BIG DATA PARALLEL PROGRAMMING

Project Report

Shankaranarayanan Bangalore Ramalingam

Table of Contents

Problem Statement:

The aim of this project is predict if a person will be getting an ischemic heart disease and stroke or not, and this is based on several attributes from the previous medical data. This data was taken from a GitHub page, but this was previously even a kaggle competition.

Keywords: Class imbalance, IBM Watson studio, Decision Tree, Logistic Regression, Random Forrest

Introduction:

As I had mentioned in the Problem statement section, we are going to predict probability of a person facing a heart stroke. According to the world health organization, ischemic heart disease and stroke is one of the largest killers in the world. This is caused by different attributes such as places they live, smoking habits, type of work etc. We will try to predict this based on some attributes provided in the dataset.

In this project we are using supervised machine learning classification algorithms to solve this. In supervised machine learning each data attribute is a labelled class and we have to train a model by using different algorithms and predict the data that may have lack of class label. In this heart stroke data we can see that there are 43400 entries and has 12 attributes totally. Each entry in the dataset consist of information about an individual. In order to perform this task of classification I have done the task using pyspark to understand the dataset and also implement the machine learning algorithm. The cloud platform used in the project is the IBM Watson studio.

I had created a project in the IBM Watson studio, following which I had selected an environment in which python and spark is installed already. After created the environment, I had uploaded the input file (train and test data) within the project. I had added a new jupyter notebook in the cloud and loaded the data in the jupyter notebook. Following which I had added performed explanatory analysis in which there was the process of cleaning data, handling class imbalance. After explanatory analysis there are many string attributes in the dataset which must be converted to numerical attributes. In order to perform this, I have used the StringIndexer. After this I have used the vector assembler and standard scaler in the dataset.

I have implemented 3 machine learning algorithms in the dataset, namely they are Logistic Regression, Decision Tree algorithm and Random forest algorithm. Each of the algorithm will return a ROC value, following which I have applied hyper parameter tuning on each of the algorithm to increase the ROC value by choosing the best parameter.

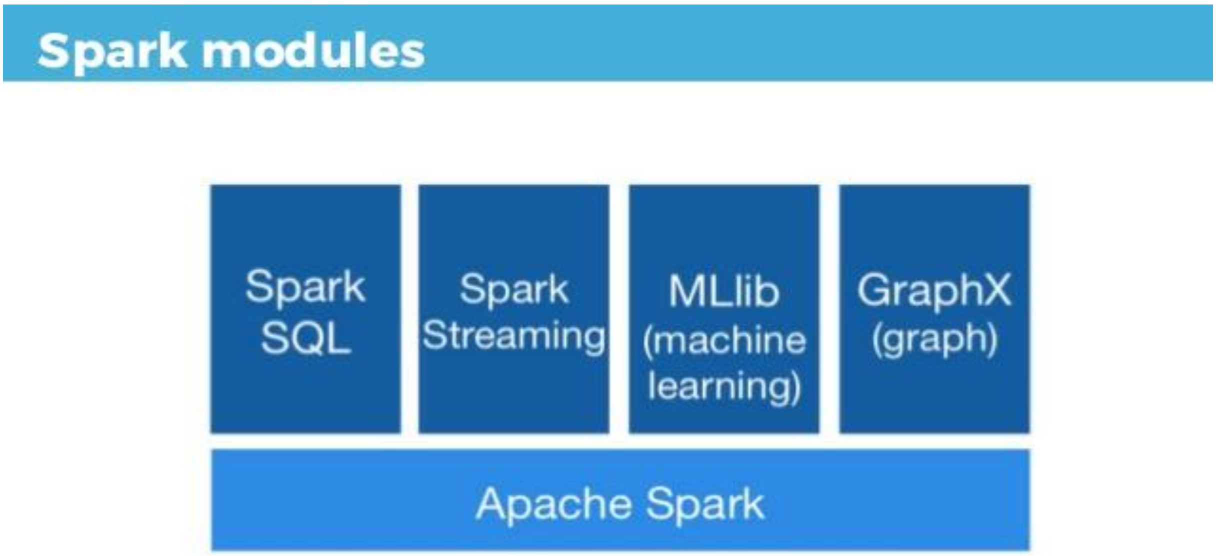
Following which I had run the whole setup as a job in IBM Watson studio, and logs can also be monitored in the cloud environment itself. One of the main reasons for choosing the IBM Watson studio is the ease of use, and presence of a lot of data connectors.

Discussion

In this part the whole project methodology will be explained that means the flow of project including IBM Watson studio configuration, loading dataset, explanatory analysis, data cleaning, handling class imbalance, upsampling, StringIndexer, Vector Assembler, standard scaler, machine learning model implementation (3 namely). All the algorithms are evaluated based on ROC and then we select the best model to make the prediction on this basis. After which we will apply each of the models are run on the test data as well.

Apache Spark:

Apache spark is the open source project which was provides cluster computing framework. Spark provides the interface for programming entire clusters with implicit data parallelism and fault tolerance. It is used for large scale data analytics as provides features such as parallelism. Apache spark can also be used to implement machine learning algorithm with the help of the MLlib library (This is part of the pyspark library of Python). Apache Spark has many modules which can be used to implement the machine learning models on the data. The screenshot of the spark modules in given below



But here in this project we are only using the Spark SQL and MLlib module in spark as mentioned above. Hence, we will only provide explanation for those modules.

Spark SQL: This module was built on top of the spark core, and it provides a data abstraction called Dataframes, which provides structured and semi-structured data. The Spark SQL module also provides the SQL language support, with command-line interfaces and JDBC/ODBC servers.

MLlib: Apache Spark provides a Machine learning API called MLlib. Pyspark has this machine learning API in Python as well. Many common machine learning and statistical algorithms have been implemented and shipped with MLlib which simplifies large scale machine learning pipelines, which includes classification, regression, feature extraction, transformation etc.

Watson Studio:

The Watson studio is a cloud platform which helps data Scientists to prepare data and build models in the cloud which can be scaled. The Watson studio provides the suits which are necessary to accelerate the machine learning and deep learning workflows, which can used to infuse AI into the business development model.

Creating a project in IBM Watson Studio

First creating an account in the Watson studio, we can create a project with the help of the “lite” plan. As this is the free configuration. I had created a project with name of “BDPP_Project”.

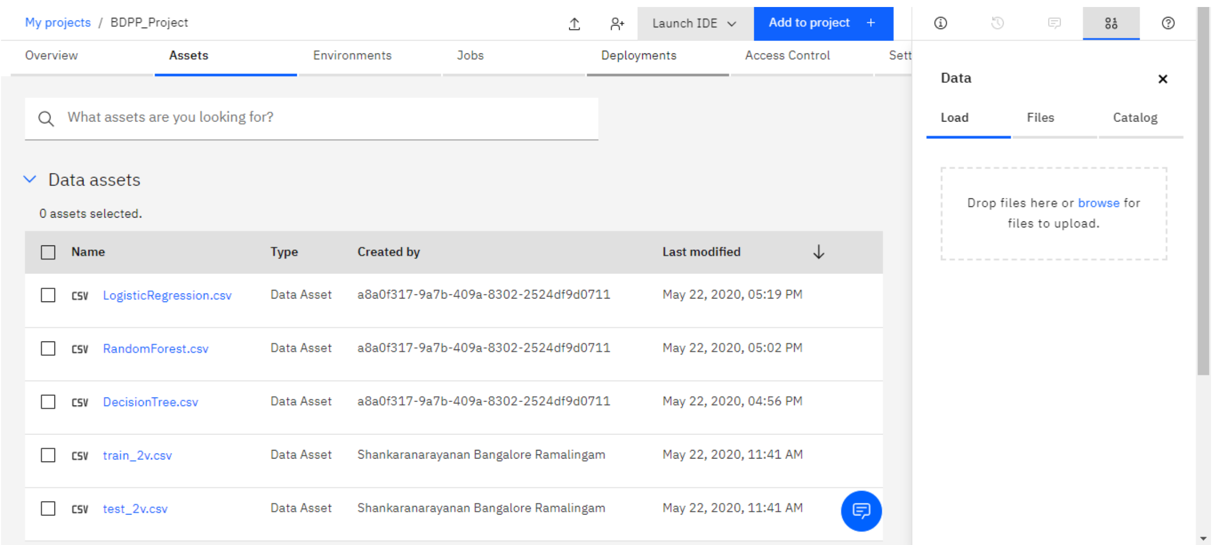
Name	Role	Collaborators	Date created	Last updated
BDPP_Project	Admin	SR	May 22, 2020	May 22, 2020

This project has an environment of Python 3.6. & Apache Spark 2.4, and an automatic cloud storage was created by the IBM Watson studio. Attaching the configuration screenshot below

Environment	Default Spark 2.4 & Python 3.6
Creator	IBM
Hardware configuration (Driver)	1 vCPU and 4 GB RAM
Hardware configuration (Executor)	1 vCPU and 4 GB RAM
Number of executors	2
Spark version	2.4
Software version	Python 3.6

Uploading Dataset in IBM Watson

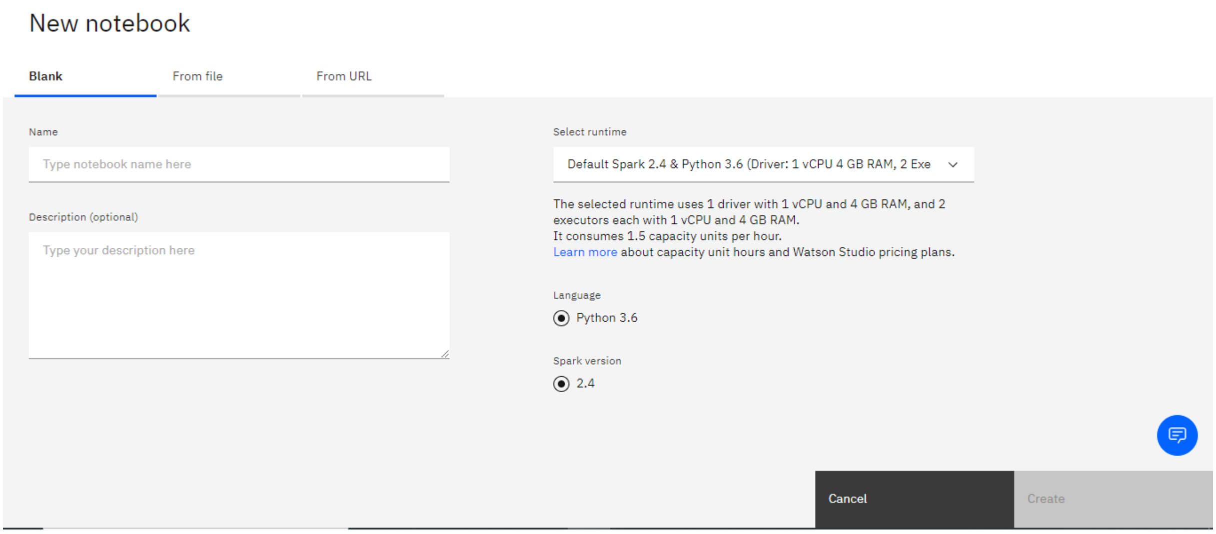
After creating an account and selecting the environment, the input files should be uploaded in the cloud, in Watson studio, there will be assets tab created after project creation. The input files have to be uploaded into here by clicking in the “New data asset” button. This is open a menu where we can upload the data into the cloud.



The new file is uploaded by clicking on the browse button, and we can upload the data into the cloud. In the above screenshot we can see that the input files are loaded in the tab, and this will serve as the bucket for our project.

Creating Jupyter Notebook

After uploading all the input files into the cloud, we have to create a notebook in which we will writing code, and where all the ML algorithms are implemented. In order to create a jupyter notebook in IBM Watson studio, click on the new Notebook button in the assets tab. Following which name of the Jupyter notebook is provided, then the runtime environment is chosen and create button is clicked. Attaching the screenshot below



Pyspark

After creating the notebook in the cloud, I had loaded the data into the Jupyter created in the cloud, attaching the screenshot of loading data in the cloud environment

```
In [193]: import ibmos2spark
# @hidden_cell
credentials = {
    'endpoint': 'https://s3-api.us-geo.objectstorage.service.networklayer.com',
    'service_id': 'iam-ServiceId-aaf06632-a925-4f1b-a1ad-4eb3f5182d4c',
    'iam_service_endpoint': 'https://iam.cloud.ibm.com/oidc/token',
    'api_key': 'cfdE0b2otcqGpFI7mW6b1nD_tZKjR9z-zqgROrxPmu2a'
}

configuration_name = 'os_4fe49bf4d14c41f9a72c9ef73f278c1a_configs'
cos = ibmos2spark.CloudObjectStorage(sc, credentials, configuration_name, 'bluemix_cos')

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
heart_stroke_train = spark.read\
  .format('org.apache.spark.sql.execution.datasources.csv.CSVFileFormat')\
  .option('header', 'true')\
  .load(cos.url('train_2v.csv', 'bdppproject-donotdelete-pr-bibcnvcar6fhub'))
heart_stroke_train.take(5)
```

Here we are using the IBM cloud object storage feature in the Watson studio, which will help us load the csv in the jupyter notebook. Similarly, even the test data is also loaded in the cloud.

```
testSet = spark.read\
  .format('org.apache.spark.sql.execution.datasources.csv.CSVFileFormat')\
  .option('header', 'true')\
  .load(cos.url('test_2v.csv', 'bdppproject-donotdelete-pr-bibcnvcar6fhub'))
testSet.take(5)
```

Exploratory Analysis

After loading the data into the cloud, I am doing exploratory analysis. Firstly, I am checking the data length and checking the schema for both the train and test data.

```
heart_stroke_train.count(),len(heart_stroke_train.columns)
(43400, 12)
```

```
testSet.count(),len(testSet.columns)
(18601, 11)
```

```
heart_stroke_train.printSchema()

root
 |-- id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: string (nullable = true)
 |-- hypertension: string (nullable = true)
 |-- heart_disease: string (nullable = true)
 |-- ever_married: string (nullable = true)
 |-- work_type: string (nullable = true)
 |-- Residence_type: string (nullable = true)
 |-- avg_glucose_level: string (nullable = true)
 |-- bmi: string (nullable = true)
 |-- smoking_status: string (nullable = true)
 |-- stroke: string (nullable = true)
```

```
testSet.printSchema()

root
 |-- id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- age: string (nullable = true)
 |-- hypertension: string (nullable = true)
 |-- heart_disease: string (nullable = true)
 |-- ever_married: string (nullable = true)
 |-- work_type: string (nullable = true)
 |-- Residence_type: string (nullable = true)
 |-- avg_glucose_level: string (nullable = true)
 |-- bmi: string (nullable = true)
 |-- smoking_status: string (nullable = true)
```

We can see that 12 and 11 attributes in the train and test set respectively. The data description is shown below in the table

Columns	**Description**
ID	Patient ID
Gender	Gender of the Patient
Age	Age of the Patient
Hypertension	1-Incase suffering from hypertension, 0-No hypertension
Heart Disease	1- In case of suffering from heart disease,0-No heart disease
Ever_married	Yes/No
Work_type	Type of Occupation
Residence Type	Area of residence(Rural, Urban)
Avg_glucose_level	Average_glucose_level(measured after meal)
BMI	Body mass index
Smoking_status	Patient’s smoke status
Stroke	0-no stroke, 1-suffered Stroke

Showing Data

The following screenshot shows the top 20 rows in the train and test dataset

```
heart_stroke_train.show()
```

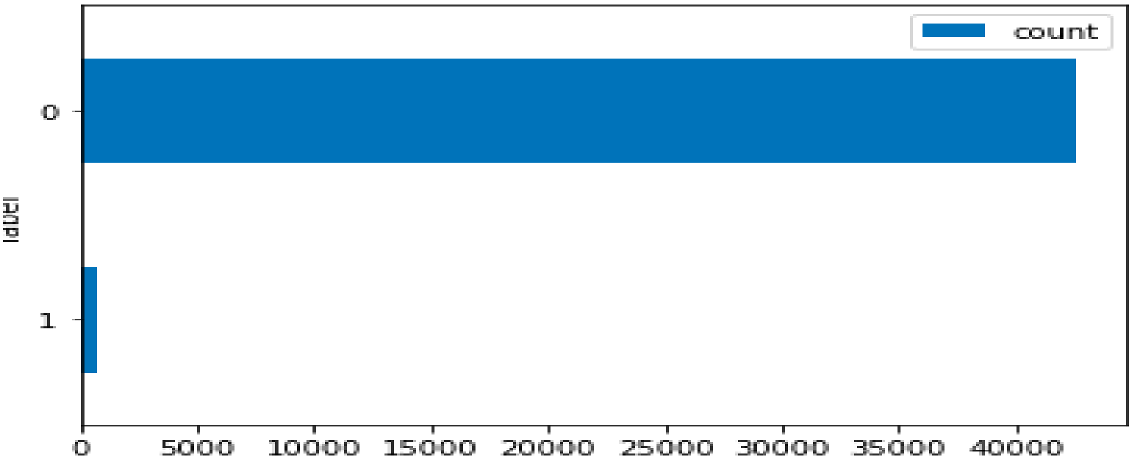
```
+-----+------+----+------------+-------------+------------+-------------+--------------+-----------------+----+---------------+
-----+
|   id|gender| age|hypertension|heart_disease|ever_married|    work_type|Residence_type|avg_glucose_level| bmi| smoking_status|
label|
+-----+------+----+------------+-------------+------------+-------------+--------------+-----------------+----+---------------+
-----+
|30669|  Male| 3.0|           0|            0|          No|     children|         Rural|            95.12|18.0|           null|
0|
|30468|  Male|58.0|           1|            0|         Yes|      Private|         Urban|            87.96|39.2|   never smoked|
0|
|16523|Female| 8.0|           0|            0|          No|      Private|         Urban|           110.89|17.6|           null|
0|
|56543|Female|70.0|           0|            0|         Yes|      Private|         Rural|            69.04|35.9|formerly smoked|
0|
|46136|  Male|14.0|           0|            0|          No| Never_worked|         Rural|           161.28|19.1|           null|
0|
|32257|Female|47.0|           0|            0|         Yes|      Private|         Urban|           210.95|50.1|           null|
0|
|52800|Female|52.0|           0|            0|         Yes|      Private|         Urban|            77.59|17.7|formerly smoked|
0|
|41413|Female|75.0|           0|            1|         Yes|Self-employed|         Rural|           243.53|27.0|   never smoked|
0|
|15266|Female|32.0|           0|            0|         Yes|      Private|         Rural|            77.67|32.3|         smokes|
0|
|28674|Female|74.0|           1|            0|         Yes|Self-employed|         Urban|           205.84|54.6|   never smoked|
0|
|10460|Female|79.0|           0|            0|         Yes|     Govt_job|         Urban|            77.08|35.0|           null|
0|
|64908|  Male|79.0|           0|            1|         Yes|      Private|         Urban|            57.08|22.0|formerly smoked|
0|
|63884|Female|37.0|           0|            0|         Yes|      Private|         Rural|           162.96|39.4|   never smoked|
0|
|37893|Female|37.0|           0|            0|         Yes|      Private|         Rural|             73.5|26.1|formerly smoked|
0|
|67855|Female|40.0|           0|            0|         Yes|      Private|         Rural|            95.04|42.4|   never smoked|
0|
|25774|  Male|35.0|           0|            0|          No|      Private|         Rural|            85.37|33.0|   never smoked|
0|
|19584|Female|20.0|           0|            0|          No|      Private|         Urban|            84.62|19.7|         smokes|
0|
|24447|Female|42.0|           0|            0|         Yes|      Private|         Rural|            82.67|22.5|   never smoked|
0|
|49589|Female|44.0|           0|            0|         Yes|     Govt_job|         Urban|            57.33|24.6|         smokes|
0|
|17986|Female|79.0|           0|            1|         Yes|Self-employed|         Urban|            67.84|25.2|         smokes|
0|
+-----+------+----+------------+-------------+------------+-------------+--------------+-----------------+----+---------------+
-----+
only showing top 20 rows
```

```
testSet.show()
```

```
+-----+------+----+------------+-------------+------------+-------------+--------------+-----------------+----+---------------+
|   id|gender| age|hypertension|heart_disease|ever_married|    work_type|Residence_type|avg_glucose_level| bmi| smoking_status|
+-----+------+----+------------+-------------+------------+-------------+--------------+-----------------+----+---------------+
|36306|  Male|80.0|           0|            0|         Yes|      Private|         Urban|            83.84|21.1|formerly smoked|
|61829|Female|74.0|           0|            1|         Yes|Self-employed|         Rural|            179.5|26.0|formerly smoked|
|14152|Female|14.0|           0|            0|          No|     children|         Rural|            95.16|21.2|           null|
|12997|  Male|28.0|           0|            0|          No|      Private|         Urban|            94.76|23.4|           null|
|40801|Female|63.0|           0|            0|         Yes|     Govt_job|         Rural|            83.57|27.6|   never smoked|
| 9348|Female|66.0|           1|            0|         Yes|      Private|         Urban|           219.98|32.2|   never smoked|
|51550|Female|49.0|           0|            0|         Yes|Self-employed|         Rural|            74.03|25.1|           null|
|60512|  Male|46.0|           0|            0|         Yes|     Govt_job|         Urban|            120.8|32.5|   never smoked|
|31309|Female|75.0|           0|            0|         Yes|Self-employed|         Rural|            78.71|28.0|   never smoked|
|39199|  Male|75.0|           0|            0|         Yes|Self-employed|         Urban|             77.2|25.7|         smokes|
|15160|Female|17.0|           0|            0|          No|      Private|         Rural|            78.16|21.9|           null|
|21705|Female|10.0|           0|            0|          No|     children|         Urban|           107.23|19.4|           null|
|19042|Female|47.0|           0|            0|         Yes|      Private|         Rural|             91.6|26.7|   never smoked|
|12249|Female|42.0|           0|            0|         Yes|      Private|         Urban|            83.05|32.3|           null|
|33104|Female|67.0|           0|            0|         Yes|     Govt_job|         Urban|            236.6|24.2|   never smoked|
|55264|Female|52.0|           0|            0|          No|Self-employed|         Urban|           109.49|24.5|   never smoked|
|29445|  Male|73.0|           0|            0|         Yes|Self-employed|         Rural|           109.66|40.0|           null|
|49013|Female|19.0|           0|            0|          No|      Private|         Rural|            88.51|22.1|           null|
|  276|  Male|15.0|           0|            0|          No|     children|         Rural|           101.36|22.3|           null|
|47721|Female|37.0|           0|            0|         Yes|     Govt_job|         Urban|           165.44|36.1|formerly smoked|
+-----+------+----+------------+-------------+------------+-------------+--------------+-----------------+----+---------------+
only showing top 20 rows
```

Handling missing data and upsampling

In this dataset, there are two output classes 0 and 1, and in the data 1 output class has 763 entries while the 0 output class has 42617, attaching the screenshot below



Now in order to handle the imbalance in data we have performed upsampling of the 1 output class.

```
heart_stroke_train_pos=heart_stroke_train.filter(heart_stroke_train["label"]==1)
heart_stroke_train_neg=heart_stroke_train.filter(heart_stroke_train["label"]==0)
```

The dataset is split into 2 classes which is 1 and 0, but here we will only upsample the 1 class, to handle the imbalance,

```
from pyspark.sql.functions import col
heart_stroke_train_pos= heart_stroke_train_pos.where(col('label')==1).sample(True,(42617/783)*3.0, seed = 2018)
heart_stroke_train_neg=heart_stroke_train_neg.where(col('label')==0)
heart_stroke_train = heart_stroke_train_neg.union(heart_stroke_train_pos)
```

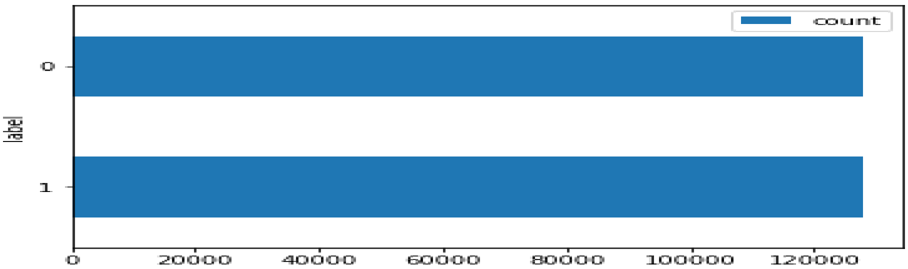
```
heart_stroke_train_neg=heart_stroke_train_neg.where(col('label')==0).sample(True, 3.0, seed = 2018)
heart_stroke_train = heart_stroke_train_neg.union(heart_stroke_train_pos)
```

As shown above the data is split into 2 classes and we have used the sample method to upsample the data. We can see that the fraction multiple is negative outcomes by positive outcomes.

But in order to scale up the data, we can see that the whole data is multiplied 3 times, hence the number of entries in increased in both the classes, and this will result in both class entries having almost equal number of entries, the below output justifies that

```
+-----+------+
|label| count|
+-----+------+
|    1|128078|
|    0|128061|
+-----+------+

<matplotlib.axes._subplots.AxesSubplot at 0x7fb0f58af978>
```



Handling Missing Data

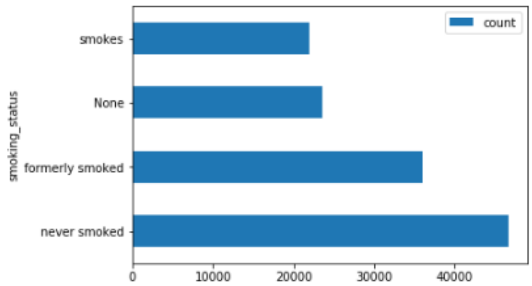
In this dataset, the columns “smoking_status” and “bmi” have null values in them, these null values have to be removed from the dataset. But the strategy used in both the columns are different, this is due to the datatype of both the columns. From the above **printSchema** and **dtypes** image we can see that “smoking_status” column is of string type and “bmi” is of integer type.

For the “smoking_status” column we have used to the **fillna** method provided by the dataframe object in spark. This method will replace all the null string to “No Info”. The below figure shows the table before and after using fillna method.

```
heart_stroke_train.filter(heart_stroke_train['label']==1).groupBy('smoking_status').count().orderBy('count',ascending=False).sho
w()
smoking_status_plot=heart_stroke_train.filter(heart_stroke_train['label']==1).groupBy('smoking_status').count().orderBy('count',
ascending=False)
data=smoking_status_plot.toPandas()
data.plot(x='smoking_status',y='count',kind='barh')
```

```
+--------------+-----+
| smoking_status|count|
+--------------+-----+
|   never smoked|46593|
|formerly smoked|35945|
|           null|23628|
|         smokes|21912|
+--------------+-----+

<matplotlib.axes._subplots.AxesSubplot at 0x7fb0f685d898>
```

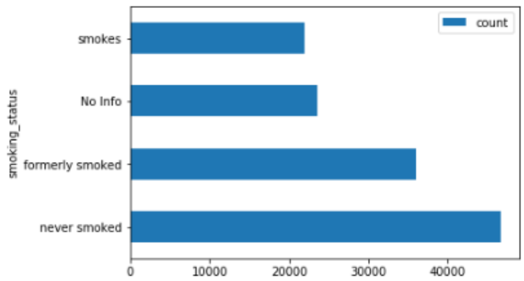


As we can see that there are 23628 entries which are in the “null” value section, and this null has been replaced with “No Info”.

```
smoking_status_plot=updated_info.filter(updated_info['label']==1).groupBy('smoking_status').count().orderBy('count',ascending=Fa
lse)
updated_info.filter(updated_info['label']==1).groupBy('smoking_status').count().orderBy('count',ascending=False).show()
data=smoking_status_plot.toPandas()
data.plot(x='smoking_status',y='count',kind='barh')
```

```
+--------------+-----+
| smoking_status|count|
+--------------+-----+
|   never smoked|46593|
|formerly smoked|35945|
|        No Info|23628|
|         smokes|21912|
+--------------+-----+

<matplotlib.axes._subplots.AxesSubplot at 0x7fb130034f60>
```



After this I noticed that the BMI column has null values present in it, I had replaced the null values present in the bmi column with the mean value.

```
+----+
| bmi|
+----+
|36.6|
|null|
|32.5|
|34.4|
|24.0|
|29.0|
|27.4|
|22.8|
|null|
|24.2|
|29.7|
|36.8|
|27.3|
|null|
|28.2|
|30.9|
|37.5|
|25.8|
|37.8|
|null|
+----+
```

```
+------------------+
|               bmi|
+------------------+
|29.202138491982318|
|              31.4|
|              30.9|
|              26.6|
|              26.4|
|              31.5|
|              30.7|
|              27.5|
|              28.5|
|              25.8|
|              24.2|
|              25.6|
|              27.3|
|              25.0|
|              30.3|
|              30.6|
|              29.3|
|              28.0|
|              27.0|
|              29.7|
+------------------+
```

The above screenshots shows the null value and the null values replaced after using the mean method on the dataframe. The below screenshot shows the code which was used to replace the value.

```
from pyspark.sql.functions import mean
mean = updated_info.select(mean(updated_info['bmi'])).collect()
mean_bmi = mean[0][0]
updated_info = updated_info.fillna(mean_bmi,['bmi'])
updated_info.filter(updated_info['label']==1).select('bmi')
```

```
DataFrame[bmi: double]
```

```
testSet=testSet.na.fill(mean_bmi,['bmi'])
```

Data Filtering

Here you can see the different filter data and their graphical representations.

```
+-------------+-----+
|    work_type|count|
+-------------+-----+
|      Private|72171|
|Self-employed|40901|
|     Govt_job|14695|
|     children|  311|
+-------------+-----+
```

```
+-----------+------+
|ever_married| count|
+-----------+------+
|        Yes|115023|
|         No| 13055|
+-----------+------+
```

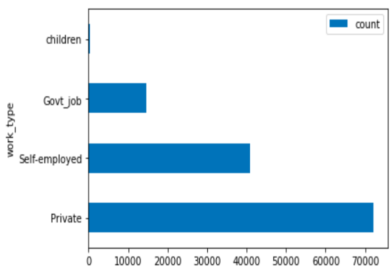
```
+--------------+-----+
|Residence_type|count|
+--------------+-----+
|         Urban|65082|
|         Rural|62996|
+--------------+-----+
```

```
+----+-----+
| age|count|
+----+-----+
|82.0| 5829|
|81.0| 7166|
|80.0| 7986|
|79.0|11350|
|78.0| 9512|
|77.0| 4018|
|76.0| 3865|
|75.0| 3715|
|74.0| 3930|
|73.0| 2418|
|72.0| 3442|
|71.0| 3167|
|70.0| 4078|
|69.0| 3266|
|68.0| 3405|
|67.0| 3753|
|66.0| 2733|
|65.0| 2989|
|64.0| 1356|
|63.0| 2947|
+----+-----+
only showing top 20 rows
```

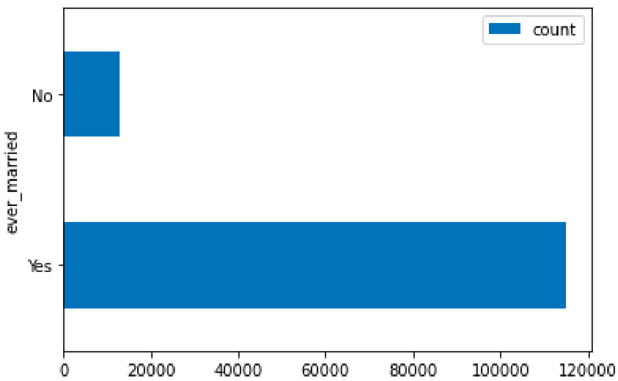
```
+-----------------+-----+
|              bmi|count|
+-----------------+-----+
|29.202138491982318|22764|
|             31.4| 1468|
|             30.9| 1459|
|             26.6| 1438|
|             26.4| 1406|
|             31.5| 1328|
|             30.7| 1312|
|             27.5| 1276|
|             28.5| 1228|
|             25.8| 1160|
|             24.2| 1144|
|             25.6| 1016|
|             27.3| 1006|
|             30.3|  999|
|             25.0|  999|
|             30.6|  993|
|             29.3|  983|
|             28.0|  971|
|             27.0|  961|
|             29.7|  947|
+-----------------+-----+
only showing top 20 rows
```

Graphical Representation

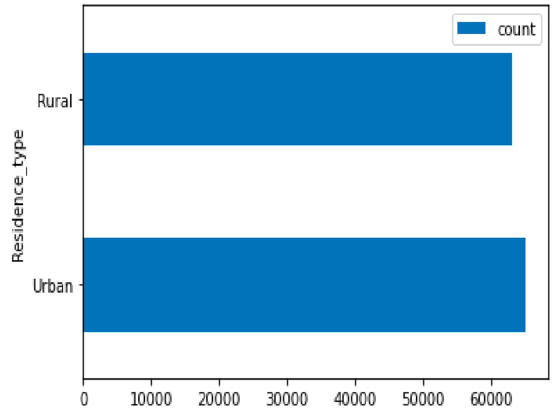
```
<matplotlib.axes._subplots.AxesSubplot at 0x11b1718d0>
```



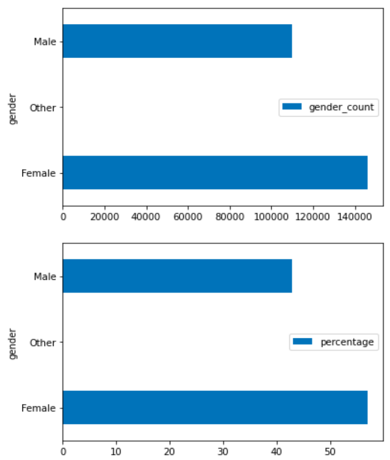

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b863278>
```



<matplotlib.axes._subplots.AxesSubplot at 0x11b988908>



<matplotlib.axes._subplots.AxesSubplot at 0x11cacf390>



String Indexer, OneHotEncoderEstimator

The machine learning algorithms prefer to working with numbers than string values, and in our dataset we have a lot of string attributes, the StringIndexer will help us replace the string values with numbers, if we look at the our schema we can see that the following columns were of string attributes.

1. Gender
2. Ever_married
3. Work_type
4. Residence_type
5. Smoking_status

The OneHotEncoderEstimator will be useful to create continuous values and this will create categorical features. The below screenshot shows the implementation of the gender column being changed to numerical attributes.

```
#StringIndexer for gender column
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoderEstimator
gender_indexer=StringIndexer(inputCol="gender",outputCol="genderIndex")
updated_info=gender_indexer.fit(updated_info).transform(updated_info)
updated_info.select(["gender","genderIndex"]).show(5)
```

```
+------+-----------+
|gender|genderIndex|
+------+-----------+
|  Male|        1.0|
|  Male|        1.0|
|  Male|        1.0|
|Female|        0.0|
|Female|        0.0|
+------+-----------+
only showing top 5 rows
```

```
#OneHotEncoderEstimate for the gender column

gender_encoder=OneHotEncoderEstimator(inputCols=["genderIndex"],outputCols=["genderVec"])
model=gender_encoder.fit(updated_info)
updated_info=model.transform(updated_info)
updated_info.select("genderVec").take(5)
```

```
[Row(genderVec=SparseVector(2, {1: 1.0})),
 Row(genderVec=SparseVector(2, {1: 1.0})),
 Row(genderVec=SparseVector(2, {1: 1.0})),
 Row(genderVec=SparseVector(2, {0: 1.0})),
 Row(genderVec=SparseVector(2, {0: 1.0}))]
```

The above 2 screenshots are only an example of one of the attributes, similarly the String Indexer and OneHotEncoderEstimator has been implemented for all the String attributes.

Vector Assembler. Standard Scalar

Vector assembler is used to combine all the attributes in one feature col and this feature or scaled features col is used for training the model as well.

Stand scaler is used for scale the values in feature col. If there are values that have much difference, then the stand scaler scale all the values and generate scaled features col. You can see the following snap.

```
+--------------------+--------------------+
|            features|      scaledFeatures|
+--------------------+--------------------+
|(11,[1,2,6,7,9],[...|[-1.1536667115755...|
|[0.0,1.0,58.0,1.0...|[-1.1536667115755...|
|[0.0,1.0,58.0,1.0...|[-1.1536667115755...|
|(11,[0,2,5,6,7,9]...|[0.86679808460811...|
|(11,[0,2,5,6,7,9]...|[0.86679808460811...|
+--------------------+--------------------+
only showing top 5 rows
```

Splitting Data

Here I am splitting the training dataframe into train dataset and the validation set. In my project the data is split in the 70:30 ratio, wherein 70 is the train dataset and 30 is the validation set. After splitting the data, the number of entries in each of the sets are provided in the below screenshot

```
: val_data.count(),len(val_data.columns)
: (76773, 24)
: train_data.count(),len(train_data.columns)
: (179366, 24)
```

Machine Learning

Machine learning basically performs some rules on the dataset and make prediction of test data. This is supervised machine learning task and in supervised machine learning the data set always given with the label class. Now the data has been ready for the processing for machine leaning. Now I will use different number of algorithm and will find the ROC that will help to evaluate the algorithm.

Logistic Regression

The logistic Regression is classification algorithm, wherein we predict the label class by using the information provided in the dataset. I train the model in using the training data, and then run the prediction in the test set. But to find the ROC of the model, I have used the validation dataset which I had created. The results dataframe from the test set for each entry is shown below.

```
+-----+----------+--------------------+----+-------------+
|label|prediction|         probability| age|    work_type|
+-----+----------+--------------------+----+-------------+
|    0|       0.0|[0.80379707391931...|37.0|      Private|
|    0|       0.0|[0.96184693802282...|21.0|      Private|
|    0|       1.0|[0.22446153607421...|79.0|      Private|
|    0|       0.0|[0.78616625016903...|44.0|      Private|
|    0|       0.0|[0.67685275119944...|54.0|Self-employed|
|    0|       0.0|[0.60146018483361...|56.0|      Private|
|    0|       1.0|[0.41770503525074...|67.0|      Private|
|    0|       0.0|[0.98620092203542...| 5.0|     children|
|    0|       0.0|[0.98620092203542...| 5.0|     children|
|    0|       1.0|[0.45922127555439...|68.0|      Private|
|    0|       1.0|[0.45922127555439...|68.0|      Private|
|    0|       0.0|[0.98675212113989...| 5.0|     children|
|    0|       0.0|[0.97903799113763...|13.0|      Private|
|    0|       0.0|[0.97903799113763...|13.0|      Private|
|    0|       0.0|[0.94493038029532...|26.0|      Private|
|    0|       0.0|[0.94493038029532...|26.0|      Private|
|    0|       1.0|[0.33123146765210...|52.0|Self-employed|
|    0|       0.0|[0.90034772131555...|36.0|      Private|
|    0|       1.0|[0.20142166051380...|74.0|     Govt_job|
|    0|       0.0|[0.79407088763163...|46.0|     Govt_job|
+-----+----------+--------------------+----+-------------+
```

Decision Tree

Decision tree algorithm belongs to a family of supervised machine learning algorithm. The main goal of the decision tree algorithm is predict the output (class or target value) by learning simple decision rules inferred from prior data. Like Logistic Regression we have used train dataset to train model, and done the prediction in the test model. The ROC of the model was 0.82 before hyperparameter tuning, and after that ROC was 0.88

```
DataFrame[label: int, prediction: double, probability: vector,

+-----+----------+--------------------+----+-------------+
|label|prediction|         probability| age|    work_type|
+-----+----------+--------------------+----+-------------+
|    0|       1.0|[0.15041696695150...|80.0|Self-employed|
|    0|       1.0|[0.15041696695150...|80.0|Self-employed|
|    0|       0.0|[0.80543530543530...|37.0|      Private|
|    0|       0.0|           [1.0,0.0]|21.0|      Private|
|    0|       0.0|           [1.0,0.0]|44.0|      Private|
|    0|       1.0|[0.27169359664871...|79.0|      Private|
|    0|       1.0|[0.27169359664871...|79.0|      Private|
|    0|       1.0|[0.48888888888888...|44.0|      Private|
|    0|       1.0|[0.48888888888888...|44.0|      Private|
|    0|       0.0|           [1.0,0.0]|34.0|      Private|
|    0|       1.0|[0.15882967607105...|54.0|Self-employed|
|    0|       0.0|[0.60396039603960...|67.0|      Private|
|    0|       0.0|           [1.0,0.0]| 5.0|     children|
|    0|       0.0|           [1.0,0.0]| 5.0|     children|
|    0|       0.0|           [1.0,0.0]| 5.0|     children|
|    0|       0.0|[0.60396039603960...|68.0|      Private|
|    0|       0.0|           [1.0,0.0]| 4.0|     children|
|    0|       0.0|           [1.0,0.0]| 4.0|     children|
|    0|       1.0|[0.48888888888888...|44.0|      Private|
|    0|       0.0|           [1.0,0.0]| 5.0|     children|
+-----+----------+--------------------+----+-------------+
only showing top 20 rows
```

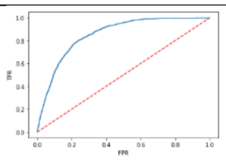
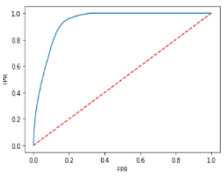
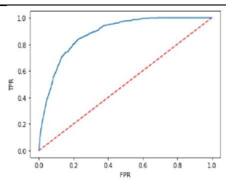
Random Forest Tree

Random forest, like its name implies, consists of many individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The ROC for this 0.85 and after parameter tuning it is 0.88.

```
+-----+----------+--------------------+----+-------------+
|label|prediction|         probability| age|    work_type|
+-----+----------+--------------------+----+-------------+
|    0|       0.0|[0.85518521905847...|37.0|      Private|
|    0|       0.0|[0.86267010483866...|21.0|      Private|
|    0|       1.0|[0.24102076143563...|79.0|      Private|
|    0|       0.0|[0.81515478841869...|44.0|      Private|
|    0|       0.0|[0.69635537790376...|54.0|Self-employed|
|    0|       0.0|[0.61814925040722...|56.0|      Private|
|    0|       1.0|[0.29385888361690...|67.0|      Private|
|    0|       0.0|[0.90740495233839...| 5.0|     children|
|    0|       0.0|[0.90740495233839...| 5.0|     children|
|    0|       1.0|[0.30806297945393...|68.0|      Private|
|    0|       1.0|[0.30806297945393...|68.0|      Private|
|    0|       0.0|[0.94025611990864...| 5.0|     children|
|    0|       0.0|[0.91534075092428...|13.0|      Private|
|    0|       0.0|[0.91534075092428...|13.0|      Private|
|    0|       0.0|[0.90205032490179...|26.0|      Private|
|    0|       0.0|[0.90205032490179...|26.0|      Private|
|    0|       1.0|[0.44685142955909...|52.0|Self-employed|
|    0|       0.0|[0.83269488014656...|36.0|      Private|
|    0|       1.0|[0.19920038422445...|74.0|     Govt_job|
|    0|       0.0|[0.81672608781590...|46.0|     Govt_job|
+-----+----------+--------------------+----+-------------+
```

Model Selection:

We can see that, there were 3 models which were implemented, and the best models are chosen based on the ROC of the each of the models. The below table compares different models which are used in the project.

Category	Logistic Regression	Decision Tree Classification	Random Forest Classification
AreaUnderROC before Parameter tuning	0.8515995624882962	0.8601771331423859	0.8672775922191699
AreaUnderROC after Parameter tuning	0.8519365138452384	0.8901265969186091	0.880087906275998
ROC Curve after hyperparameter tuning			

Creating Output Link on Jupyter Notebook

Finally, after choosing best model, and the prediction being applied for each of the models in the test data. After that the predicted values from the test dataframe will be stored in csv file in the cloud assests.

The below screenshot shows the ouput csv stored in the cloud, and they can be downloaded from there as csv.

Name	Type	Created by	Last modified
CSV LogisticRegression.csv	Data Asset	a8a0f317-9a7b-409a-8302-2524df9d0711	May 22, 2020, 08:49 PM
CSV RandomForest.csv	Data Asset	a8a0f317-9a7b-409a-8302-2524df9d0711	May 22, 2020, 08:32 PM
CSV DecisionTree.csv	Data Asset	a8a0f317-9a7b-409a-8302-2524df9d0711	May 22, 2020, 08:26 PM

We can see that this is done for each of the models.

Running as Job in Watson Studio

After completing the code in the jupyter notebook, i have submitted the cluster job with main python file which is present in the cluster, we can see that logs of the job submitted in the cloud, the screenshot containing the job details is shown below.

Job name	Associated asset	Last run	Started by	Created by
This is the result	Notebook	Finished Jun 13, 2020, 01:01 AM	Shankaranarayanan Bangalore Ramalingam Jun 13, 2020, 12:40 AM	Shankaranarayanan Bangalore Ramalingam

The below screenshot shows the logs generated when running the job in the cloud.

```
...
[I 2020-06-12 19:04:18.243 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type status on channel iopub
[I 2020-06-12 19:04:18.249 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] Sending Shell Message type execute_request
[I 2020-06-12 19:04:18.324 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type status on channel iopub
[I 2020-06-12 19:04:18.335 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type execute_input on channel iopub
[I 2020-06-12 19:04:20.063 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:25.057 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:33.067 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:34.075 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:38.078 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:43.090 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:44.092 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:45.098 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:46.101 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:47.080 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:48.102 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:49.106 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:50.106 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:51.109 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:52.110 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:53.119 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:54.113 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:55.099 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:56.110 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:57.109 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
[I 2020-06-12 19:04:58.122 NotebookApp] [b179f840-e4e8-45df-a106-ade5593122f0] registered message type spark_monitor_msg on channel iopub
```

Comparing local performance and cloud performance

The main aim of using the project is cloud is to have a better preformance, where in we can excute the same python file much faster. I have run my jupyter notebook in my local machine and also in IBM watson studio. Attaching the 2 times at the below table.

Local Machine Execution Time (seconds)	Cloud Execution Time(seconds)
1927.7977	44.85