECONOMIC, CRIME

Create a new table: CARDIO_TRAIN



You are loading the file *cardio_train.csv* into *JJF96486.CARDIO_TRAIN*

Code page (character encoding): 1208 (UTF-8)

Separator: ;

Header in first row: ●

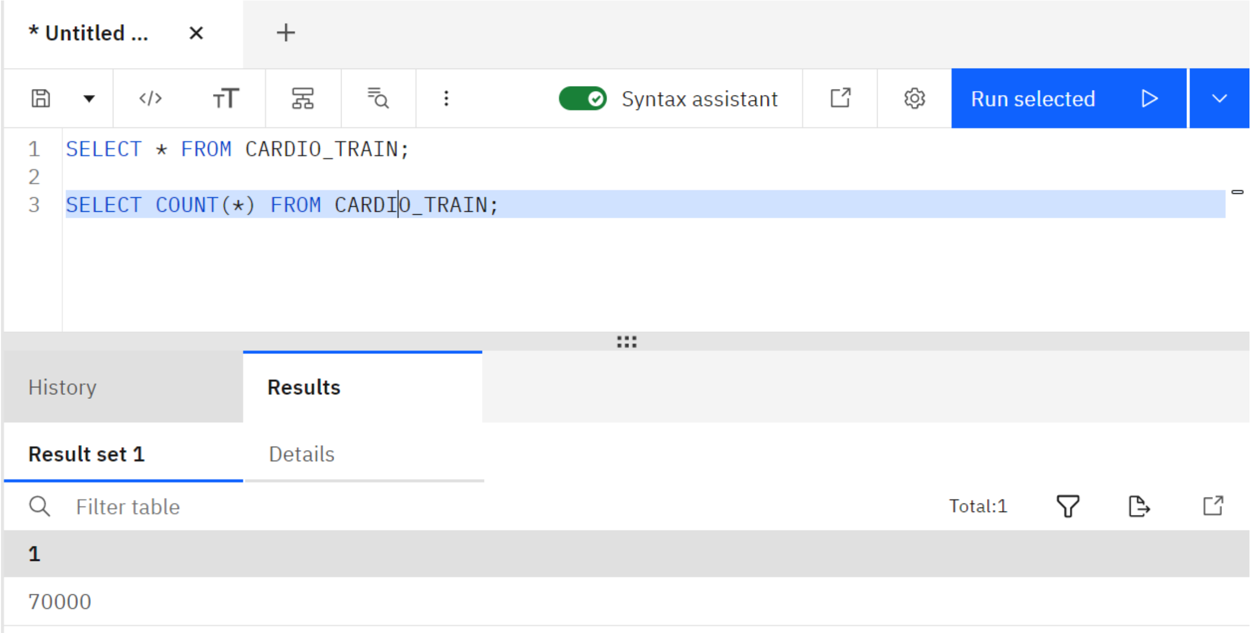
Time & date format:

Detect data types: ●

	ID INTEGER	AGE INTEGER	GENDER SMALLINT	HEIGHT SMALLINT	WEIGHT DECIMAL(5,1)	AP_HI SMALLINT	AP_LO SMAL
1	0	18393	2	168	62.0	110	80
2	1	20228	1	156	85.0	140	90
3	2	18857	1	165	64.0	130	70
4	3	17623	2	169	82.0	150	100
5	4	17474	1	156	56.0	100	60
6	8	21914	1	151	67.0	120	80
7	9	22113	1	157	93.0	130	80

Make queries:

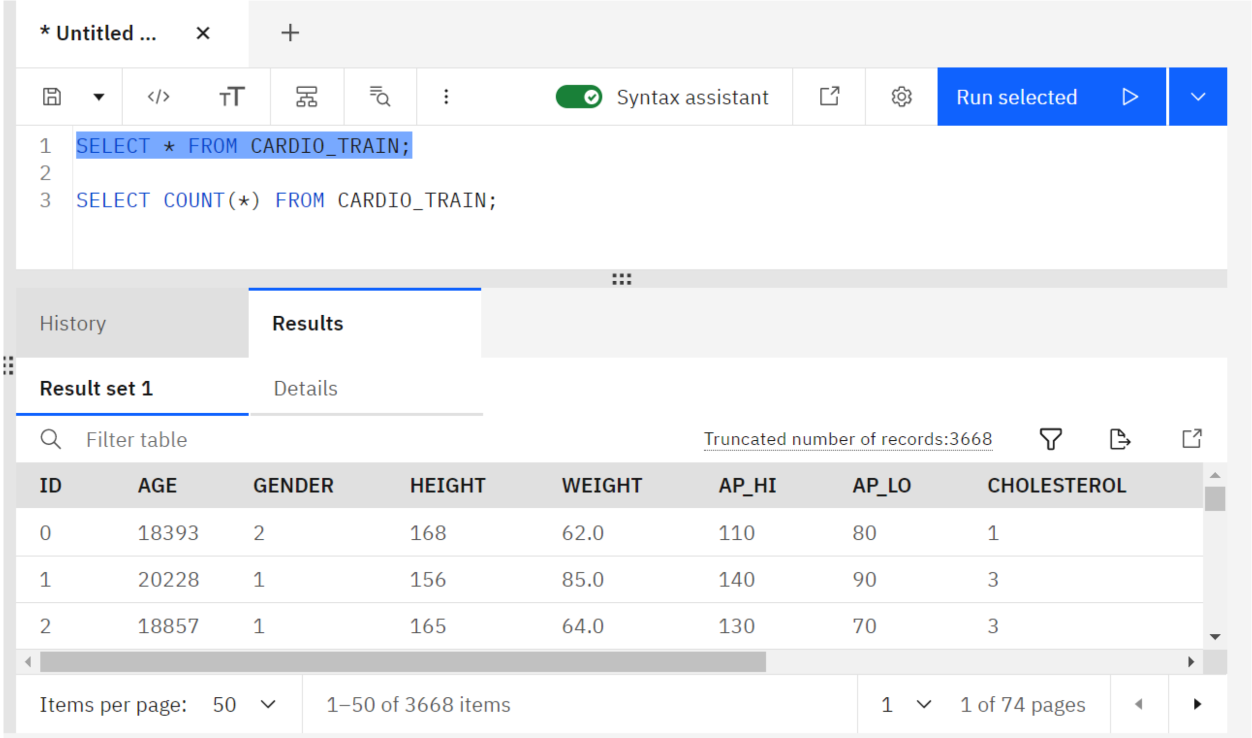
- Count number of rows in the dataset:



```
SELECT * FROM CARDIO_TRAIN;

SELECT COUNT(*) FROM CARDIO_TRAIN;
```

```
1
70000
```

- Checking some rows:



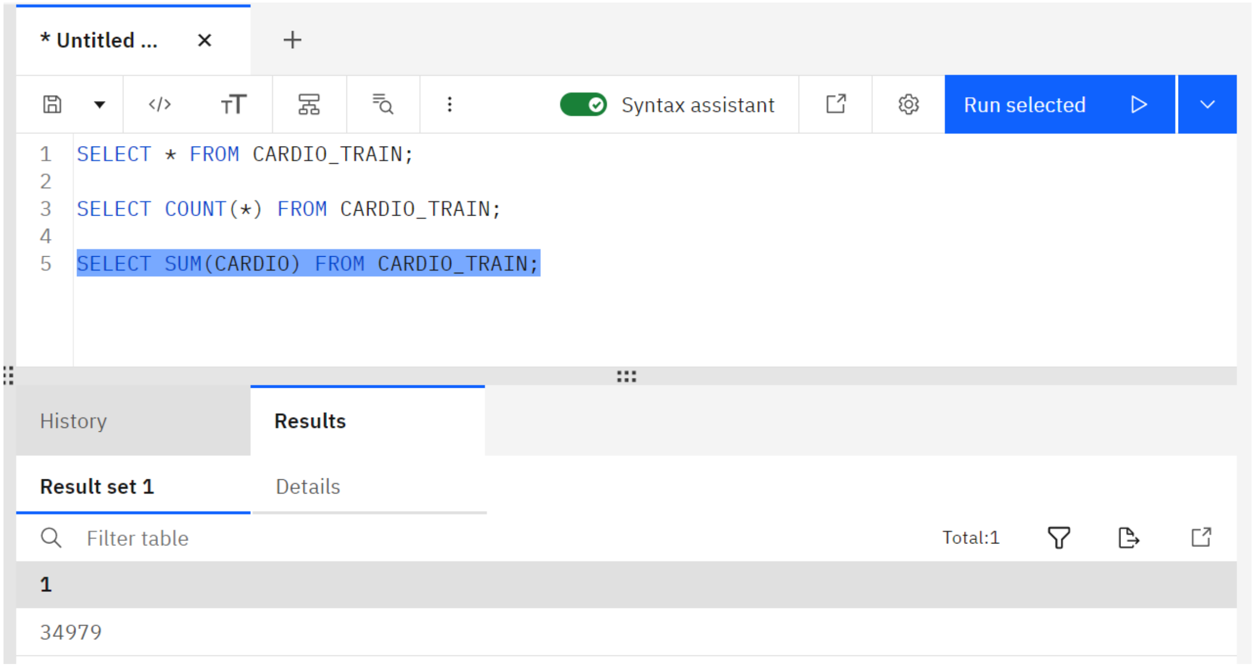
```
SELECT * FROM CARDIO_TRAIN;

SELECT COUNT(*) FROM CARDIO_TRAIN;
```

Truncated number of records:3668

ID	AGE	GENDER	HEIGHT	WEIGHT	AP_HI	AP_LO	CHOLESTEROL
0	18393	2	168	62.0	110	80	1
1	20228	1	156	85.0	140	90	3
2	18857	1	165	64.0	130	70	3

Items per page: 50 1–50 of 3668 items 1 1 of 74 pages

- Count number of disease cases:

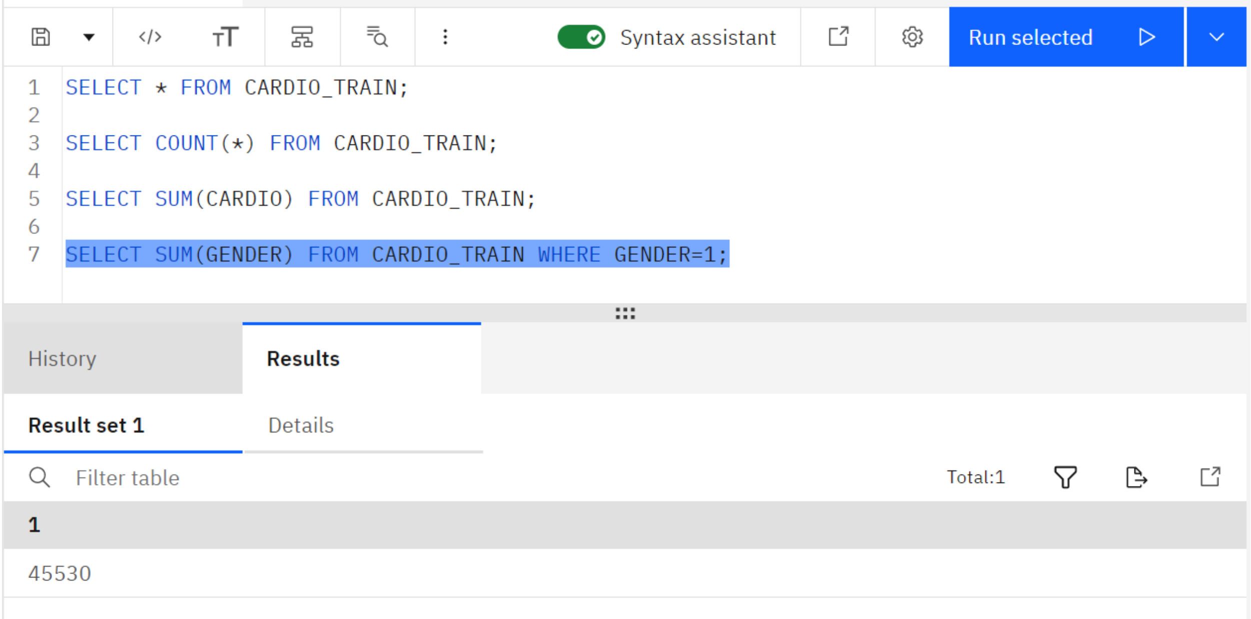


```
SELECT * FROM CARDIO_TRAIN;

SELECT COUNT(*) FROM CARDIO_TRAIN;

SELECT SUM(CARDIO) FROM CARDIO_TRAIN;
```

Total:1

1
34979

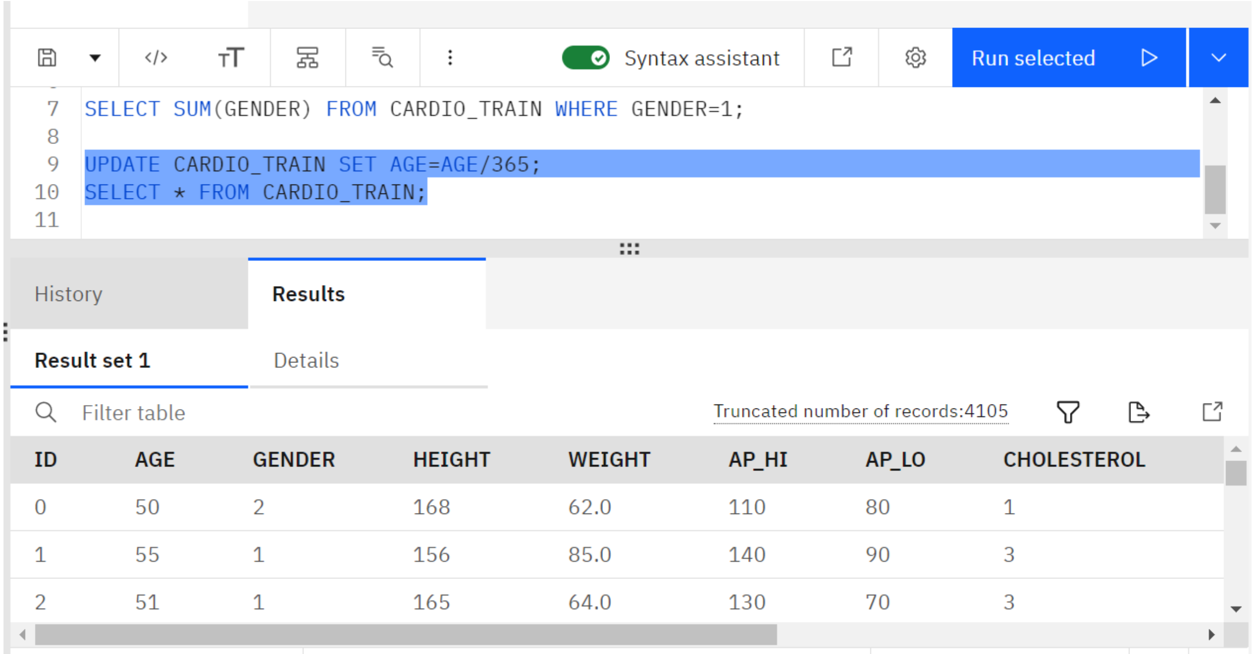
- Count gender in the data:



```
SELECT * FROM CARDIO_TRAIN;

SELECT COUNT(*) FROM CARDIO_TRAIN;

SELECT SUM(CARDIO) FROM CARDIO_TRAIN;

SELECT SUM(GENDER) FROM CARDIO_TRAIN WHERE GENDER=1;
```

1
45530

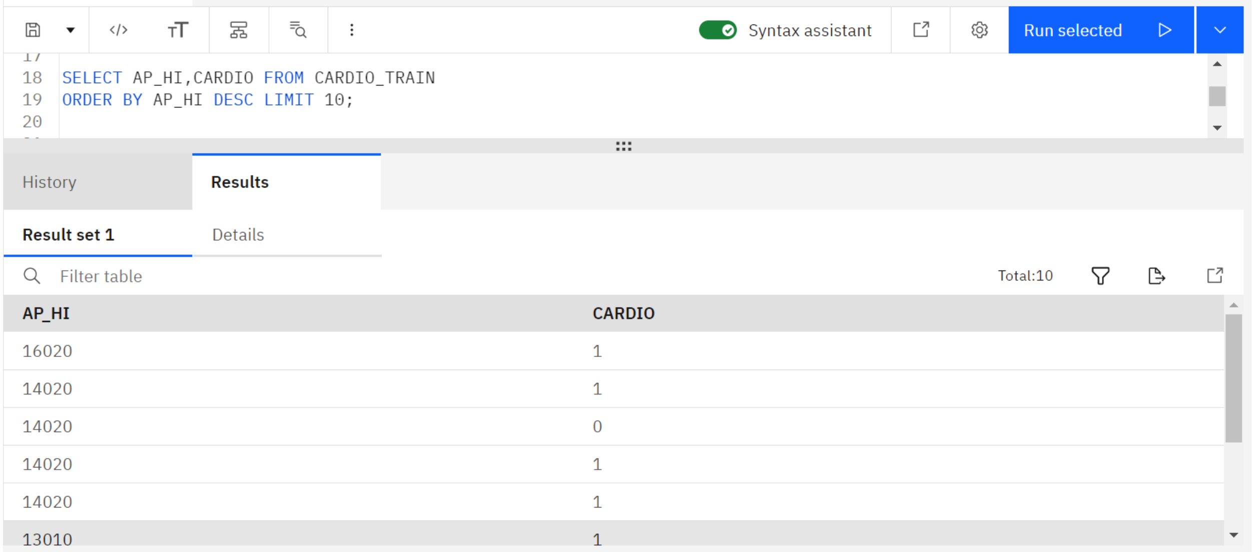
- Convert age to year:



```
SELECT SUM(GENDER) FROM CARDIO_TRAIN WHERE GENDER=1;

UPDATE CARDIO_TRAIN SET AGE=AGE/365;
SELECT * FROM CARDIO_TRAIN;

```

ID	AGE	GENDER	HEIGHT	WEIGHT	AP_HI	AP_LO	CHOLESTEROL
0	50	2	168	62.0	110	80	1
1	55	1	156	85.0	140	90	3
2	51	1	165	64.0	130	70	3

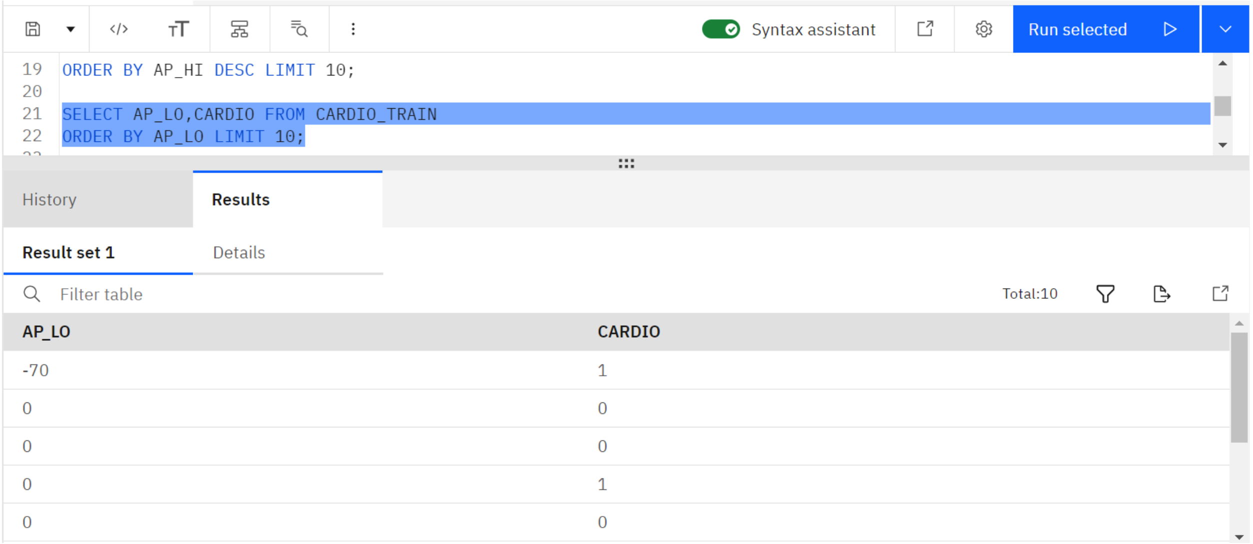
- Sort and check 10 people who has the highest AP_HI



```
SELECT AP_HI,CARDIO FROM CARDIO_TRAIN
ORDER BY AP_HI DESC LIMIT 10;
```

AP_HI	CARDIO
16020	1
14020	1
14020	0
14020	1
14020	1
13010	1

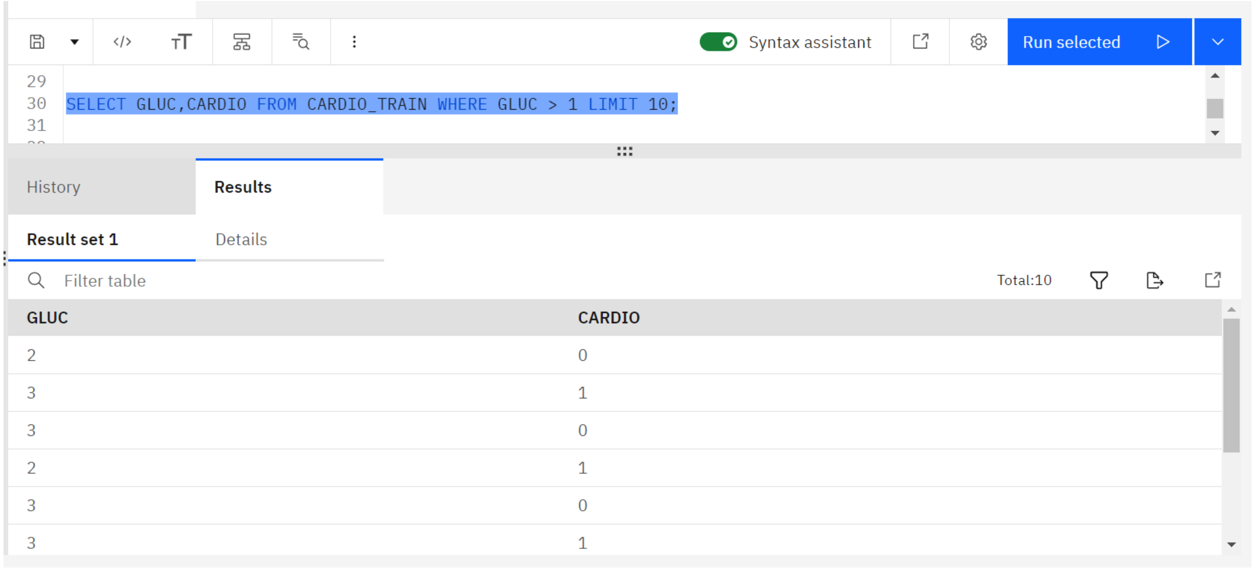
- Sort and take 10 people with the lowest AP_LO



```
ORDER BY AP_HI DESC LIMIT 10;

SELECT AP_LO,CARDIO FROM CARDIO_TRAIN
ORDER BY AP_LO LIMIT 10;
```

AP_LO	CARDIO
-70	1
0	0
0	0
0	1
0	0

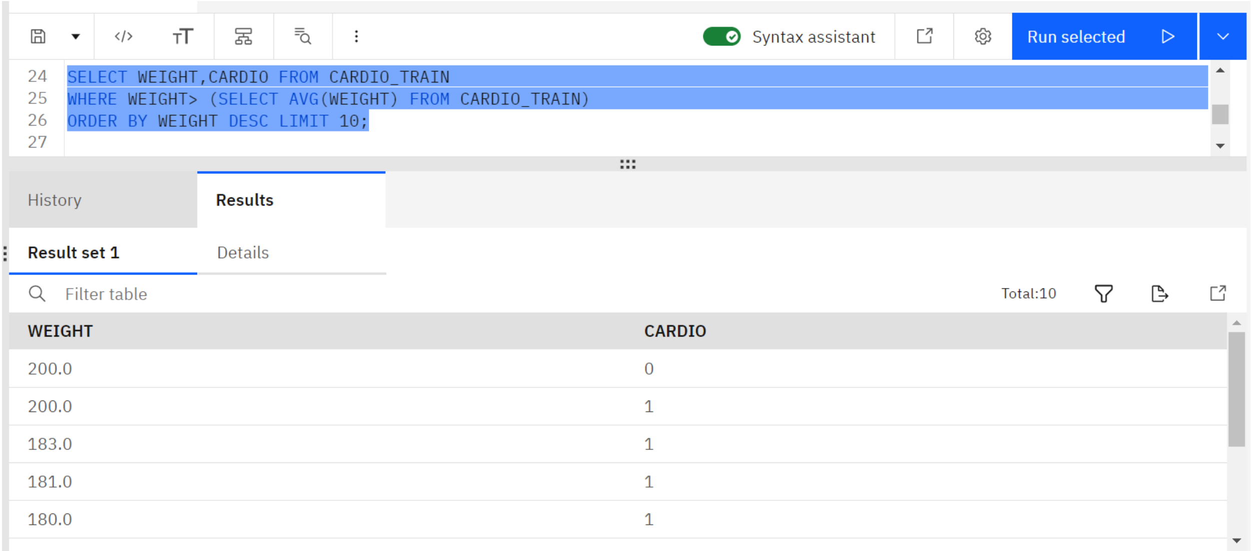
- Check 10 people with high glucose:



```
SELECT GLUC,CARDIO FROM CARDIO_TRAIN WHERE GLUC > 1 LIMIT 10;
```

GLUC	CARDIO
2	0
3	1
3	0
2	1
3	0
3	1

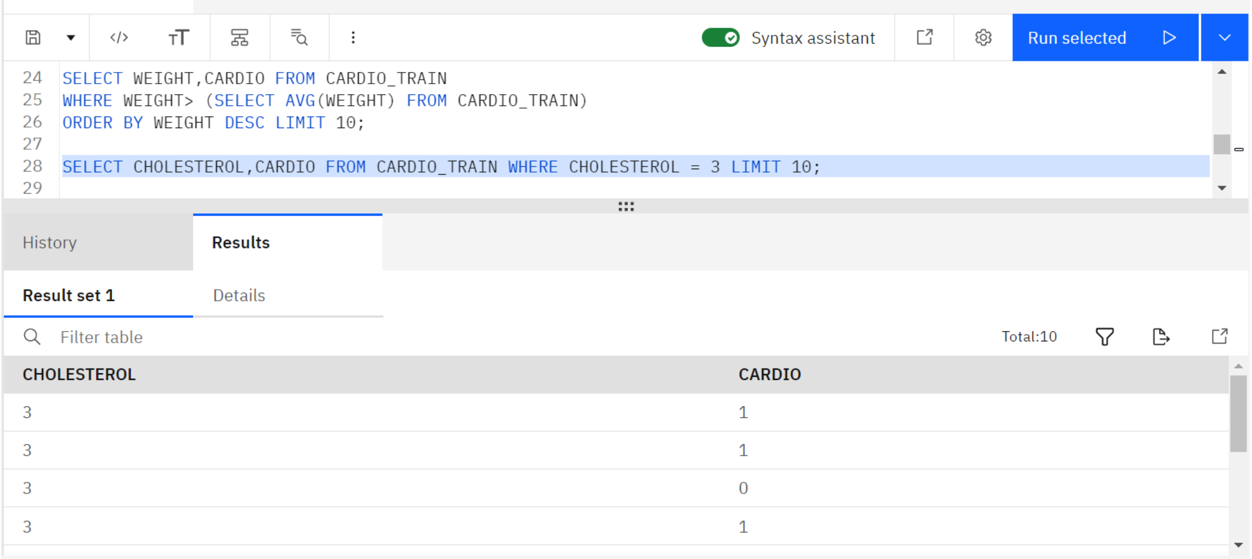
- Check 10 people whose weight higher normal:



```
SELECT WEIGHT,CARDIO FROM CARDIO_TRAIN
WHERE WEIGHT> (SELECT AVG(WEIGHT) FROM CARDIO_TRAIN)
ORDER BY WEIGHT DESC LIMIT 10;
```

WEIGHT	CARDIO
200.0	0
200.0	1
183.0	1
181.0	1
180.0	1

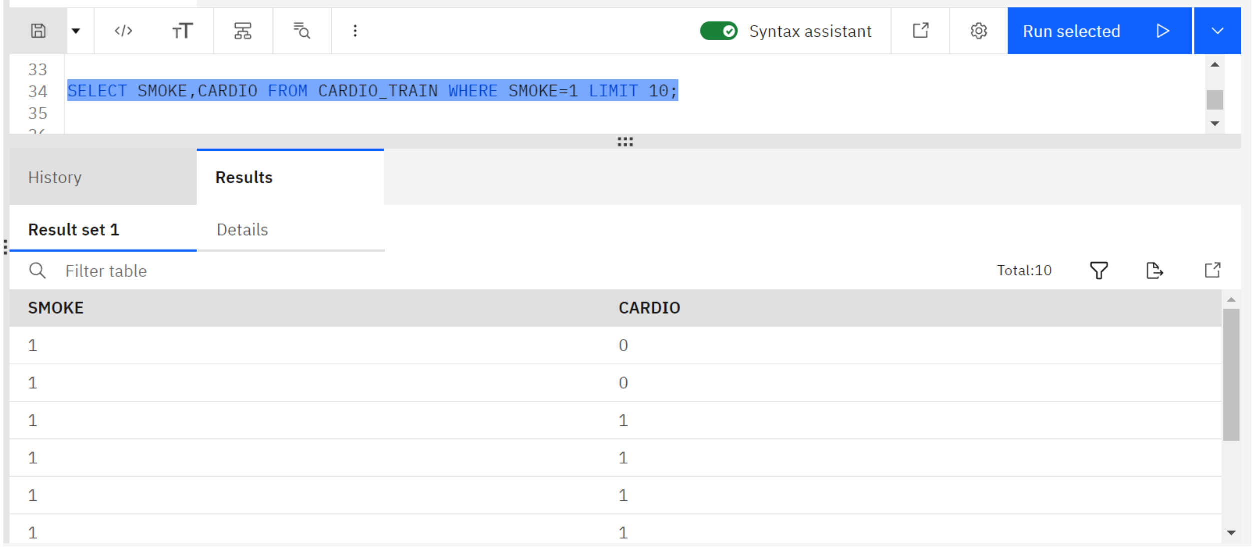
- Check 10 people whose CHOLESTEROL = 3:



```sql
SELECT WEIGHT,CARDIO FROM CARDIO_TRAIN
WHERE WEIGHT> (SELECT AVG(WEIGHT) FROM CARDIO_TRAIN)
ORDER BY WEIGHT DESC LIMIT 10;

SELECT CHOLESTEROL,CARDIO FROM CARDIO_TRAIN WHERE CHOLESTEROL = 3 LIMIT 10;
```

CHOLESTEROL	CARDIO
3	1
3	1
3	0
3	1

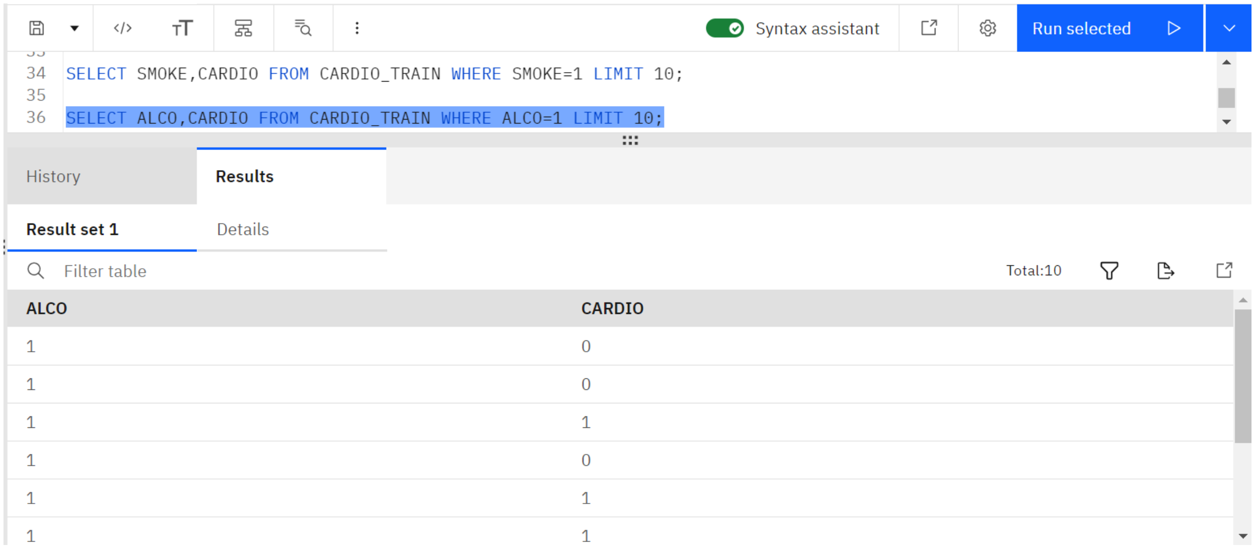
- Get 10 people who smoke frequently



```sql
SELECT SMOKE,CARDIO FROM CARDIO_TRAIN WHERE SMOKE=1 LIMIT 10;
```

SMOKE	CARDIO
1	0
1	0
1	1
1	1
1	1
1	1

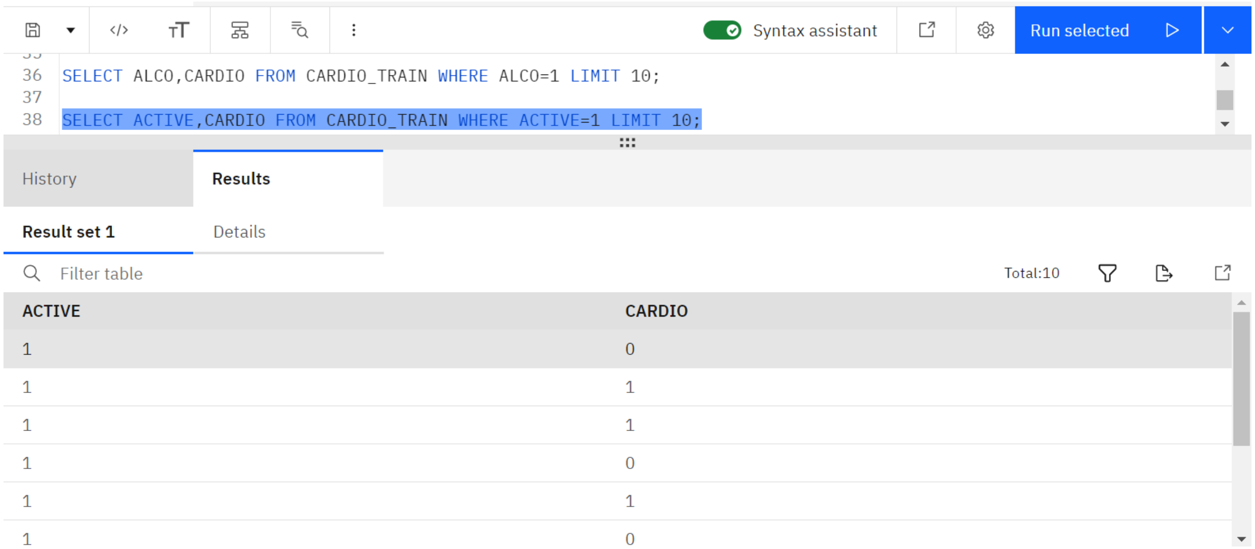
- Check 10 people who drink alcohol



```
SELECT SMOKE,CARDIO FROM CARDIO_TRAIN WHERE SMOKE=1 LIMIT 10;

SELECT ALCO,CARDIO FROM CARDIO_TRAIN WHERE ALCO=1 LIMIT 10;
```

ALCO	CARDIO
1	0
1	0
1	1
1	0
1	1
1	1

- Check 10 people who do exercise



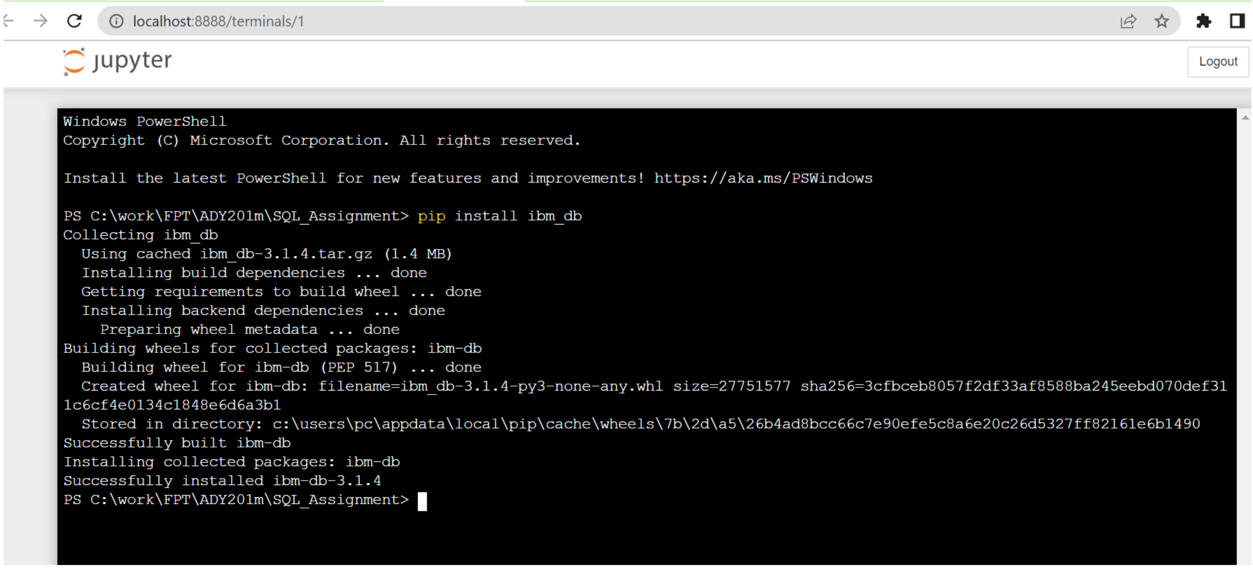
```
SELECT ALCO,CARDIO FROM CARDIO_TRAIN WHERE ALCO=1 LIMIT 10;

SELECT ACTIVE,CARDIO FROM CARDIO_TRAIN WHERE ACTIVE=1 LIMIT 10;
```

ACTIVE	CARDIO
1	0
1	1
1	1
1	0
1	1
1	0

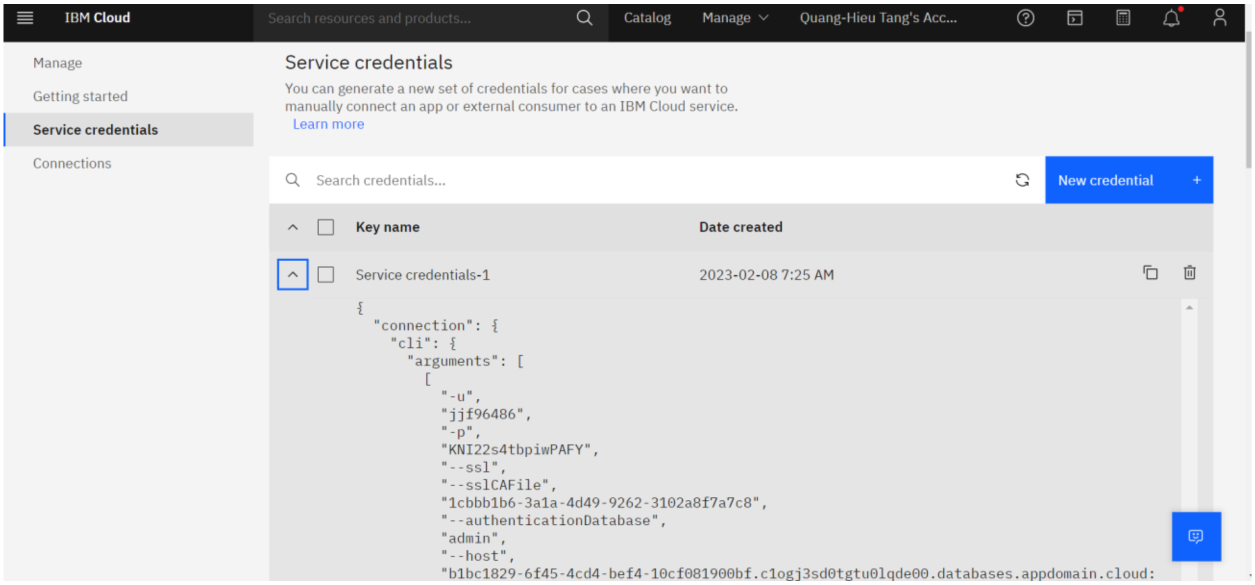
Connect to database:

In this task, you should connect with ibm db2 database using ibm_db API of python. Follow the instructions to install ibm_db package. Next, you should generate a credential of the database on cloud. Please review course material to know how to do it.

Note: In this lab, we use jupyter notebook to complete the rest of the assignment. You can easily get it by pip.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\work\FPT\ADY201m\SQL_Assignment> pip install ibm_db
Collecting ibm_db
  Using cached ibm_db-3.1.4.tar.gz (1.4 MB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
    Preparing wheel metadata ... done
Building wheels for collected packages: ibm-db
  Building wheel for ibm-db (PEP 517) ... done
  Created wheel for ibm-db: filename=ibm_db-3.1.4-py3-none-any.whl size=27751577 sha256=3cfbceb8057f2df33af8588ba245eebd070def31
1c6cf4e0134c1848e6d6a3b1
  Stored in directory: c:\users\pc\appdata\local\pip\cache\wheels\7b\2d\a5\26b4ad8bcc66c7e90efe5c8a6e20c26d5327ff82161e6b1490
Successfully built ibm-db
Installing collected packages: ibm-db
Successfully installed ibm-db-3.1.4
PS C:\work\FPT\ADY201m\SQL_Assignment>
```



Service credentials

You can generate a new set of credentials for cases where you want to manually connect an app or external consumer to an IBM Cloud service.

□	Key name	Date created
□	Service credentials-1	2023-02-08 7:25 AM

```
{
  "connection": {
    "cli": {
      "arguments": [
        [
          "-u",
          "jjf96486",
          "-p",
          "KNI22s4tbpiwPAFY",
          "--ssl",
          "--sslCAFile",
          "1cbbb1b6-3a1a-4d49-9262-3102a8f7a7c8",
          "--authenticationDatabase",
          "admin",
          "--host",
          "b1bc1829-6f45-4cd4-bef4-10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:
```

I. Connect to Db2 database on Cloud using Python

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Import the ibm_db Python library
- Enter the database connection credentials
- Create the database connection
- Do sql query data using cursor and display
- Retrieve data for further offline analysis
- Close the database connection

Import the ibm_db Python library The ibm_db API provides a variety of useful Python functions for accessing and manipulating data in an IBM® data server database, including functions for connecting to a database, preparing and issuing SQL statements, fetching rows from result sets, calling stored procedures, committing and rolling back transactions, handling errors, and retrieving metadata.

We first import the ibm_db library into our Python Application

Execute the following cells by clicking within it and then press Shift and Enter keys simultaneously

If you can not run this notebook commands, you may need to install these libraries by by using this command: pip install ibm_db in the terminal

In [20]:

```
import ibm_db
```

Create the DB2 database connection

Ibm_db API uses the IBM Data Server Driver for ODBC and CLI APIs to connect to IBM DB2 and Informix.

Lets build the dsn connection string using the credentials you entered above

In [21]:

```
#Replace the placeholder values with your actual Db2 hostname, username, and password:
dsn_hostname = "b1bc1829-6f45-4cd4-bef4-10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdor
dsn_uid = "jjf96486"        # e.g. "abc12345"
dsn_pwd = "KNI22s4tbpiwPAFY"      # e.g. "7dBZ3wWt9XN6$o0J"

dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "bludb"            # e.g. "BLUDB"
dsn_port = "32304"                # e.g. "32733"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
dsn_security = "SSL"              #i.e. "SSL"
```

In [22]:

```
#DO NOT MODIFY THIS CELL. Just RUN it with Shift + Enter
#Create the dsn connection string
dsn = (
    "DRIVER={0};"
    "DATABASE={1};"
    "HOSTNAME={2};"
    "PORT={3};"
    "PROTOCOL={4};"
    "UID={5};"
    "PWD={6};"
    "SECURITY={7};").format(dsn_driver, dsn_database, dsn_hostname, dsn_port, dsn_protocol

#print the connection string to check correct values are specified
print(dsn)
```

```
DRIVER={IBM DB2 ODBC DRIVER};DATABASE=bludb;HOSTNAME=b1bc1829-6f45-4cd4-bef4-10cf081900bf.
c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32304;PROTOCOL=TCPIP;UID=jjf96486;PWD=
KNI22s4tbpiwPAFY;SECURITY=SSL;
```

Now establish the connection to the database

In [23]:

```
#DO NOT MODIFY THIS CELL. Just RUN it with Shift + Enter
#Create database connection

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected to database: ", dsn_database, "as user: ", dsn_uid, "on host: ", dsn

except:
    print ("Unable to connect: ", ibm_db.conn_errormsg() )
```

```
Connected to database:  bludb as user:  jjf96486 on host:  b1bc1829-6f45-4cd4-bef4-10cf081
900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud
```

In [24]:

```
#Retrieve Metadata for the Database Server
server = ibm_db.server_info(conn)

print ("DBMS_NAME: ", server.DBMS_NAME)
print ("DBMS_VER:  ", server.DBMS_VER)
print ("DB_NAME:   ", server.DB_NAME)
```

```
DBMS_NAME:  DB2/LINUXX8664
DBMS_VER:   11.05.0800
DB_NAME:    BLUDB
```

In [25]:

```
#Retrieve Metadata for the Database Client / Driver
client = ibm_db.client_info(conn)

print ("DRIVER_NAME:          ", client.DRIVER_NAME)
print ("DRIVER_VER:           ", client.DRIVER_VER)
print ("DATA_SOURCE_NAME:     ", client.DATA_SOURCE_NAME)
print ("DRIVER_ODBC_VER:      ", client.DRIVER_ODBC_VER)
print ("ODBC_VER:             ", client.ODBC_VER)
print ("ODBC_SQL_CONFORMANCE: ", client.ODBC_SQL_CONFORMANCE)
print ("APPL_CODEPAGE:        ", client.APPL_CODEPAGE)
print ("CONN_CODEPAGE:        ", client.CONN_CODEPAGE)
```

```
DRIVER_NAME:           DB2CLI.DLL
DRIVER_VER:            11.05.0800
DATA_SOURCE_NAME:      BLUDB
DRIVER_ODBC_VER:       03.51
ODBC_VER:              03.01.0000
ODBC_SQL_CONFORMANCE:  EXTENDED
APPL_CODEPAGE:         1252
CONN_CODEPAGE:         1208
```

II. Access DB2 on Cloud using Python

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Retrieve data from CARDIO_TRAIN table
- Do following queries using API:
  1. Count number of rows of the table
  2. Count the number of cases which have Cardiovascular disease
  3. Find how many women and men in the experiment
  4. Sort data descendingly using AP_HI column, limiting to 10 results. Count how many percent of the illness in these 10 samples.
  5. Retrieve data of patient who has weight more than average weight from the table, limiting the result to 10 rows. How many percent of chance do they got the vascular problem.
  6. Obtain the ones who consume glucose far more than normal(GLUC > 1), limiting the results to 10. How many people got the illness in the result.
  7. Query to get 10 oldest people in the table. How many of them got vascular problem.
  8. Write code to get 10 people whose CHOLESTEROL=3. How many people got heart problem.
  9. Get 10 people who smoke from the table. How many of they who got heart disease.
  10. Get 10 people who practice exercise in the table. How many people got the disease.
- Download the table to csv file to use for offline analysis and then close the connection to save resource.

Task 1: Count number of rows of the table

In this task, we will use COUNT function to retrieve the number of rows in the table

In [30]:

```
#1. Retrieve how many rows from the table
query = "SELECT COUNT(ID) FROM CARDIO_TRAIN"

#Now execute the drop statment
results = ibm_db.exec_immediate(conn, query)
#Fetch the Dictionary (for the first row only) - replace ... with your code
ibm_db.fetch_both(results)
```

Out[30]:

```
{'1': 70000, 0: 70000}
```

Task 2: Count the number of cases which have Cardiovascular disease

In this task, will will count how many people in the experiment got the illness from 70K cases.

In [32]:

```
#1. Query statement
query = "SELECT COUNT(CARDIO) FROM CARDIO_TRAIN WHERE CARDIO=1"

#Now execute the drop statment
results = ibm_db.exec_immediate(conn, query)
#Fetch the Dictionary (for the first row only) - replace ... with your code
ibm_db.fetch_both(results)
```

Out[32]:

```
{'1': 34979, 0: 34979}
```

NOTE: There are 34979 cases in the dataset which have cardio-vascular problem.

Task 3: How many women and men in the experiment

We count Gender = 1 and Gender = 2

In [35]:

```
#1. Query statement
query = "SELECT COUNT(GENDER) FROM CARDIO_TRAIN WHERE GENDER=1"

#Now execute the drop statment
results = ibm_db.exec_immediate(conn, query)
#Fetch the Dictionary (for the first row only) - replace ... with your code
ibm_db.fetch_both(results)
```

Out[35]:

```
{'1': 45530, 0: 45530}
```

Task 4: Sort data descendingly using AP_HI column, limiting to 10 results. Count how many percent of the illness people in these 10 samples.

Retrieve 10 results from the table and sort descendingly by AP_HI, then count how many cases where CARDIO=1

In [39]:

```
#1. Query statement
query = "SELECT AP_HI,CARDIO FROM CARDIO_TRAIN ORDER BY AP_HI DESC LIMIT 10"

#2. Execute statement
results = ibm_db.exec_immediate(conn, query)
#Fetch the rest of the rows
count = 0
while ibm_db.fetch_row(results) != False:
    ap_hi = float(ibm_db.result(results, 0))
    cardio = int(ibm_db.result(results, "CARDIO"))
    if cardio==1: count+=1
    print(" AP_HI:",  ap_hi, "CARDIO:", cardio)
print("There are {}/10 cases got illness.".format(count))
```

```
 AP_HI: 16020.0 CARDIO: 1
 AP_HI: 14020.0 CARDIO: 1
 AP_HI: 14020.0 CARDIO: 0
 AP_HI: 14020.0 CARDIO: 1
 AP_HI: 14020.0 CARDIO: 1
 AP_HI: 13010.0 CARDIO: 1
 AP_HI: 13010.0 CARDIO: 0
 AP_HI: 11500.0 CARDIO: 1
 AP_HI: 11020.0 CARDIO: 1
 AP_HI: 2000.0 CARDIO: 1
There are 8/10 cases got illness.
```

NOTE: Normal blood pressure is lower than 200, but there are some values much higher than that. Hence, there might be some outliers in the dataset. Anyway, high presure causes hearty problems.

Task 5: Retrieve data of patient who has weight more than average weight from the table, limiting the result to 10 rows. How many percent of chance do they got the vascular problem.

Calculate average of weight, then select 10 people who has weight more than that. Check how many of them got problem.

In [43]:

```
#1. Query statement
query = "SELECT WEIGHT,CARDIO FROM CARDIO_TRAIN \
WHERE WEIGHT>(SELECT AVG(WEIGHT) \
FROM CARDIO_TRAIN) \
ORDER BY WEIGHT DESC LIMIT 10"

#2. Execute statement
results = ibm_db.exec_immediate(conn, query)
#Fetch the rest of the rows
count = 0
while ibm_db.fetch_row(results) != False:
    weight = float(ibm_db.result(results, 0))
    cardio = int(ibm_db.result(results, "CARDIO"))
    if cardio==1: count+=1
    print(" WEIGHT:",  weight, "CARDIO:", cardio)
print("There are {}/10 cases got illness.".format(count))
```

```
 WEIGHT: 200.0 CARDIO: 0
 WEIGHT: 200.0 CARDIO: 1
 WEIGHT: 183.0 CARDIO: 1
 WEIGHT: 181.0 CARDIO: 1
 WEIGHT: 180.0 CARDIO: 1
 WEIGHT: 180.0 CARDIO: 1
 WEIGHT: 180.0 CARDIO: 1
 WEIGHT: 180.0 CARDIO: 1
 WEIGHT: 178.0 CARDIO: 0
 WEIGHT: 178.0 CARDIO: 1
There are 8/10 cases got illness.
```

Task 6: Obtain the ones who consume glucose far more than normal(GLUC > 1), limiting the results to 10. How many people got the illness in the result.

In [47]:

```
#1. Query statement
query = "SELECT GLUC,CARDIO FROM CARDIO_TRAIN WHERE GLUC > 1 LIMIT 10"

#2. Execute statement
results = ibm_db.exec_immediate(conn, query)
#Fetch the rest of the rows
count = 0
while ibm_db.fetch_row(results) != False:
    gluc = float(ibm_db.result(results, 0))
    cardio = int(ibm_db.result(results, "CARDIO"))
    if cardio==1: count+=1
    print(" GLUC:",  gluc, "CARDIO:", cardio)
print("There are {}/10 cases got illness.".format(count))
```

```
 GLUC: 2.0 CARDIO: 0
 GLUC: 3.0 CARDIO: 1
 GLUC: 3.0 CARDIO: 0
 GLUC: 2.0 CARDIO: 1
 GLUC: 3.0 CARDIO: 0
 GLUC: 3.0 CARDIO: 1
 GLUC: 2.0 CARDIO: 0
 GLUC: 2.0 CARDIO: 1
 GLUC: 2.0 CARDIO: 0
 GLUC: 2.0 CARDIO: 1
There are 5/10 cases got illness.
```

Task 7: Query to get 10 oldest people in the table. How many of them got vascular problem.

In [48]:

```
#1. Query statement
query = "SELECT AGE,CARDIO FROM CARDIO_TRAIN ORDER BY AGE DESC LIMIT 10"

#2. Execute statement
results = ibm_db.exec_immediate(conn, query)
#Fetch the rest of the rows
count = 0
while ibm_db.fetch_row(results) != False:
    age = float(ibm_db.result(results, 0))
    cardio = int(ibm_db.result(results, "CARDIO"))
    if cardio==1: count+=1
    print(" AGE:",  age, "CARDIO:", cardio)
print("There are {}/10 cases got illness.".format(count))
```

```
 AGE: 64.0 CARDIO: 0
 AGE: 64.0 CARDIO: 1
 AGE: 64.0 CARDIO: 1
 AGE: 64.0 CARDIO: 0
 AGE: 64.0 CARDIO: 1
 AGE: 64.0 CARDIO: 1
 AGE: 64.0 CARDIO: 1
 AGE: 64.0 CARDIO: 0
 AGE: 64.0 CARDIO: 1
 AGE: 64.0 CARDIO: 1
There are 7/10 cases got illness.
```

Task 8: Write code to get 10 people whose CHOLESTEROL=3. How many people got heart problem.

In [49]:

```
#1. Query statement
query = "SELECT CHOLESTEROL,CARDIO FROM CARDIO_TRAIN WHERE CHOLESTEROL=3 LIMIT 10"

#2. Execute statement
results = ibm_db.exec_immediate(conn, query)
#Fetch the rest of the rows
count = 0
while ibm_db.fetch_row(results) != False:
    cocholesterol = float(ibm_db.result(results, 0))
    cardio = int(ibm_db.result(results, "CARDIO"))
    if cardio==1: count+=1
    print(" CHOLESTEROL:",  cocholesterol, "CARDIO:", cardio)
print("There are {}/10 cases got illness.".format(count))
```

```
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 0
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 1
 CHOLESTEROL: 3.0 CARDIO: 1
There are 9/10 cases got illness.
```

Task 9: Get 10 people who smoke from the table. How many of they who got heart disease.

In [50]:

```
#1. Query statement
query = "SELECT SMOKE,CARDIO FROM CARDIO_TRAIN WHERE SMOKE=1 LIMIT 10"

#2. Execute statement
results = ibm_db.exec_immediate(conn, query)
#Fetch the rest of the rows
count = 0
while ibm_db.fetch_row(results) != False:
    smoke = float(ibm_db.result(results, 0))
    cardio = int(ibm_db.result(results, "CARDIO"))
    if cardio==1: count+=1
    print(" SMOKE:",  smoke, "CARDIO:", cardio)
print("There are {}/10 cases got illness.".format(count))
```

```
 SMOKE: 1.0 CARDIO: 0
 SMOKE: 1.0 CARDIO: 0
 SMOKE: 1.0 CARDIO: 1
 SMOKE: 1.0 CARDIO: 1
 SMOKE: 1.0 CARDIO: 1
 SMOKE: 1.0 CARDIO: 1
 SMOKE: 1.0 CARDIO: 0
 SMOKE: 1.0 CARDIO: 1
 SMOKE: 1.0 CARDIO: 1
 SMOKE: 1.0 CARDIO: 1
There are 7/10 cases got illness.
```

Task 10: Get 10 people who practice exercise in the table. How many people got the disease.

In [51]:

```
#1. Query statement
query = "SELECT ACTIVE,CARDIO FROM CARDIO_TRAIN WHERE ACTIVE=1 LIMIT 10"

#2. Execute statement
results = ibm_db.exec_immediate(conn, query)
#Fetch the rest of the rows
count = 0
while ibm_db.fetch_row(results) != False:
    active = float(ibm_db.result(results, 0))
    cardio = int(ibm_db.result(results, "CARDIO"))
    if cardio==1: count+=1
    print(" ACTIVE:",  active, "CARDIO:", cardio)
print("There are {}/10 cases got illness.".format(count))
```

```
 ACTIVE: 1.0 CARDIO: 0
 ACTIVE: 1.0 CARDIO: 1
 ACTIVE: 1.0 CARDIO: 1
 ACTIVE: 1.0 CARDIO: 0
 ACTIVE: 1.0 CARDIO: 1
 ACTIVE: 1.0 CARDIO: 0
 ACTIVE: 1.0 CARDIO: 0
 ACTIVE: 1.0 CARDIO: 0
 ACTIVE: 1.0 CARDIO: 0
 ACTIVE: 1.0 CARDIO: 0
There are 3/10 cases got illness.
```

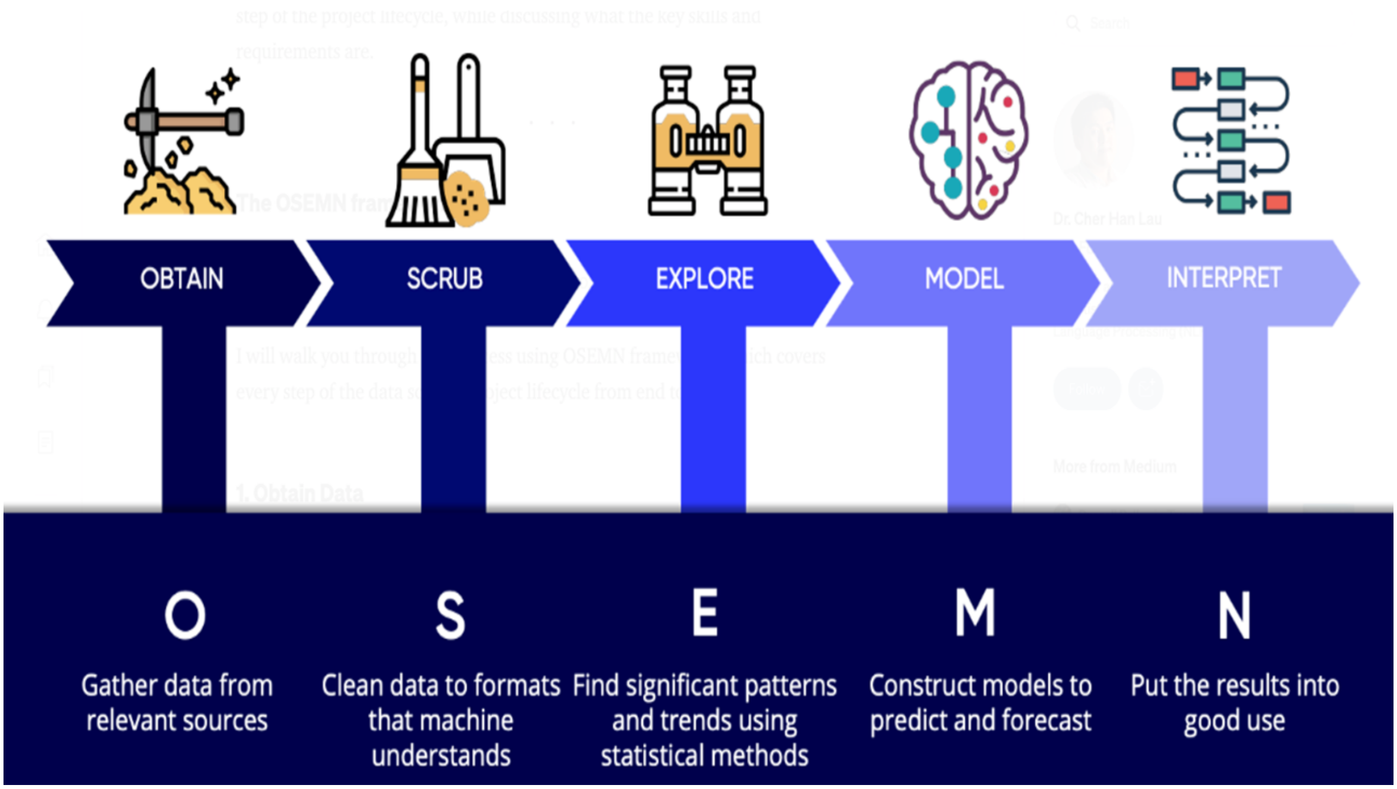
Save file to csv and close the connection

We free all resources by closing the connection. Remember that it is always important to close connections so that we can avoid unused connections taking up resources.

In [52]:
```
import pandas as pd
import ibm_db_dbi

#connection for pandas
pconn = ibm_db_dbi.Connection(conn)

#query statement to retrieve all rows in CARDIO_TRAIN table
selectQuery = "select * from CARDIO_TRAIN"

#retrieve the query results into a pandas dataframe
df = pd.read_sql(selectQuery, pconn)

df.to_csv("CARDIO_TRAIN.csv")

ibm_db.close(conn)
```

Out[52]:
```
True
```

III. Data Analysis with Python


OBTAIN
SCRUB
EXPLORE
MODEL
INTERPRET
O
Gather data from relevant sources
S
Clean data to formats that machine understands
E
Find significant patterns and trends using statistical methods
M
Construct models to predict and forecast
N
Put the results into good use

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Read data from csv files and perform data wrangling
- Visualize data to get insigth of problem
- Clean and standardize data
- Using statistic model to verify hypotheses
- Create model to classify the problem
- Fine-tuning model to have better result.

Read data from csv files and perform data wrangling

In [123…

```
df = pd.read_csv("CARDIO_TRAIN.csv")
df.head(10)
```

Out[123…

	Unnamed: 0	ID	AGE	GENDER	HEIGHT	WEIGHT	AP_HI	AP_LO	CHOLESTEROL	GLUC	SMOKE	ALCO	ACTIVE
0	0	0	50	2	168	62.0	110	80	1	1	0	0	1
1	1	1	55	1	156	85.0	140	90	3	1	0	0	1
2	2	2	51	1	165	64.0	130	70	3	1	0	0	0
3	3	3	48	2	169	82.0	150	100	1	1	0	0	1
4	4	4	47	1	156	56.0	100	60	1	1	0	0	0
5	5	8	60	1	151	67.0	120	80	2	2	0	0	0
6	6	9	60	1	157	93.0	130	80	3	1	0	0	1
7	7	12	61	2	178	95.0	130	90	3	3	0	0	1
8	8	13	48	1	158	71.0	110	70	1	1	0	0	1
9	9	14	54	1	164	68.0	110	60	1	1	0	0	0

Describe and do some statistic sumation about data

In [54]:

```
df.describe()
```

Out[54]:

	Unnamed: 0	ID	AGE	GENDER	HEIGHT	WEIGHT	AP_HI	AP_
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.0000
mean	34999.500000	49972.419900	52.840671	1.349571	164.359229	74.205686	128.817286	96.6304
std	20207.403759	28851.302323	6.766774	0.476838	8.210126	14.395761	154.011419	188.4725
min	0.000000	0.000000	29.000000	1.000000	55.000000	10.000000	-150.000000	-70.0000
25%	17499.750000	25006.750000	48.000000	1.000000	159.000000	65.000000	120.000000	80.0000
50%	34999.500000	50001.500000	53.000000	1.000000	165.000000	72.000000	120.000000	80.0000
75%	52499.250000	74889.250000	58.000000	2.000000	170.000000	82.000000	140.000000	90.0000
max	69999.000000	99999.000000	64.000000	2.000000	250.000000	200.000000	16020.000000	11000.0000

In [55]:

```
df.isnull()
```

Out[55]:

	Unnamed: 0	ID	AGE	GENDER	HEIGHT	WEIGHT	AP_HI	AP_LO	CHOLESTEROL	GLUC	SMOKE	ALCO	A
0	False	False	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	False	False	

	Unnamed: 0	ID	AGE	GENDER	HEIGHT	WEIGHT	AP_HI	AP_LO	CHOLESTEROL	GLUC	SMOKE	ALCO	A
...	...	...	...	...	...	...	...	...	...	...	...	...	
69995	False	False	False	False	False	False	False	False	False	False	False	False	
69996	False	False	False	False	False	False	False	False	False	False	False	False	
69997	False	False	False	False	False	False	False	False	False	False	False	False	
69998	False	False	False	False	False	False	False	False	False	False	False	False	
69999	False	False	False	False	False	False	False	False	False	False	False	False	

70000 rows × 14 columns

Note: This data has no null data, so we do not need to fill nan value.

Visualize data and calculate correlation

We need to have preliminery estimation about the dataset and visualization is a useful tool for this.

In [58]:

```
# Calculate correlation of CARDIO over others features.
corr = df.corr().CARDIO.sort_values(ascending=False)
corr
```

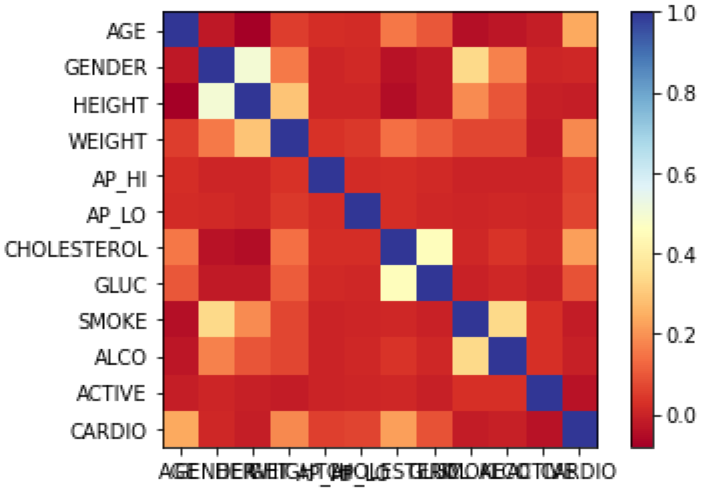
Out[58]:

```
CARDIO          1.000000
AGE             0.237985
CHOLESTEROL     0.221147
WEIGHT          0.181659
GLUC            0.089307
AP_LO           0.065719
AP_HI           0.054475
GENDER          0.008109
Unnamed: 0      0.003800
ID              0.003799
ALCO           -0.007330
HEIGHT         -0.010821
SMOKE          -0.015486
ACTIVE         -0.035653
Name: CARDIO, dtype: float64
```

In [62]:

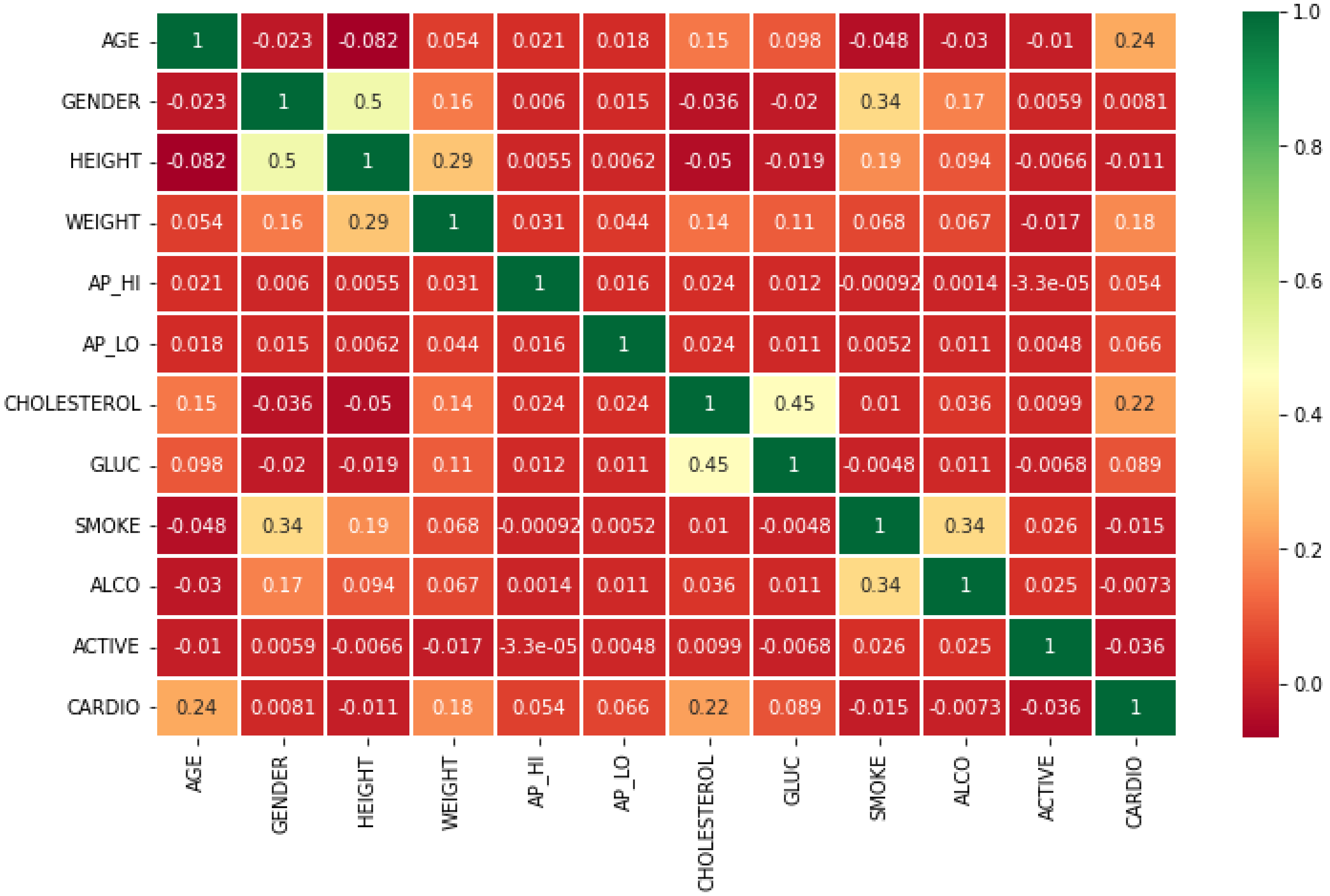
```
# Heat map for correlation using Matplotlib
import matplotlib.pyplot as plt
corr = df[['AGE','GENDER','HEIGHT','WEIGHT','AP_HI','AP_LO',
           'CHOLESTEROL','GLUC','SMOKE','ALCO','ACTIVE','CARDIO']].corr()
# Displaying dataframe as an heatmap
# with diverging colourmap as RdYlBu
plt.imshow(corr, cmap ="RdYlBu")

# Displaying a color bar to understand
# which color represents which range of data
plt.colorbar()

# Assigning labels of x-axis
# according to dataframe
plt.xticks(range(len(corr)), corr.columns)

# Assigning labels of y-axis
# according to dataframe
plt.yticks(range(len(corr)), corr.index)
```

```
# Displaying the figure
plt.show()
```



In [77]:

```
# Heat map for correlation using seaborn
import seaborn as sns
# Defining figure size
# for the output plot
fig, ax = plt.subplots(figsize = (12, 7))
# Displaying dataframe as an heatmap
# with diverging colourmap as RdYlGn
sns.heatmap(corr, cmap ='RdYlGn', linewidths = 0.30, annot = True)
plt.show()
```

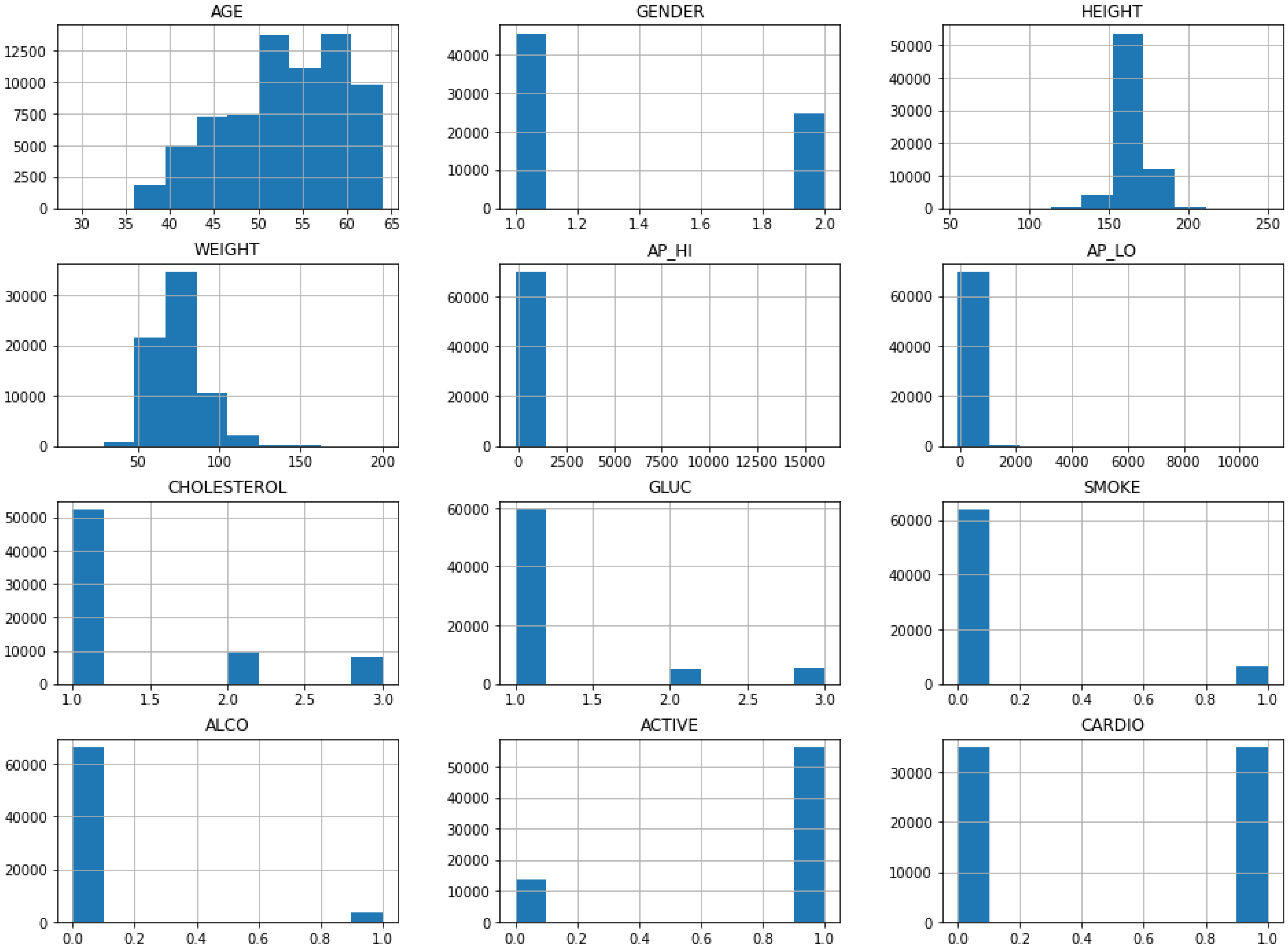


NOTE: As we can interpret from the figure, AGE, CHOLESTEROL, and WEIGHT are 3 factors which are the most likely cause the problem. But to concret our hypotheses, we need to verify them using some statistic algorithms.

Other Figures

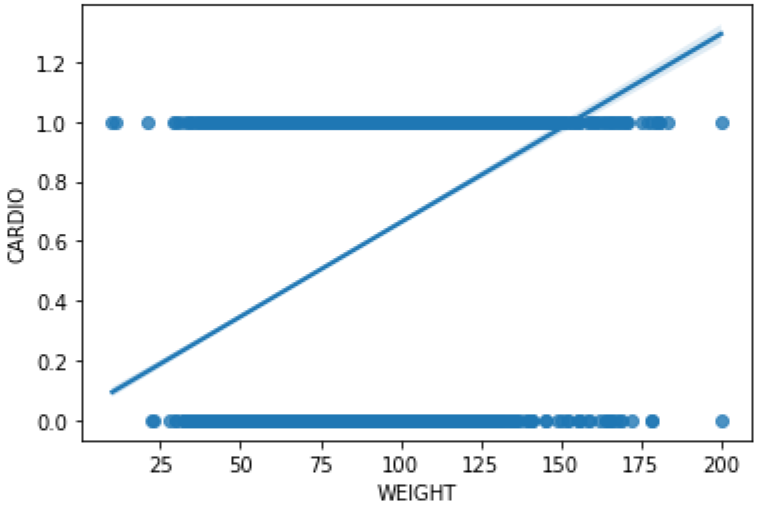
In [97]:

```
# Histogram of features
df[['AGE','GENDER','HEIGHT','WEIGHT','AP_HI','AP_LO',
           'CHOLESTEROL','GLUC','SMOKE','ALCO','ACTIVE','CARDIO']].hist(figsize=[16, 12])
plt.show()
```



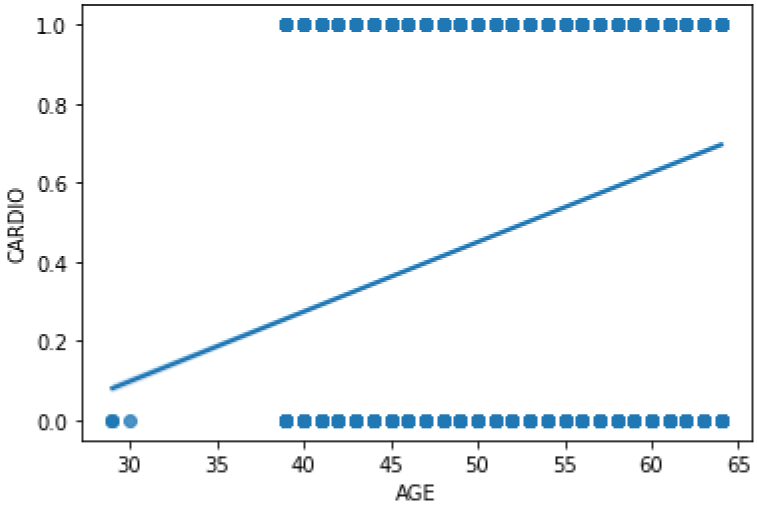
In [91]:

```
# draw regplot
sns.regplot(x = "WEIGHT",
            y = "CARDIO",
            data = df,
            dropna = True)
# show the plot
ax.set(xlabel='WEIGHT', ylabel='CARDIO')
plt.show()
```



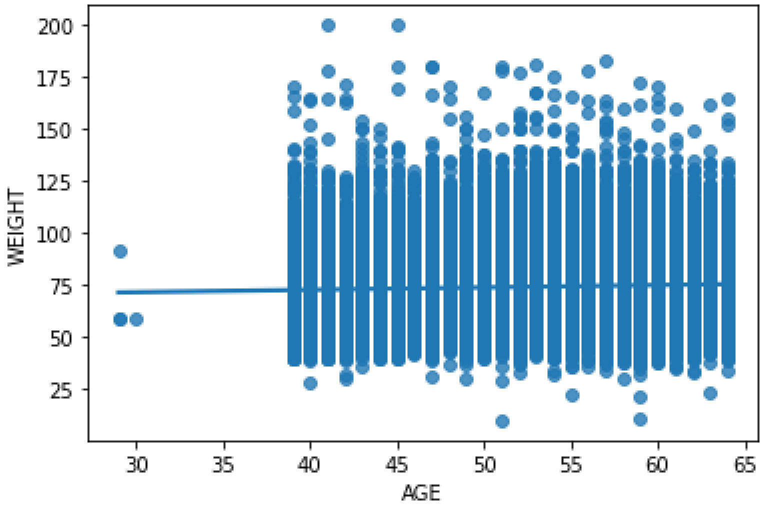
In [92]:

```
# draw regplot
sns.regplot(x = "AGE",
            y = "CARDIO",
            data = df,
            dropna = True)
# show the plot
ax.set(xlabel='AGE', ylabel='CARDIO')
plt.show()
```



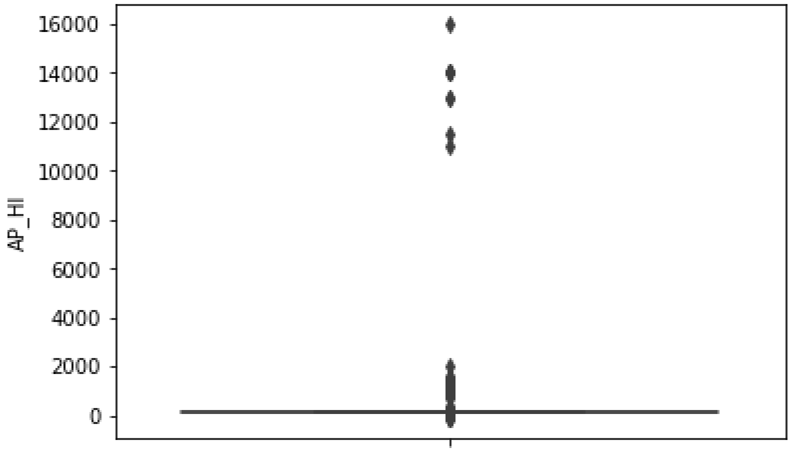
In [93]:

```
# draw regplot
sns.regplot(x = "AGE",
            y = "WEIGHT",
            data = df,
            dropna = True)
# show the plot
ax.set(xlabel='AGE', ylabel='WEIGHT')
plt.show()
```



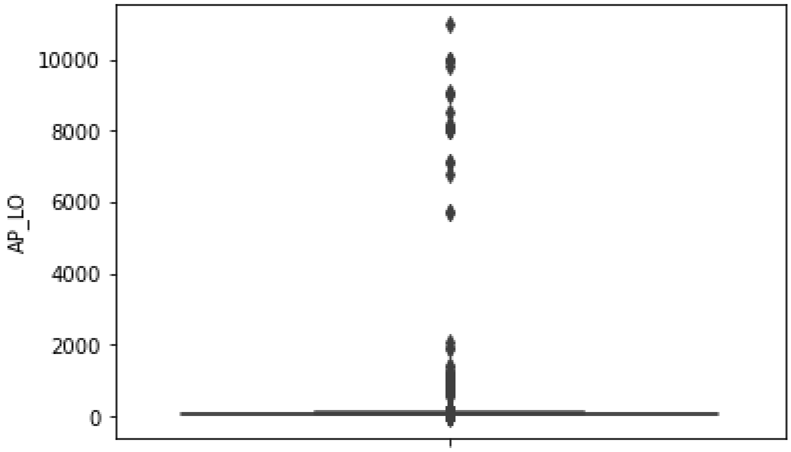
In [100…

```
# Box plot for AP_H
sns.boxplot(y= "AP_HI", data = df)
plt.show()
```



In [101…

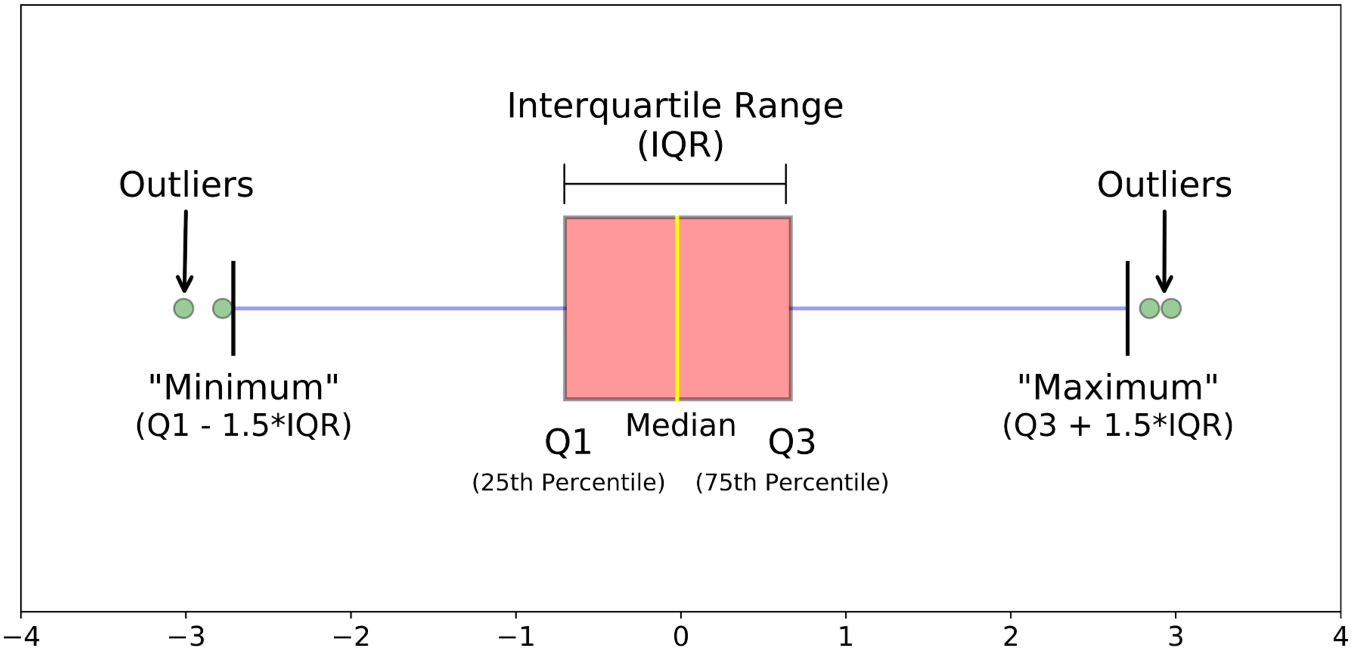
```
# Box plot for AP_LO
sns.boxplot(y= "AP_LO", data = df)
plt.show()
```



NOTE: There are many outliers in AP_HI and AP_LO, we need to modify these data for the last step to classify the disease

Clean and standardize data

As we can see, collected data contains abnormal values of AP_HI and AP_LO. Therefore, It will affect to the analyst result. Hence, we will replace outliers of AP_HI with maximum, outliers of AP_LO with minimum.



In [124…

```
# Process for AP_HI
first_quantile,third_quantile = df.AP_HI.quantile([0.25, 0.75])
iqr = third_quantile - first_quantile
maximum = third_quantile + 1.5*iqr
minimum = first_quantile - 1.5*iqr
print("maximum:",maximum,"minimum:",minimum)
df.loc[df.AP_HI>maximum,'AP_HI'] = maximum
df.loc[df.AP_HI<minimum,'AP_HI'] = minimum
```

```
maximum: 170.0 minimum: 90.0
```

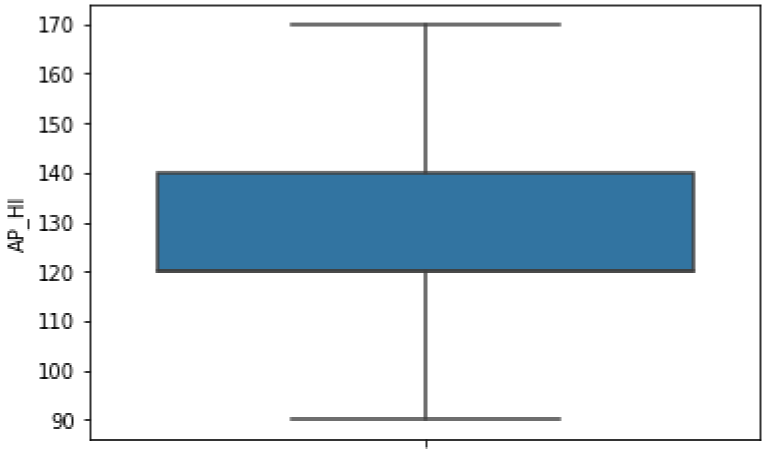
In [125…

```
# Process for AP_LO
first_quantile,third_quantile = df.AP_LO.quantile([0.25, 0.75])
iqr = third_quantile - first_quantile
maximum = third_quantile + 1.5*iqr
minimum = first_quantile - 1.5*iqr
print("maximum:",maximum,"minimum:",minimum)
df.loc[df.AP_LO>maximum,'AP_LO'] = maximum
df.loc[df.AP_LO<minimum,'AP_LO'] = minimum
```

```
maximum: 105.0 minimum: 65.0
```

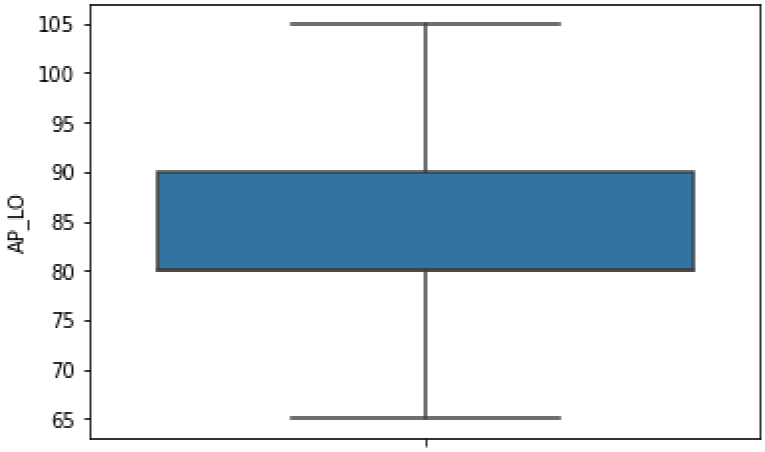
In [126…

```
# Redraw Box plot for AP_H
sns.boxplot(y= "AP_HI", data = df)
plt.show()
```



In [127…

```
# Redraw Box plot for AP_H
sns.boxplot(y= "AP_LO", data = df)
plt.show()
```



Statistic

We can dig further into the data using some statistic tools

CHOLESTEROL vs CARDIO

Null Hypothesis: CHOLESTEROL is not related to the disease.

- Since CHOLESTEROL is categorical variables with 3 options: 1, 2, and 3. We can use ANNOVA test to verify null hypothesis.

In [128…

```
# Importing library
from scipy.stats import f_oneway
# Conduct the one-way ANOVA
low_cholesterol=df[df['CHOLESTEROL']==1].CARDIO
high_cholesterol=df[df['CHOLESTEROL']==2].CARDIO
very_high_cholesterol = df[df['CHOLESTEROL']==3].CARDIO
f_statistic, p_value = f_oneway(low_cholesterol, high_cholesterol, very_high_cholesterol)
print("F_Statistic: {0}, P-Value: {1}".format(f_statistic,p_value))
```

```
F_Statistic: 1799.6607856699602, P-Value: 0.0
```

Conclusion: Since the p-value is less than 0.05, we will reject the null hypothesis as there is significant evidence that at least one of the means differ.

AGE vs CARDIO

Null Hypothesis: AGE is not related to the disease.

- We can split AGE by their mean and testing whether lower age and higher age have a same rate of heart disease or not.

In [132…

```
from scipy.stats import levene,ttest_ind
lower_age_group = df[df.AGE<df.AGE.mean()].CARDIO
higher_age_group = df[df.AGE>=df.AGE.mean()].CARDIO

# Using levene test to check whether 2 variance is equal
f_statistic, p_value =levene(lower_age_group,
                             higher_age_group, center='mean')
print("Levene Test: F_Statistic: {0}, P-Value: {1}".format(f_statistic,p_value))
# If p-value is greater than 0.05 we can assume equality of variance

f_statistic, p_value = ttest_ind(lower_age_group,
                             higher_age_group, equal_var = p_value>0.05)
print("T-Test: F_Statistic: {0}, P-Value: {1}".format(f_statistic,p_value))
```

```
Levene Test: F_Statistic: 133.98315419526062, P-Value: 5.880069082322 94e-31
T-Test: F_Statistic: -52.51350890203989, P-Value: 0.0
```

Conclusion: Since the TTEst p-value is less than 0.05, we will reject the null hypothesis as there is significant evidence that 2 means differ.

WEIGHT vs CARDIO

Null Hypothesis: WEIGHT is not related to the disease.

- Since weight is a continuous variable and cardio is also can be considered as continuous variable. We can use peason to test the hypothesis

In [135…

```
from scipy.stats import pearsonr
coeff, p_value = pearsonr(df.WEIGHT, df.CARDIO)
print("coeff:",coeff,"p_value:",p_value)
```

```
coeff: 0.18165940834451674 p_value: 0.0
```

Conclusion: Since the Peason p-value is less than 0.05, we will reject the null hypothesis as there exists a relationship between the twos.

ALCOHOL vs CARDIO

Null Hypothesis: ALCOHOL is not related to the disease.

- Since ALCOHOL is a binary variable and CARDIO is also a binary one. We can use chi square to test the hypothesis.

In [139…

```
from scipy.stats import chi2_contingency
# Create a cross tab table
cont_table  = pd.crosstab(df.ALCO, df.CARDIO)

# calculate chi square values
chi2_contingency(cont_table, correction = True)
```

Out[139…

```
(3.696547466479263,
 0.05452518218322108,
 1,
 array([[33137.8708, 33098.1292],
        [ 1883.1292,  1880.8708]]))
```

Conclusion: Since the p-value(0.054525) is greater than 0.05, we fail to reject the null hypothesis. As there is no sufficient evidence that alcoholic people might have greater chance to get illness.

OSL Model

There is another library called OSL which can show the relationship between features in statsmodels library

In [141…

```
from statsmodels.formula.api import ols
import statsmodels.api as sm
## X is the input variables (or independent variables)
X = df[['AGE','GENDER','HEIGHT','WEIGHT','AP_HI','AP_LO',
           'CHOLESTEROL','GLUC','SMOKE','ALCO','ACTIVE']]

## add an intercept (beta_0) to our model
X = sm.add_constant(X)

## y is the target/dependent variable
y = df['CARDIO']

model = sm.OLS(y, X).fit()
predictions = model.predict(X)

# Print out the statistics
model.summary()
```

```
C:\Users\PC\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: In
a future version of pandas all arguments of concat except for the argument 'objs' will be
keyword-only
  x = pd.concat(x[::order], 1)
```

Out[141…

OLS Regression Results

Dep. Variable:	CARDIO	**R-squared:**	0.235
Model:	OLS	**Adj. R-squared:**	0.235
Method:	Least Squares	**F-statistic:**	1954.
Date:	Sun, 21 May 2023	**Prob (F-statistic):**	0.00
Time:	15:57:05	**Log-Likelihood:**	-41431.
No. Observations:	70000	**AIC:**	8.289e+04
Df Residuals:	69988	**BIC:**	8.300e+04
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1.6708	0.042	-39.948	0.000	-1.753	-1.589
AGE	0.0101	0.000	40.131	0.000	0.010	0.011
GENDER	-0.0041	0.004	-0.979	0.328	-0.012	0.004
HEIGHT	-0.0008	0.000	-3.229	0.001	-0.001	-0.000
WEIGHT	0.0021	0.000	16.542	0.000	0.002	0.002
AP_HI	0.0096	0.000	64.319	0.000	0.009	0.010

AP_LO	0.0040	0.000	14.993	0.000	0.003	0.005
CHOLESTEROL	0.0941	0.003	33.686	0.000	0.089	0.100
GLUC	-0.0207	0.003	-6.381	0.000	-0.027	-0.014
SMOKE	-0.0257	0.007	-3.953	0.000	-0.039	-0.013
ALCO	-0.0379	0.008	-4.840	0.000	-0.053	-0.023
ACTIVE	-0.0437	0.004	-10.487	0.000	-0.052	-0.036

Omnibus:	26450.210	**Durbin-Watson:**	1.981
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3466.466
Skew:	0.049	**Prob(JB):**	0.00
Kurtosis:	1.914	**Cond. No.**	6.11e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.11e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Create model to classify the problem

In this section, we learn how to create a mulitple linear regression to classify the problem

Standardization data

To perform better, we need to standardization to scale down data range. In this case, we use StandardScaler to do this task.

In [154…

```
df = df[['AGE','GENDER','HEIGHT','WEIGHT','AP_HI','AP_LO',
            'CHOLESTEROL','GLUC','SMOKE','ALCO','ACTIVE','CARDIO']]
# Scale AGE, HEIGHT, WEIGHT, AP_HI, AP_LO using standard scaler
```

We transform numeric data and then concatenate with categorical data

In [163…

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data = scaler.fit_transform(df[['AGE', 'HEIGHT', 'WEIGHT', 'AP_HI', 'AP_LO']])
```

In [164…

```
import numpy as np
data = np.concatenate([data,df[['GENDER','CHOLESTEROL','GLUC','SMOKE','ALCO','ACTIVE','CAR
X = data[:,:11]
y = data[:,11]
#data.shape (70000, 12)
#print(X.shape,y.shape) (70000, 11) (70000,)
```

```
(70000, 11) (70000,)
```

Split data

Next, splitting data into 2 parts, 1 for training and 1 for testing . We take 30% for testing and 70% for training

In [165…

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Build Model

We will build a simple model to do this classification problem. Ridge model is a regulated model with modifiable alpha parameter to mitigate overfitting phenominal. We will use Ridge model for this illustration.

In [187…

```
from sklearn.linear_model import RidgeClassifier
ridge = RidgeClassifier(alpha = 0.0001)
ridge.fit(X_train,y_train)
```

Out[187…

```
RidgeClassifier(alpha=0.0001)
```

Evaluation Model

In this example, we will use Accuracy to be metrix for performance.

In [188…

```
from sklearn.metrics import accuracy_score
y_hat = ridge.predict(X_test)
accuracy_score(y_test, y_hat)
```

Out[188…

```
0.7301428571428571
```

NOTE: We can predict with accuracy as 73% which is relatively good. Try another model to see whether we can improve this or not

Finetunning model

There are vast of models in practice. For example, RandomForest, XGBoost, Gradient Boosting, Decision Tree ... We will test with some models to check the performance

RandomForest

In [190…

```
from sklearn.ensemble import RandomForestClassifier
rndF = RandomForestClassifier()
#Train
rndF.fit(X_train,y_train)
#Evaluation
y_hat = rndF.predict(X_test)
accuracy_score(y_test, y_hat)
```

Out[190…

```
0.7093333333333334
```

GradientBoostingClassifier

In [192…

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(X_train,y_train)
#Evaluation
```

```
y_hat = gbc.predict(X_test)
accuracy_score(y_test, y_hat)
```

Out[192…

```
0.7396190476190476
```

NOTE: This model with default parameters will give 74% accuracy, so we can use this model to improve our performance rather than RidgeClassifier

AdaBoostClassifier

In [200…

```
from sklearn.ensemble import AdaBoostClassifier

adac = AdaBoostClassifier(n_estimators=100, random_state=0)
adac.fit(X_train, y_train)
#Evaluation
y_hat = adac.predict(X_test)
accuracy_score(y_test, y_hat)
```

Out[200…

```
0.7341428571428571
```

Bagging Classifier

In [201…

```
from sklearn.ensemble import BaggingClassifier
baggc = BaggingClassifier(KNeighborsClassifier(), max_samples=0.5, max_features=0.5)
baggc.fit(X_train,y_train)
y_hat = baggc.predict(X_test)
accuracy_score(y_test, y_hat)
```

Out[201…

```
0.7293333333333333
```

Extra TreeClassifier

In [202…

```
from sklearn.ensemble import ExtraTreesClassifier
eTrc= ExtraTreesClassifier(n_estimators=100, random_state=0)
eTrc.fit(X_train,y_train)
y_hat = eTrc.predict(X_test)
accuracy_score(y_test, y_hat)
```

Out[202…

```
0.6938095238095238
```

Stack Classifier

In [206…

```
from sklearn.linear_model import RidgeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier
estimators = [('ridge', RidgeClassifier()),
             ('knr', KNeighborsClassifier(n_neighbors=20,
                      metric='euclidean'))]
final_estimator = GradientBoostingClassifier(
    n_estimators=25, subsample=0.5, min_samples_leaf=25, max_features=1,
     random_state=42)
stackC = StackingClassifier(
   estimators=estimators,
   final_estimator=final_estimator)
stackC.fit(X_train,y_train)
y_hat = stackC.predict(X_test)
accuracy_score(y_test, y_hat)
```

```
Out[206…  0.7344761904761905
```

Watson Studio and Github:

Lastly, you should do the same thing but on Watson studio. After that, you should upload your files to repository on Github. Please read course material to know how to do it.