# CIS5560 Term Project Tutorial


**Authors: Aishwarya Gupta, Priyanka Thota, Rahul Bhogale, Prathushkumar Dathuri**

**Instructor: Dr. Jongwook Woo**

**Date: 05/18/2023**

# Lab Tutorial

Aishwarya Gupta (agupta25@calstatela.edu), Rahul Bhogale (rbhogal@calstatela.edu), Priyanka Thota (pthota2@calstatela.edu), Prathushkumar Dathuri (pdathur@calstatela.edu)

05/18/2023

# INSURING SMILES: PREDICTING ROUTINE DENTAL COVERAGE WITH ML ALGORITHMS

---

## Problem Statement:

To predict the % of routine dental services covered under Health Insurance Plans or not.

## Objectives:

In this hands-on lab, you will learn how to:

- Import the dataset manually from CMS.gov dataset
- Create Spark cluster and Notebook
- Loading and Cleaning of Dataset i.e. (Data Preparation)
- Feature Selection & Defining Pipeline
- Data Split
- Train & Test Models

- Evaluation & Prediction of various models
- Compare metrics and conclude the results
- Calculate Feature Importance of the Best Model – (GBT)
- Run the project.py file into PySpark Hadoop cluster
- Observe & compare the results for the actual dataset

# Use GitBash CLI to run the below codes for Specs & Configs:

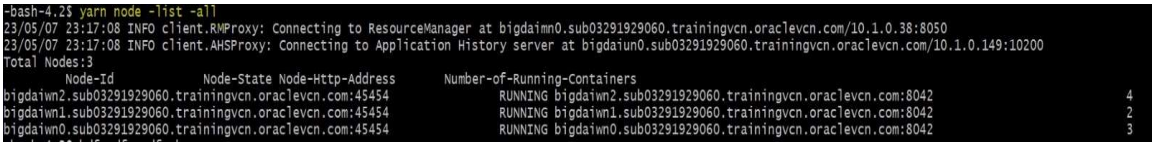
Download GitBash Terminal from https://gitforwindows.org/ [3]

Git Bash comes included as part of the Git for Windows package. Download and install Git for Windows like other Windows applications. Once downloaded find the included .exe file and open to execute Git Bash.

## Hadoop Platform Specs:

- Hadoop version: 3.1.2
  `$ hdfs version`


- # Of nodes: 5 : Master nodes - 2 and Worker nodes – 3
  `$ yarn node -list -all`




- Total Memory Size: 390.7GB
  `$ hdfs dfs -df -h`

## CPU Configuration:

- CPU Speed: 1995.312 MHz
- # Of CPU cores: 6

```
$ lscpu
```

```
-bash-4.2$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Platinum 8167M CPU @ 2.00GHz
Stepping:              4
CPU MHz:               1995.312
BogoMIPS:              3990.62
Virtualization:        VT-x
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
L3 cache:              16384K
NUMA node0 CPU(s):     0-7
```

# PART -1

## Step 1: Download the dataset from CMS.gov website

1. Download the ***benefits and cost sharing csv files*** for the year 2017 to 2021 from the below website and save it as ***'benefit.csv'*** [1].
   https://www.cms.gov/cciio/resources/data-resources/marketplace-puf
   The above dataset is a balanced dataset with approximately 60% of health insurance plans covering dental services compared to remaining 40%.

2. ***Randomly select up to 50,000 rows*** from the above file and create a new dataset for testing in Databricks as it has limitations of uploading the data up to few Mbs and save it as ***'benefits1.csv'***. Run the below code in Jupyter Notebook [5] https://jupyter.org/

```
import pandas as pd
import random

# Load the original dataset file into a DataFrame
df = pd.read_csv('\filepath\original_dataset.csv')

# Randomly select 50,000 rows from the DataFrame
num_rows = min(50000, len(df))  # Select either 50,000 rows
or the total number of rows if it is less than 50,000
random.seed(42)  # Set a seed for reproducibility
indices = random.sample(range(len(df)), num_rows)
df_sample = df.loc[indices]
```

```
# Save the selected rows as a new CSV file
df_sample.to_csv('benefits1.csv', index=False)
```

In [1]:
```
import pandas as pd
import random

# Load the original dataset file into a DataFrame
df = pd.read_csv('C:\\Users\\agupta25\\Desktop\\5560 project\\benefits_cost_sharing.csv')

# Randomly select 10,000 rows from the DataFrame
num_rows = min(50000, len(df))  # Select either 50,000 rows or the total number of rows if it's less than 50,000
random.seed(42)  # Set a seed for reproducibility
indices = random.sample(range(len(df)), num_rows)
df_sample = df.loc[indices]

# Save the selected rows as a new CSV file
df_sample.to_csv('benefits1.csv', index=False)
df_sample
```

```
C:\Users\agupta25\AppData\Local\Temp\ipykernel_20236\2633191297.py:5: DtypeWarning: Columns (9,12,19) have mixed types. Specify
dtype option on import or set low_memory=False.
  df = pd.read_csv('C:\\Users\\agupta25\\Desktop\\5560 project\\benefits_cost_sharing.csv')
```

Out[1]:

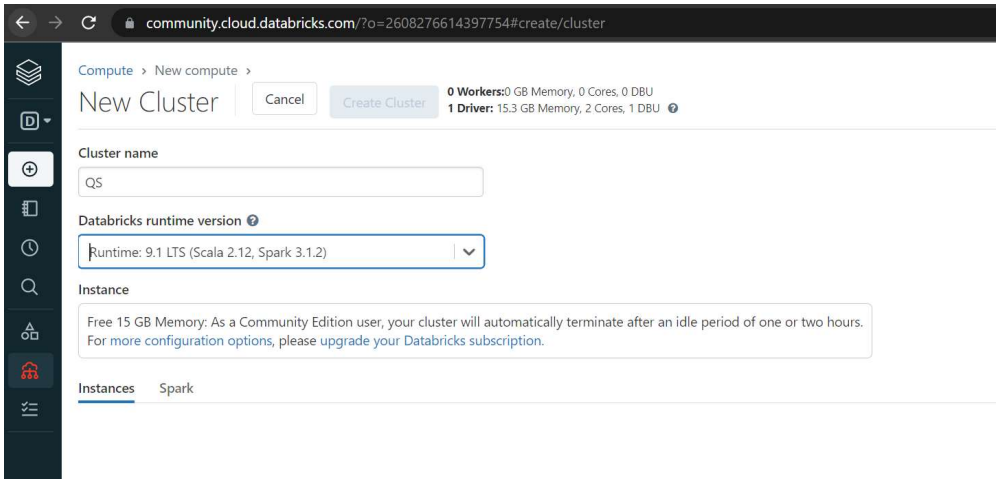
| | BusinessYear | StateCode | IssuerId | SourceName | ImportDate | StandardComponentId | PlanId | BenefitName | CopayInnTier1 | CopayInnTier2 | . |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **5363900** | 2021 | MI | 98185 | SERFF | 2020-12-22 20:15:52 | 98185MI0180012 | 98185MI0180012-05 | Routine Eye Exam (Adult) | NaN | NaN | . |
| **933912** | 2017 | MI | 37651 | SERFF | 2016-10-19 03:51:02 | 37651MI0150005 | 37651MI0150005-00 | Dental Check-Up for Children | NaN | NaN | . |
| **209805** | 2017 | MO | 32753 | HIOS | 2017-01-19 02:20:17 | 32753MO0770023 | 32753MO0770023-03 | Routine Eye Exam (Adult) | NaN | NaN | . |

# Step 2: Create a Spark Cluster and Notebook

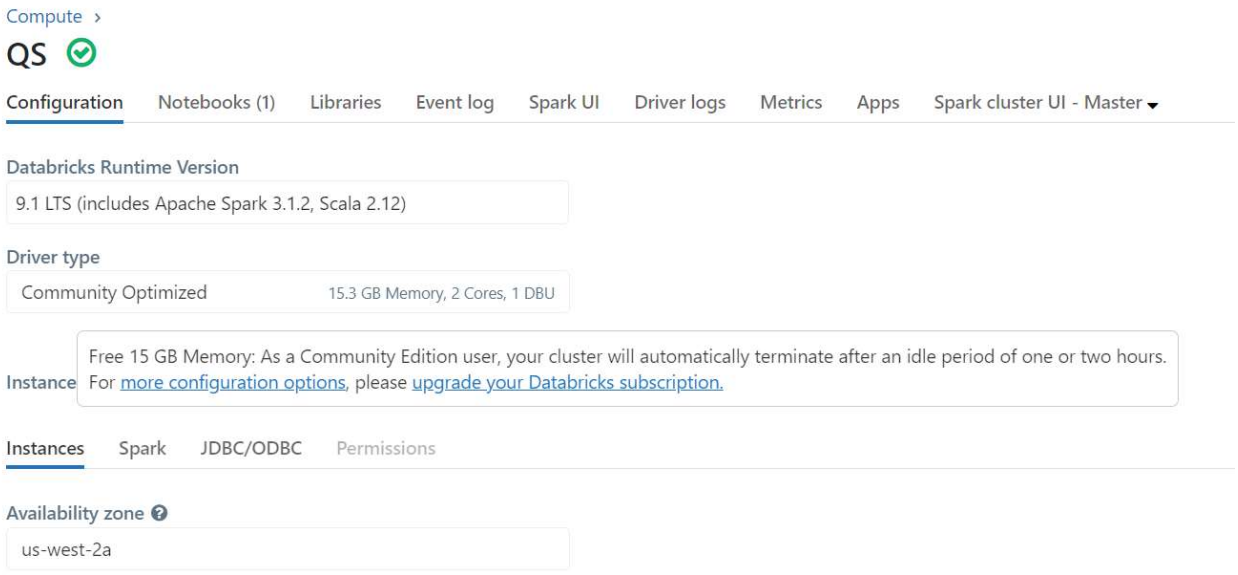
1. Log in at [4] https://community.cloud.databricks.com/

2. In the sidebar of the page above, click Compute.

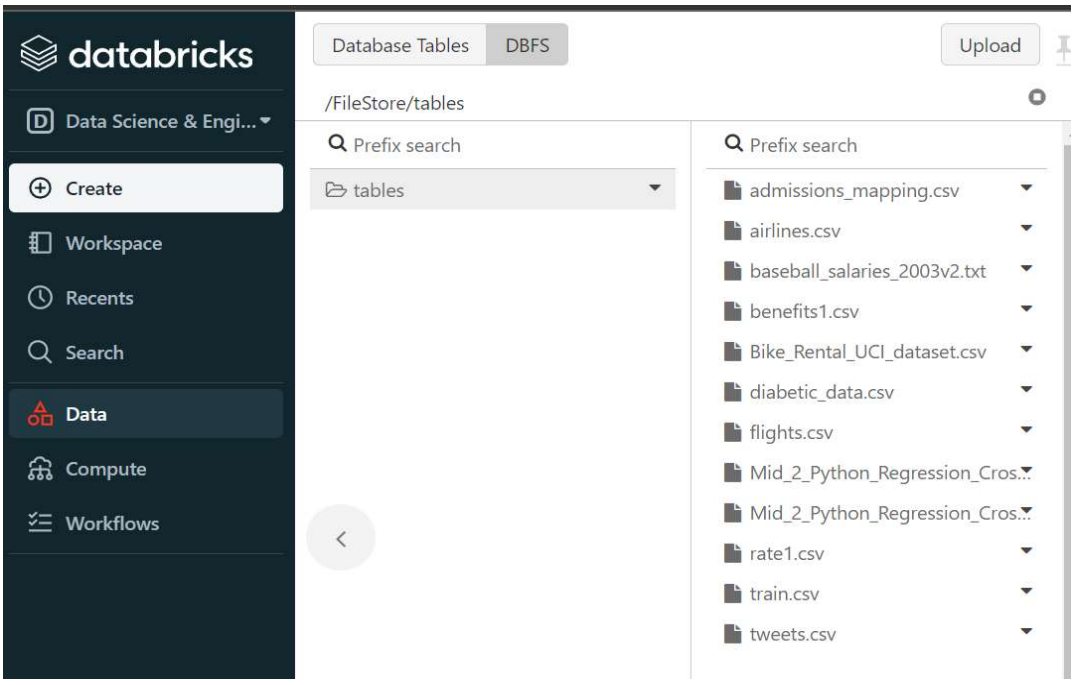
3. On the Compute page, click Create Cluster. a. Create a cluster [4] with “select 9.1 LTS (Scala 2.12, Spark 3.1.2)” b. It may take 5-10 minutes to see the cluster

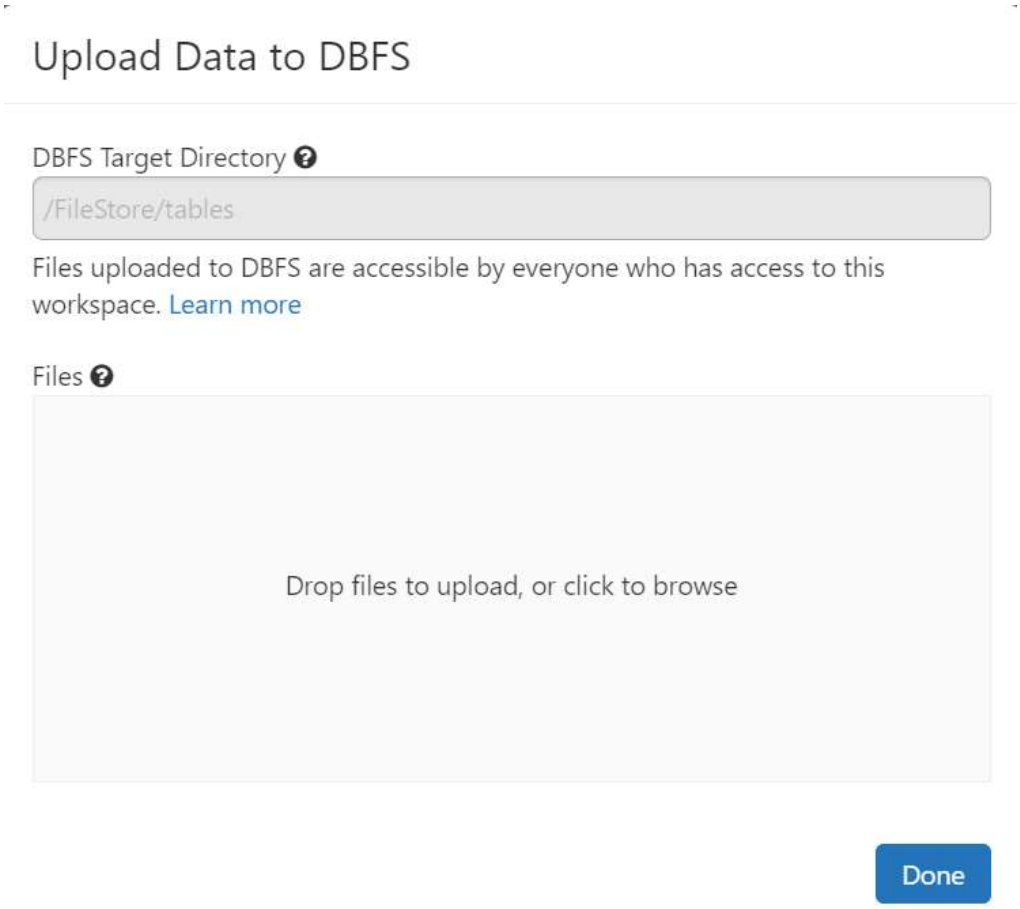


Once, the cluster is created , we can see the following specification of Cluster on the compute page, click on it and you would see the existing cluster.

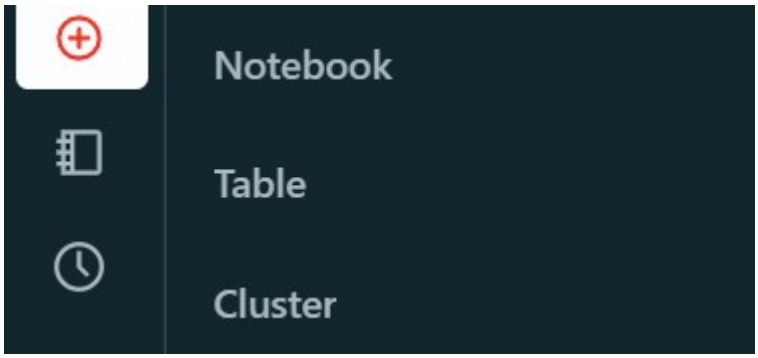


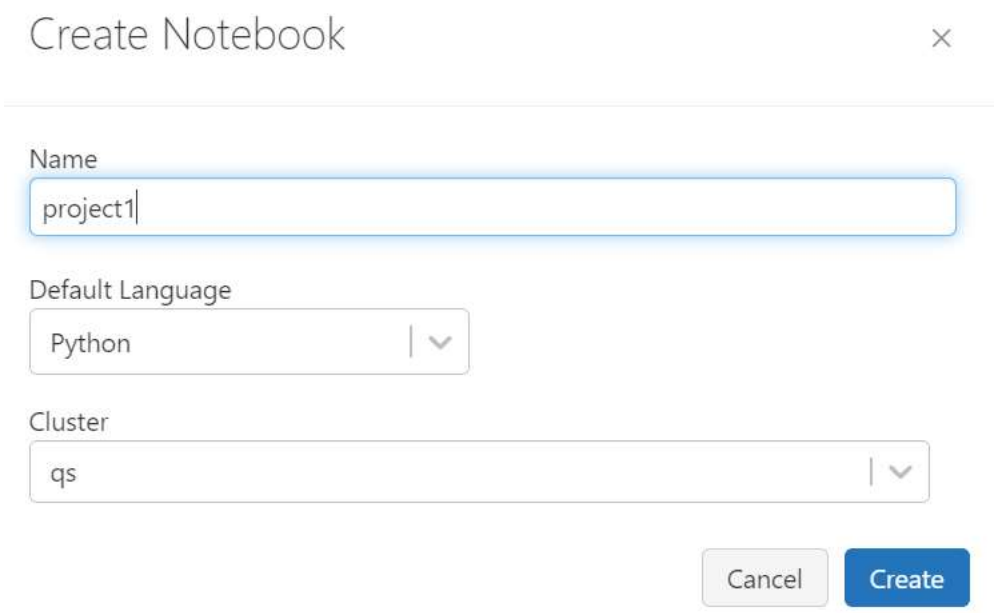

1. a. Under Data section, select DBFS tab and under create tables section, select the file to upload.






b. Once the file or dataset is uploaded into DBFS, you need to create a new notebook by clicking on Create page on right bar menu which is shown below.





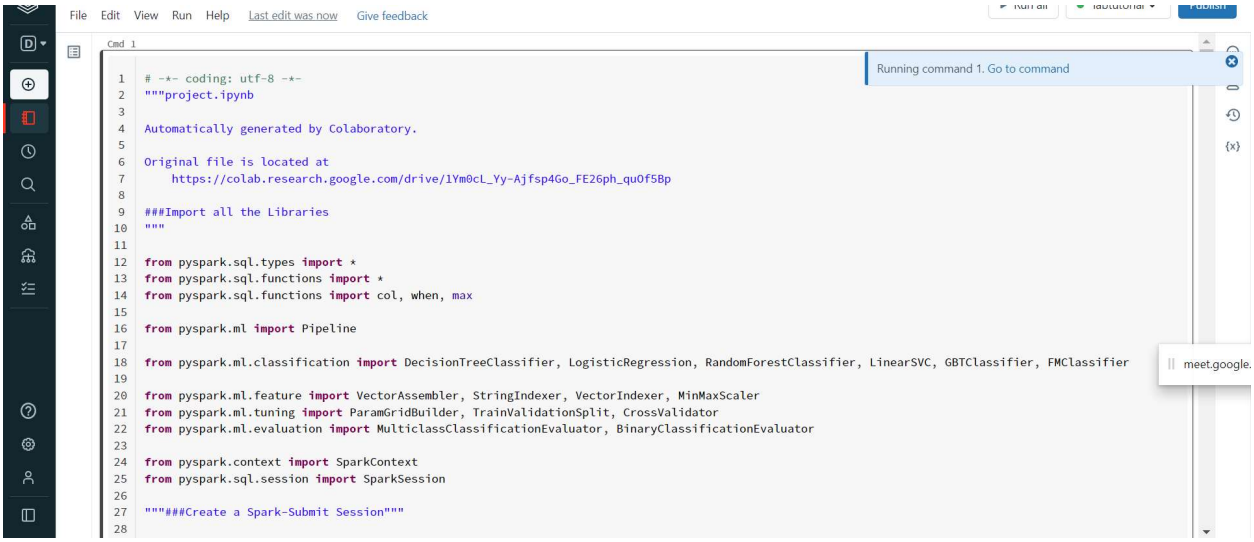

Once the notebook is created, you would be directed to databricks notebook as shown below.


Connect your notebook to Cluster – QS that was created above.

# Step 3: Import all the Libraries

Once the above notebook is created, copy the below code, and paste it in 1st cell import all the libraries as mentioned below.

```
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.sql.functions import col, when, max

from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier,
LogisticRegression, RandomForestClassifier, LinearSVC, GBTClassifier,
FMClassifier

from pyspark.ml.feature import VectorAssembler, StringIndexer,
VectorIndexer, MinMaxScaler
from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit,
CrossValidator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator,
BinaryClassificationEvaluator

from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
```




# Step 4: Create a Spark-Submit Session

Write the below code in the following next cells.

This step is done to create a spark-submit session to run the same file with .py in Command Line Interface (PySpark CLI).

```
PYSPARK_CLI = True # conditional statement to run only at shell
if PYSPARK_CLI:
    sc = SparkContext.getOrCreate()
    spark = SparkSession(sc)
# Limit the log
spark.sparkContext.setLogLevel("WARN")
```

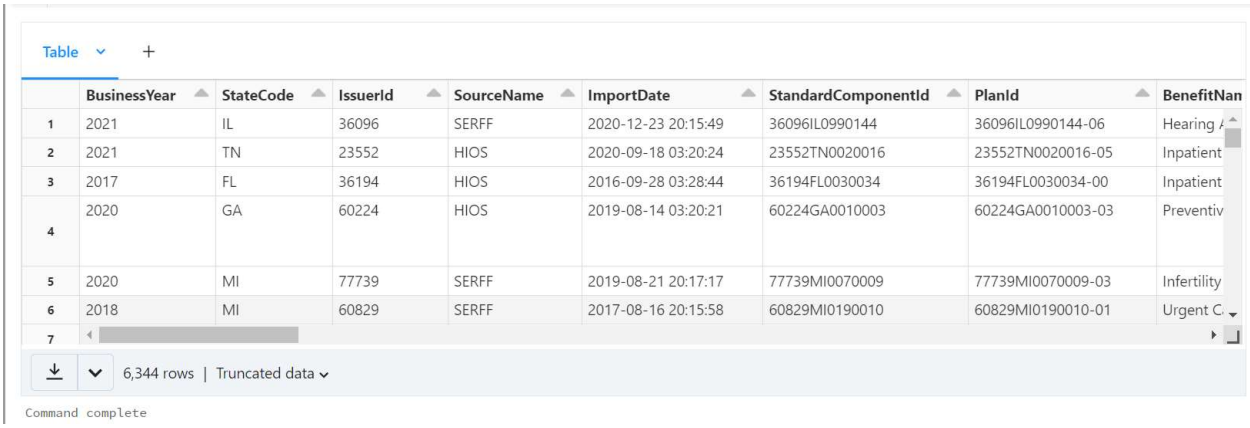
# Step 5: Load the sample dataset from DBFS

In this Step, we need to load the sample dataset into DataFrame for our model prediction and analysis that is present in DBFS.

Copy the below code and paste it in the following cells.

The dataset loaded from DBFS has `benefits1.csv` as filename.

```
# Oracle BDCE

#csv = spark.read.csv('/user/agupta25/project/benefits1.csv',
inferSchema=True, header=True)



# File location and type

file_location = "/FileStore/tables/benefits1.csv"

file_type = "csv"

# CSV options

infer_schema = "true"

first_row_is_header = "true"

delimiter = ","



df = spark.read.format(file_type) \

  .option("inferSchema", infer_schema) \

  .option("header", first_row_is_header) \

  .option("sep", delimiter) \

  .load(file_location)

  display(df)
```

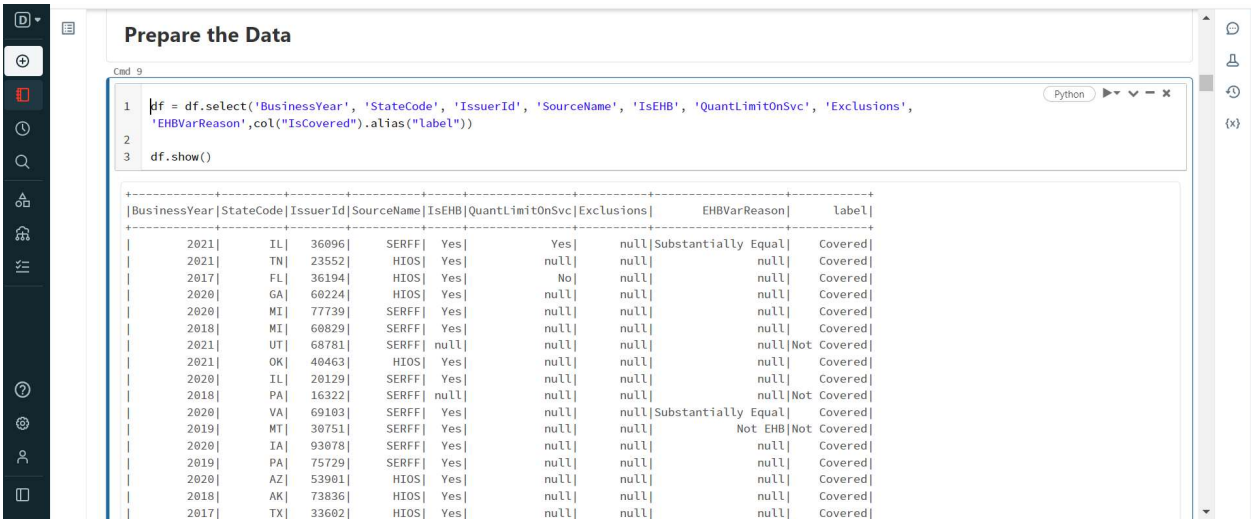




# Step 6: Prepare the Data

From the above `df` we are selecting **only independent columns** for our prediction such as `BusinessYear, StateCode, IssuerId, SourceName, IsEHB, QuantLimitOnSvc, Exclusions, EHBVarReason, IsCovered` to keep our prediction unbiased.

Copy the below code and paste it in the following cell.

```
df = df.select('BusinessYear', 'StateCode', 'IssuerId', 'SourceName',
'IsEHB', 'QuantLimitOnSvc', 'Exclusions',
'EHBVarReason',col("IsCovered").alias("label"))
df.show()
```




To see the schema of dataframe use this code .

```
df.printSchema()
```

Cmd 10

```
1 df.printSchema()
```

```
root
 |-- BusinessYear: integer (nullable = true)
 |-- StateCode: string (nullable = true)
 |-- IssuerId: integer (nullable = true)
 |-- SourceName: string (nullable = true)
 |-- IsEHB: string (nullable = true)
 |-- QuantLimitOnSvc: string (nullable = true)
 |-- Exclusions: string (nullable = true)
 |-- EHBVarReason: string (nullable = true)
 |-- label: string (nullable = true)


Command complete
```

# Step 7: count the null values from prediction col: ‘IsCovered’

From the below code, we are counting the null values from the sample dataset of our `label : IsCovered` column.

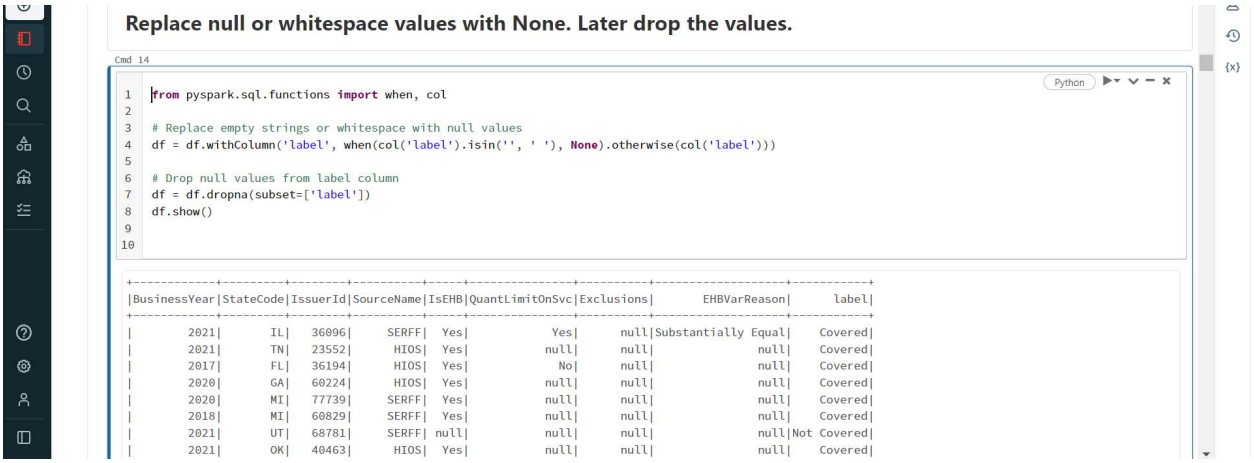
Copy the below code and paste it in the next cell.

```
from pyspark.sql.functions import col, sum

# Assuming that `df` is a Spark DataFrame and `label` is a column in `df`

null_count = df.select(sum(col("label").isNull().cast("integer"))).collect()[0][0]

print(null_count)
```


# Step 8: Replace null or whitespace values with None. Later drop the values.

In this step, we are replacing the null or empty strings with `‘None’` and later dropping it for our prediction to remain unbiased.

```
from pyspark.sql.functions import when, col

# Replace empty strings or whitespace with null values

df = df.withColumn('label', when(col('label').isin('', ' '), None).otherwise(col('label')))

# Drop null values from label column

df = df.dropna(subset=['label'])

df.show()
```

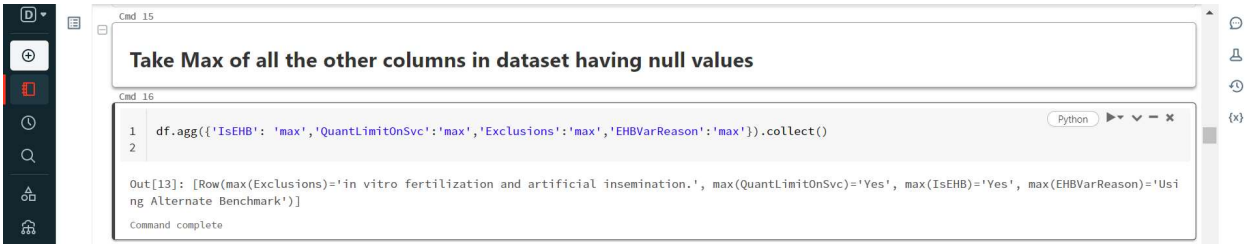



# Step 9: Take Max of all the other columns in dataset having null values

After removing the above null values from ‘label’ column, replace the existing null values from other columns of df with maximum values.

To find the maximum values of other columns, copy the below code and paste it in a new cell.

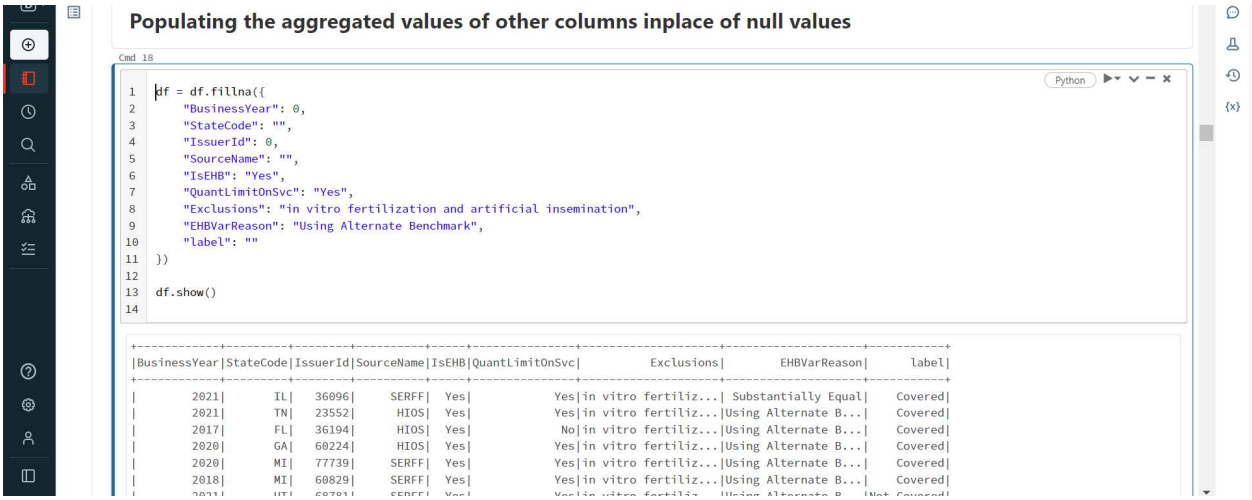
```
df.agg({'IsEHB':'max','QuantLimitOnSvc':'max','Exclusions':'max','EHBVarReason':'max'}).collect()
```



# Step 10: Populating the aggregated values of other columns in place of null values

Once, we get the maximum values of the above columns, replace the null values with maximum values.

```
df = df.fillna({
    "BusinessYear": 0,
    "StateCode": "",
    "IssuerId": 0,
    "SourceName": "",
    "IsEHB": "Yes",
    "QuantLimitOnSvc": "Yes",
    "Exclusions": "in vitro fertilization and artificial insemination",
    "EHBVarReason": "Using Alternate Benchmark",
    "label": ""
})
df.show()
```



# Step 11: Convert the label into 0 and 1 for classification modelling and prediction.

In the following step, we are categorizing the values of label column into 0 and 1.

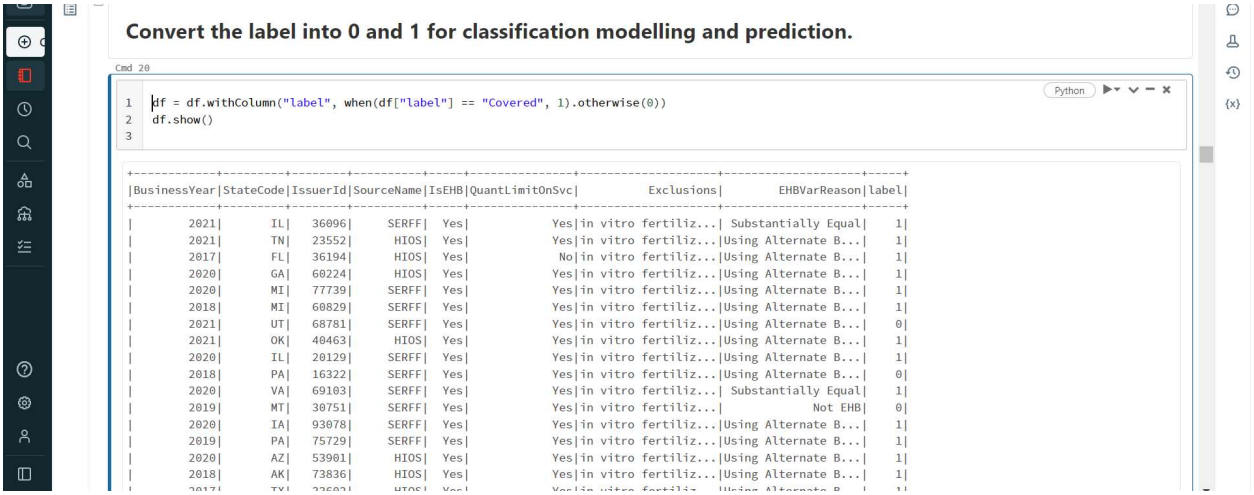
0 = Covered

1 = Not Covered

Copy the below code and paste it in a new cell.

```
df = df.withColumn("label", when(df["label"] == "Covered", 1).otherwise(0))
```

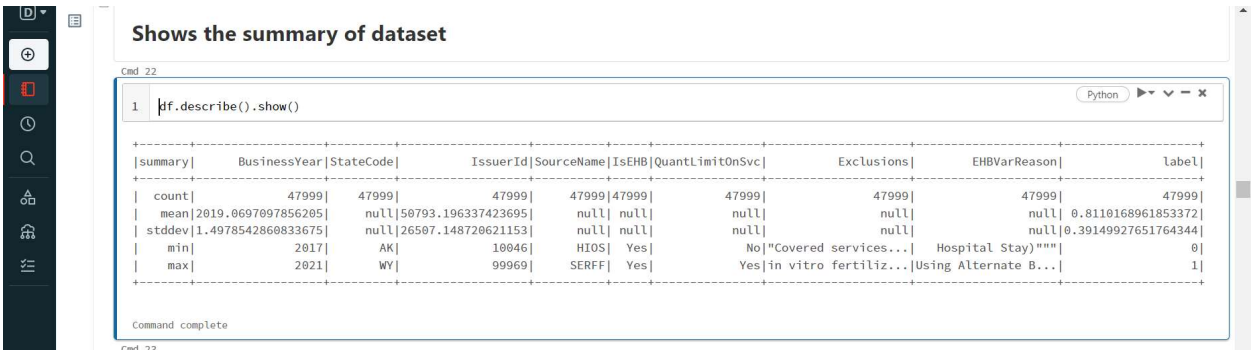
```
df.show()
```




# Step 12: Shows the summary of dataset

In this step, we are observing the summary of all the columns in df using the below code.

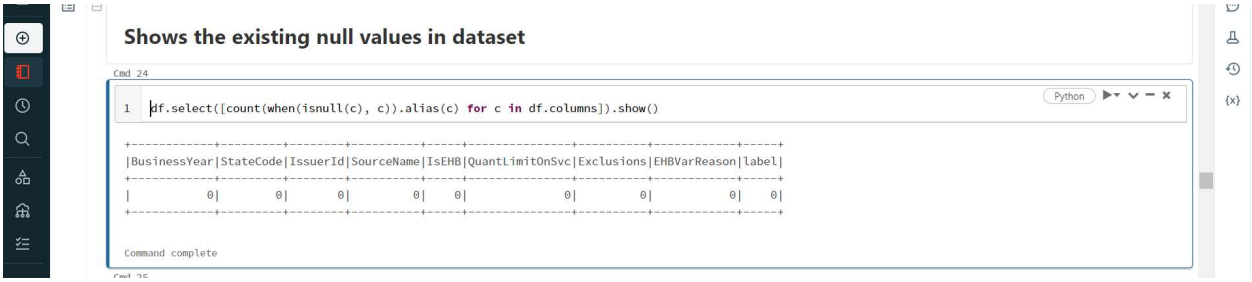
```
df.describe().show()
```



# Step 13: Shows the existing null values in dataset

Here, we are cross-verifying whether the existing dataset has any more null values through the below code.

```
df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()
```




# Step 14: Split the Data for training & testing

Now, we need to randomly split the data into 70% for training and 30% for testing purpose for modelling and prediction using the below code.

Prediction column: `IsCovered`

Label: The column used in training models as a reference to guide the learning process. It contains the known or actual values for the target variable.

True Label: The column used in testing models to evaluate their performance. It contains the actual values for the target variable that are compared to the predicted values generated by the model.

```
splits = df.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1].withColumnRenamed("label", "trueLabel")
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, " Testing Rows:", test_rows)
```

# Step15: Defining the Pipeline

- A predictive model often requires multiple stages of feature preparation, including distinguishing between continuous and categorical features and normalizing numeric features.
- Since our dataset is classification dataset with many categorical values (non-numeric values) we need convert categorical values to numerical values.
- To accomplish this, a pipeline can be used, which consists of a series of transformer and estimator stages. In this case, a pipeline with seven stages will be created.
- Here, we are defining the pipeline using StringIndexer, VectorAssembler, VectorIndexer, MinMaxscaler etc.
    1. A StringIndexer estimator that converts string values to indexes for categorical features
    2. A VectorAssembler that combines categorical features into a single vector
    3. A VectorIndexer that creates indexes for a vector of categorical features
    4. A VectorAssembler that creates a vector of continuous numeric features
    5. A MinMaxScaler that normalizes continuous numeric features
    6. A VectorAssembler that creates a vector of categorical and continuous features

Using the below code, we can prepare the pipeline.

```
strIdx_SC = StringIndexer(inputCol = "StateCode", outputCol =
"SC",handleInvalid='keep')

strIdx_SN = StringIndexer(inputCol = "SourceName", outputCol =
"SN",handleInvalid='keep')

strIdx_EHB = StringIndexer(inputCol = "IsEHB", outputCol =
"EHB",handleInvalid='keep')

strIdx_QL = StringIndexer(inputCol = "QuantLimitOnSvc", outputCol =
"QL",handleInvalid='keep')

strIdx_EX = StringIndexer(inputCol = "Exclusions", outputCol =
"EX",handleInvalid='keep')

strIdx_EHBVR = StringIndexer(inputCol = "EHBVarReason", outputCol =
"EHBVR",handleInvalid='keep')


# The following columns are categorical number such as ID so that it
should be Category features

catVect = VectorAssembler(inputCols = ["SC", "BusinessYear",
"IssuerId", "SN", "EHB","QL","EHBVR"], outputCol="catFeatures")

catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol =
"idxCatFeatures", handleInvalid="skip")
```

Cmd 28

```
strIdx_SC = StringIndexer(inputCol = "StateCode", outputCol = "SC",handleInvalid='keep')
strIdx_SN = StringIndexer(inputCol = "SourceName", outputCol = "SN",handleInvalid='keep')
strIdx_EHB = StringIndexer(inputCol = "IsEHB", outputCol = "EHB",handleInvalid='keep')
strIdx_QL = StringIndexer(inputCol = "QuantLimitOnSvc", outputCol = "QL",handleInvalid='keep')
strIdx_EX = StringIndexer(inputCol = "Exclusions", outputCol = "EX",handleInvalid='keep')
strIdx_EHBVR = StringIndexer(inputCol = "EHBVarReason", outputCol = "EHBVR",handleInvalid='keep')


# the following columns are categorical number such as ID so that it should be Category features
catVect = VectorAssembler(inputCols = ["SC", "BusinessYear", "IssuerId", "SN", "EHB","QL","EHBVR"], outputCol="catFeatures")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol = "idxCatFeatures", handleInvalid="skip")

```

```
# Cat feature vector is normalized

minMax = MinMaxScaler(inputCol = catIdx.getOutputCol(),
outputCol="normFeatures")


featVect = VectorAssembler(inputCols=["normFeatures"],
outputCol="features")
```

Cmd 31

```
# cat feature vector is normalized

minMax = MinMaxScaler(inputCol = catIdx.getOutputCol(), outputCol="normFeatures")

featVect = VectorAssembler(inputCols=["normFeatures"], outputCol="features")
```

# Step 16: Prediction Modeling using 6 different ML Algorithms

Here we are using six different ML Algorithms to ***predict whether health insurance plans would cover dental services or not.***

ML Algorithms used in our analysis would compete and would be compared with each other based on their performance through several metrics such as Accuracy, Precision and Recall and many more etc.

Algorithms are as follows:

1. Logistic Regression (LR)
2. Decision Tree (DT)
3. Random Forest (RT)
4. Factorization Machine (FM)
5. Gradient Boost (GBT)
6. Support Vector Machine (SVM)

To observe this, copy the below code and paste it in the next cell as shown from the screenshot below.

```
classification_models=["Logistic Regression (LR)","Decision Tree
(DT)","Random Forest (RT)","Factorization Machine (FM)","Gradient
Boost (GBT)","Support Vector Machine (SVM)"]


#Creating diff classification algos for testing accuracy, computing
time, precision, recall, ROC, PR

cls_mod=[]


cls_mod.insert(0,LogisticRegression(labelCol="label",featuresCol="feat
ures",maxIter=10,regParam=0.3,threshold=0.35))

cls_mod.insert(1,DecisionTreeClassifier(labelCol="label",
featuresCol="features",seed=42))

cls_mod.insert(2,RandomForestClassifier(labelCol='label',
featuresCol='features',seed=42))

cls_mod.insert(3,FMClassifier(labelCol='label',
featuresCol='features', seed=42))

cls_mod.insert(4,GBTClassifier(labelCol='label',
featuresCol='features', seed=42))

cls_mod.insert(5,LinearSVC(labelCol='label', featuresCol='features'))
```

```
6
7  classification_models=["Logistic Regression (LR)","Decision Tree (DT)","Random Forest (RT)","Factorization Machine (FM)","Gradiest
   Boost (GBT)","Support Vector Machine (SVM)"]
8
9  #creating diff clasf algos for testing accuracy,computing time, precision, recall, ROC, PR
10 cls_mod=[]
11
12 cls_mod.insert(0,LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regParam=0.3,threshold=0.35))
13 cls_mod.insert(1,DecisionTreeClassifier(labelCol="label", featuresCol="features",seed=42))
14 cls_mod.insert(2,RandomForestClassifier(labelCol='label', featuresCol='features',seed=42))
15 cls_mod.insert(3,FMClassifier(labelCol='label', featuresCol='features', seed=42))
16 cls_mod.insert(4,GBTClassifier(labelCol='label', featuresCol='features', seed=42))
17 cls_mod.insert(5,LinearSVC(labelCol='label', featuresCol='features'))
```

Command complete

```
# Define list of models made from Train Validation Split or Cross
Validation

model = []

pipeline = []
```

```
# Pipeline process the series of transformation above, which is
another transformation


for i in range(0,6):

    pipeline.insert(i,Pipeline(stages=[strIdx_SC,strIdx_SN,strIdx_EHB,
    strIdx_QL, strIdx_EX, strIdx_EHBVR, catVect, catIdx,minMax, featVect,
    cls_mod[i]]))
```


## Step 17: Shows the feature extraction count of categorical features

Here we are counting the distinct features extracted for all the columns using the below code.

```
# Transform the input data using the fitted string indexers

data_transformed = df

data_transformed = strIdx_SC_model.transform(data_transformed)

data_transformed = strIdx_SN_model.transform(data_transformed)

data_transformed = strIdx_EHB_model.transform(data_transformed)

data_transformed = strIdx_QL_model.transform(data_transformed)

data_transformed = strIdx_EX_model.transform(data_transformed)

data_transformed = strIdx_EHBVR_model.transform(data_transformed)

# Count the number of distinct values in each output column

distinct_counts = {

    "StateCode":
data_transformed.select(countDistinct("SC")).collect()[0][0],

    "SourceName":
data_transformed.select(countDistinct("SN")).collect()[0][0],
```

```
    "IsEHB":
data_transformed.select(countDistinct("EHB")).collect()[0][0],
    "QuantLimitOnSvc":
data_transformed.select(countDistinct("QL")).collect()[0][0],
    "Exclusions":
data_transformed.select(countDistinct("EX")).collect()[0][0],
    "EHBVarReason":
data_transformed.select(countDistinct("EHBVR")).collect()[0][0]}
print(distinct_counts)
```

Cmd 30

```python
# Fit the string indexers on the input data
strIdx_SC_model = strIdx_SC.fit(df)
strIdx_SN_model = strIdx_SN.fit(df)
strIdx_EHB_model = strIdx_EHB.fit(df)
strIdx_QL_model = strIdx_QL.fit(df)
strIdx_EX_model = strIdx_EX.fit(df)
strIdx_EHBVR_model = strIdx_EHBVR.fit(df)

# Transform the input data using the fitted string indexers
data_transformed = df
data_transformed = strIdx_SC_model.transform(data_transformed)
data_transformed = strIdx_SN_model.transform(data_transformed)
data_transformed = strIdx_EHB_model.transform(data_transformed)
data_transformed = strIdx_QL_model.transform(data_transformed)
data_transformed = strIdx_EX_model.transform(data_transformed)
data_transformed = strIdx_EHBVR_model.transform(data_transformed)

# Count the number of distinct values in each output column
distinct_counts = {
    "StateCode": data_transformed.select(countDistinct("SC")).collect()[0][0],
    "SourceName": data_transformed.select(countDistinct("SN")).collect()[0][0],
    "IsEHB": data_transformed.select(countDistinct("EHB")).collect()[0][0],
    "QuantLimitOnSvc": data_transformed.select(countDistinct("QL")).collect()[0][0],
    "Exclusions": data_transformed.select(countDistinct("EX")).collect()[0][0],
    "EHBVarReason": data_transformed.select(countDistinct("EHBVR")).collect()[0][0]
}

print(distinct_counts)
```

```
{'StateCode': 39, 'SourceName': 3, 'IsEHB': 1, 'QuantLimitOnSvc': 2, 'Exclusions': 576, 'EHBVarReason': 19}
Command complete
```

# Step 18: Tune hyperparameters using ParamGrid

1. Here we are ***tuning the Hyper parameters using Parameter Grid*** that are used in training the dataset using model for corresponding 6 different ML Algorithms used for our analysis.

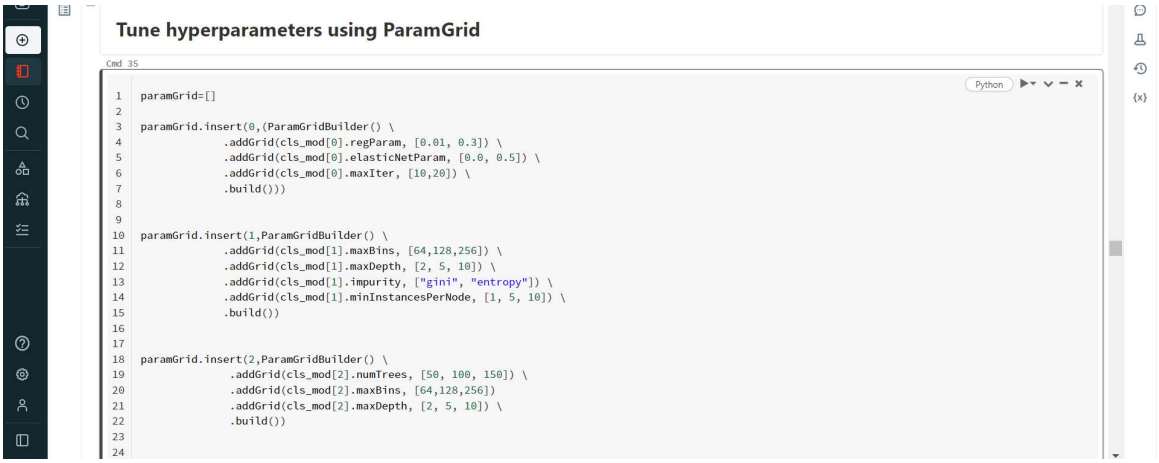
1. Logistic Regression (LR)
2. Decision Tree (DT)
3. Random Forest (RT)
4. Factorization Machine (FM)

5. Gradient Boost (GBT)
6. Support Vector Machine (SVM)

Copy the below code and run in the next cell.

```
# Fit the string indexers on the input data
strIdx_SC_model = strIdx_SC.fit(df)
strIdx_SN_model = strIdx_SN.fit(df)
strIdx_EHB_model = strIdx_EHB.fit(df)
strIdx_QL_model = strIdx_QL.fit(df)
strIdx_EX_model = strIdx_EX.fit(df)
strIdx_EHBVR_model = strIdx_EHBVR.fit(df)
paramGrid=[]
paramGrid.insert(0,(ParamGridBuilder() \
             .addGrid(cls_mod[0].regParam, [0.01, 0.3]) \
             .addGrid(cls_mod[0].elasticNetParam, [0.0, 0.5]) \
             .addGrid(cls_mod[0].maxIter, [10,20]) \
             .build()))
paramGrid.insert(1,ParamGridBuilder() \
             .addGrid(cls_mod[1].maxBins, [64,128,256]) \
             .addGrid(cls_mod[1].maxDepth, [2, 5, 10]) \
             .addGrid(cls_mod[1].impurity, ["gini", "entropy"]) \
             .addGrid(cls_mod[1].minInstancesPerNode, [1, 5, 10]) \
             .build())
paramGrid.insert(2,ParamGridBuilder() \
              .addGrid(cls_mod[2].numTrees, [50, 100, 150]) \
              .addGrid(cls_mod[2].maxBins, [64,128,256])
              .addGrid(cls_mod[2].maxDepth, [2, 5, 10]) \
              .build())
paramGrid.insert(3,ParamGridBuilder()\
.addGrid(cls_mod[3].regParam, [0.01, 0.1]) \
```

```
.addGrid(cls_mod[3].stepSize, [0.1,1])\
.addGrid(cls_mod[3].factorSize, [2,4])\
.build())
paramGrid.insert(4,ParamGridBuilder()\
.addGrid(cls_mod[4].maxDepth, [2, 5])\
.addGrid(cls_mod[4].maxIter, [10, 20])\
.addGrid(cls_mod[4].minInfoGain, [0.0])\
.build())
paramGrid.insert(5,ParamGridBuilder() \
            .addGrid(cls_mod[5].regParam, [0.01, 0.5]) \
            .addGrid(cls_mod[5].maxIter, [1, 5]) \
            .addGrid(cls_mod[5].tol, [1e-4, 1e-3]) \
            .addGrid(cls_mod[5].fitIntercept, [True, False]) \
            .addGrid(cls_mod[5].standardization, [True, False]) \
            .build())
```



```
paramGrid.insert(3,ParamGridBuilder()\
.addGrid(cls_mod[3].regParam, [0.01, 0.1]) \
.addGrid(cls_mod[3].stepSize, [0.1,1])\
.addGrid(cls_mod[3].factorSize, [2,4])\
.build())


paramGrid.insert(4,ParamGridBuilder()\
.addGrid(cls_mod[4].maxDepth, [2, 5])\
.addGrid(cls_mod[4].maxIter, [10, 20])\
.addGrid(cls_mod[4].minInfoGain, [0.0])\
.build())


paramGrid.insert(5,ParamGridBuilder() \
            .addGrid(cls_mod[5].regParam, [0.01, 0.5]) \
            .addGrid(cls_mod[5].maxIter, [1, 5]) \
            .addGrid(cls_mod[5].tol, [1e-4, 1e-3]) \
            .addGrid(cls_mod[5].fitIntercept, [True, False]) \
            .addGrid(cls_mod[5].standardization, [True, False]) \
            .build())
```

Command complete

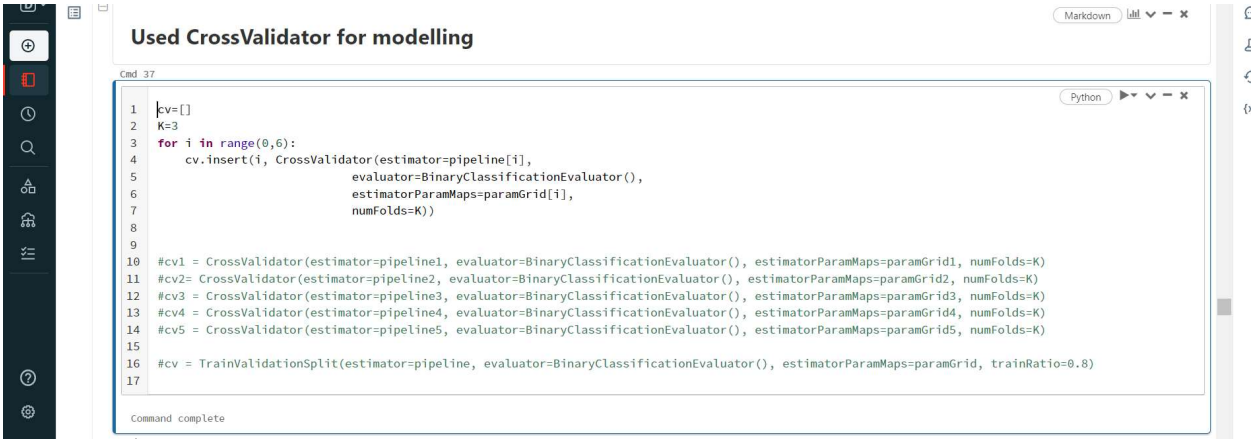
# Step 19: Using CrossValidator for modelling

To assess these above 6 algorithms, we are using Cross Validator model for training and finding the prediction column based on the features extracted and comparing the values with the actual labels or true label column of dataset i.e., `IsCovered`.

In this modelling, we are passing the above `Pipeline, ParamGrid`, and evaluator as `'Binary Classification Evaluator'` to evaluate metrics such as `Precision and Recall`

Use the below code and run it in the next cell.

```
cv=[]
K=3
for i in range(0,6):
    cv.insert(i, CrossValidator(estimator=pipeline[i],
                                evaluator=BinaryClassificationEvaluator(),
                                estimatorParamMaps=paramGrid[i],
                                numFolds=K))
```

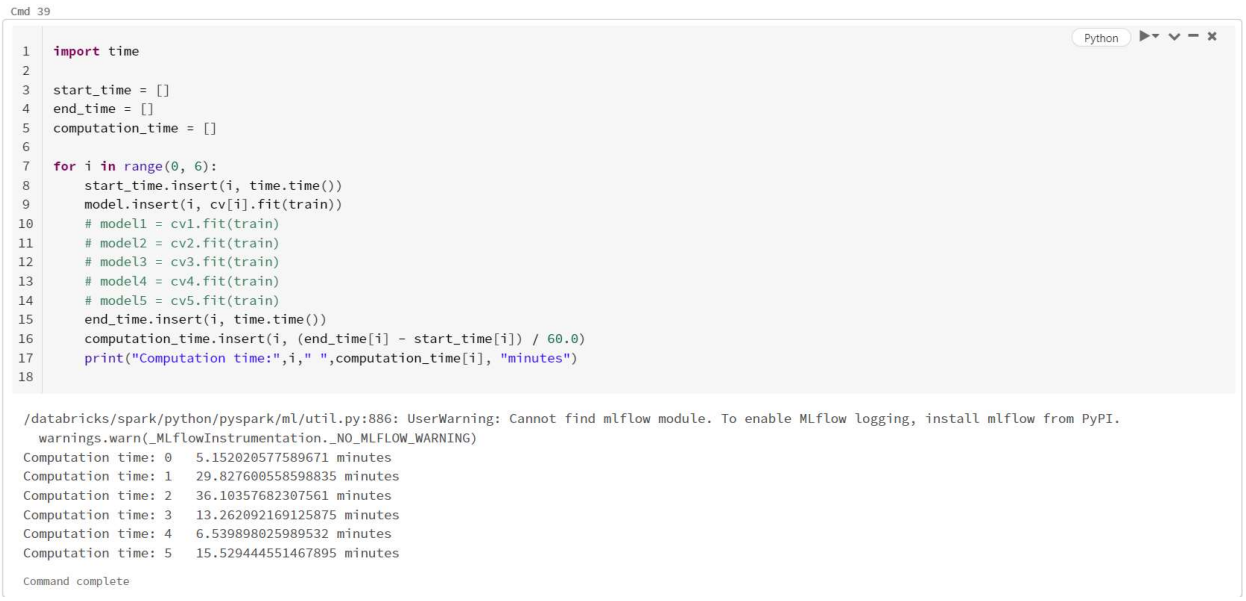



# Step 20: Calculating the computing time required to build a model

In this step, we are calculating the computing time(time taken by dataset to get trained for prediction through models) and we are using the below code for it.

Run the below code in the next cell and observe the computational values.

```
import time
start_time = []
end_time = []
computation_time = []
for i in range(0, 6):
    start_time.insert(i, time.time())
    model.insert(i, cv[i].fit(train))
    # model1 = cv1.fit(train)
    # model2 = cv2.fit(train)
    # model3 = cv3.fit(train)
    # model4 = cv4.fit(train)
    # model5 = cv5.fit(train)
    end_time.insert(i, time.time())
    computation_time.insert(i, (end_time[i] - start_time[i]) / 60.0)
    print("Computation time:",i," ",computation_time[i], "minutes")
```




# Step 21: Test the Pipeline Model using Test Dataset(30%)

The model produced by the pipeline is a transformer that will apply all the stages in the pipeline to a specified DataFrame and apply the trained model to generate predictions.

In this case, you will transform the test DataFrame using the pipeline to generate label predictions.

Here, the model which got trained using training dataset i.e., 70% of actual dataset, the same ***model is used for verifying the prediction using the Test dataset i.e., 30% of actual dataset.***

Run the below code and observe the results in the next line.

```
prediction =[]

predicted =[]

for i in range(0,6):

    prediction.insert(i,model[i].transform(test))

    prediction[i].show()

    predicted.insert(i,prediction[i].select("features",
"prediction","trueLabel"))

    predicted[i].show()
```

The resulting DataFrame is produced by applying all the transformations in the pipline to the test data. The **prediction** column contains the predicted value for the label, and the **trueLabel** column contains the actual known value from the testing data.

Cmd 40

## Test the Pipeline Model

The model produced by the pipeline is a transformer that will apply all of the stages in the pipeline to a specified DataFrame and apply the trained model to generate predictions. In this case, you will transform the **test** DataFrame using the pipeline to generate label predictions.

Cmd 41

```
1  prediction =[]
2  predicted =[]
3  for i in range(0,6):
4      prediction.insert(i,model[i].transform(test))
5      prediction[i].show()
6      predicted.insert(i,prediction[i].select("features", "prediction","trueLabel"))
7      predicted[i].show()
8
9
10
```

Results are as follows:

1. Prediction on testing dataset with all the columns

```
+-----------+---------+--------+----------+-----+--------------+------------------+--------------------+---------+----+---+---+---+-----+-----------
-------+------------------+--------------------+--------------------+--------------------+--------------------+----------+
|BusinessYear|StateCode|IssuerId|SourceName|IsEHB|QuantLimitOnSvc|          Exclusions|        EHBVarReason|trueLabel|  SC| SN|EHB| QL|EHBVR|         cat
Features|       idxCatFeatures|        normFeatures|            features|       rawPrediction|         probability|prediction|
+-----------+---------+--------+----------+-----+--------------+------------------+--------------------+---------+----+---+---+---+-----+-----------
-------+------------------+--------------------+--------------------+--------------------+--------------------+----------+
|        2017|       AK|   73836|      HIOS|  Yes|           Yes|in vitro fertiliz...| Substantially Equal|        1|38.0|0.0|0.0|0.0|  1.0|[38.0,2017.
0,7383...|[38.0,0.0,73836.0...|[1.0,0.0,0.709384...|[1.0,0.0,0.709384...|[-1.5173015736635...|[0.17985922014973...|       1.0|
|        2017|       AK|   73836|      HIOS|  Yes|           Yes|in vitro fertiliz...|Using Alternate B...|        1|38.0|0.0|0.0|0.0|  0.0|(7,[0,1,2],
[38.0,...|(7,[0,1,2],[38.0,...|(7,[0,2,4],[1.0,0...|(7,[0,2,4],[1.0,0...|[-1.7703689247076...|[0.14549645554837...|       1.0|
|        2017|       AL|   46944|      HIOS|  Yes|            No|in vitro fertiliz...|Using Alternate B...|        1|34.0|0.0|0.0|1.0|  0.0|[34.0,2017.
0,4694...|[34.0,0.0,46944.0...|[0.89473684210526...|[0.89473684210526...|[-3.5394920353324...|[0.02820920969756...|       1.0|
|        2017|       AL|   46944|      HIOS|  Yes|            No|in vitro fertiliz...|Using Alternate B...|        1|34.0|0.0|0.0|1.0|  0.0|[34.0,2017.
0,4694...|[34.0,0.0,46944.0...|[0.89473684210526...|[0.89473684210526...|[-3.5394920353324...|[0.02820920969756...|       1.0|
|        2017|       AL|   46944|       OPM|  Yes|            No|in vitro fertiliz...|Using Alternate B...|        1|34.0|2.0|0.0|1.0|  0.0|[34.0,2017.
0,4694...|[34.0,0.0,46944.0...|[0.89473684210526...|[0.89473684210526...|[-3.5118450073116...|[0.02897707697693...|       1.0|
|        2017|       AL|   46944|       OPM|  Yes|            No|in vitro fertiliz...|Using Alternate B...|        1|34.0|2.0|0.0|1.0|  0.0|[34.0,2017.
0,4694...|[34.0,0.0,46944.0...|[0.89473684210526...|[0.89473684210526...|[-3.5118450073116...|[0.02897707697693...|       1.0|
|        2017|       AL|   46944|       OPM|  Yes|           Yes|in vitro fertiliz...|Using Alternate B...|        0|34.0|2.0|0.0|0.0|  0.0|[34.0,2017.
0,4694...|[34.0,0.0,46944.0...|[0.89473684210526...|[0.89473684210526...|[-1.7081538032560...|[0.15340332932361...|       1.0|
|        2017|       AR|   37903|     SERFF|  Yes|           Yes|in vitro fertiliz...|Additional EHB Be...|        1|25.0|1.0|0.0|0.0|  2.0|[25.0,2017.
```

2. Prediction on testing dataset having (features, prediction, trueLabel)

```
+--------------------+----------+---------+
|            features|prediction|trueLabel|
+--------------------+----------+---------+
|[1.0,0.0,0.709384...|       1.0|        1|
|(7,[0,2,4],[1.0,0...|       1.0|        1|
|[0.89473684210526...|       1.0|        1|
|[0.89473684210526...|       1.0|        1|
|[0.89473684210526...|       1.0|        1|
|[0.89473684210526...|       1.0|        1|
|[0.89473684210526...|       1.0|        0|
|[0.65789473684210...|       1.0|        1|
|[0.65789473684210...|       1.0|        1|
|[0.65789473684210...|       1.0|        1|
|[0.65789473684210...|       1.0|        1|
|[0.65789473684210...|       1.0|        1|
|[0.65789473684210...|       1.0|        1|
|[0.65789473684210...|       1.0|        0|
|[0.65789473684210...|       1.0|        0|
|[0.21052631578947...|       1.0|        1|
|(7,[0,2,4],[0.210...|       1.0|        1|
|(7,[0,2,4],[0.210...|       1.0|        1|
```

# Step 22: Compute Confusion Matrix Metrics

Classifiers are typically evaluated by creating a *confusion matrix*, which indicates the number of:

- True Positives

- True Negatives

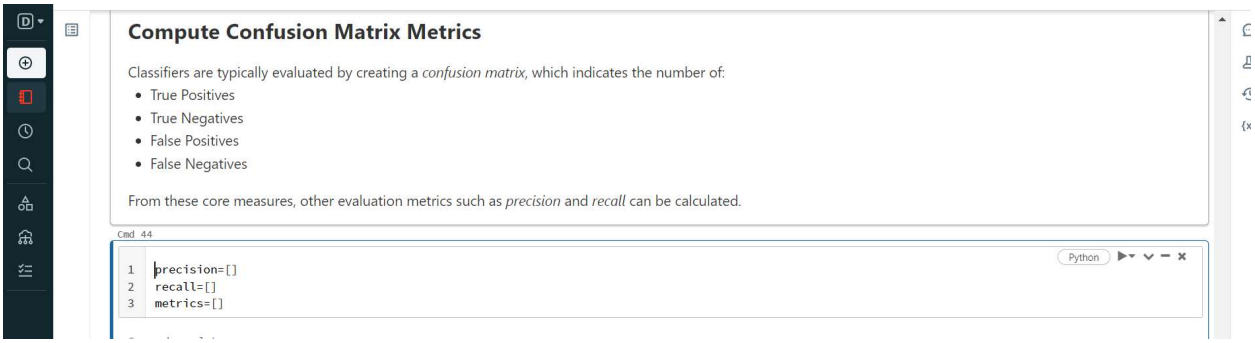
- False Positives

- False Negatives

From these core measures, other evaluation metrics such as *precision* and *recall* can be calculated.

These metrics are being used to evaluate the performance of the 6 different models.

```
precision=[]

recall=[]

metrics=[]
```




```
for i in range(0,6):

    tp = float(predicted[i].filter("prediction== 1.0 AND truelabel == 1").count())

    fp = float(predicted[i].filter("prediction== 1.0 AND truelabel == 0").count())

    tn = float(predicted[i].filter("prediction== 0.0 AND truelabel == 0").count())

    fn = float(predicted[i].filter("prediction==0.0 AND truelabel == 1").count())

    precision.insert(i,tp / (tp + fp))

    recall.insert(i,tp / (tp + fn))

    metrics.insert(i, spark.createDataFrame([
```

```
    ("TP", tp),
    ("FP", fp),
    ("TN", tn),
    ("FN", fn),
    ("Precision", tp / (tp + fp)),
    ("Recall", tp / (tp + fn))],["metric", "value"]))
    metrics[i].show()
```

Cmd 45

```
for i in range(0,6):
    tp = float(predicted[i].filter("prediction== 1.0 AND truelabel == 1").count())
    fp = float(predicted[i].filter("prediction== 1.0 AND truelabel == 0").count())
    tn = float(predicted[i].filter("prediction== 0.0 AND truelabel == 0").count())
    fn = float(predicted[i].filter("prediction==0.0 AND truelabel == 1").count())
    precision.insert(i,tp / (tp + fp))
    recall.insert(i,tp / (tp + fn))
    metrics.insert(i, spark.createDataFrame([
    ("TP", tp),
    ("FP", fp),
    ("TN", tn),
    ("FN", fn),
    ("Precision", tp / (tp + fp)),
    ("Recall", tp / (tp + fn))],["metric", "value"]))
    metrics[i].show()
```

Run the above two codes in the next line and observe 6 different results for 6 different models.

```
+---------+------------------+
|   metric|             value|
+---------+------------------+
|       TP|           11699.0|
|       FP|            2713.0|
|       TN|               0.0|
|       FN|               3.0|
|Precision|0.8117540938107133|
|   Recall|0.9997436335669116|
+---------+------------------+

+---------+------------------+
|   metric|             value|
+---------+------------------+
|       TP|           11626.0|
|       FP|            1785.0|
|       TN|             928.0|
|       FN|              76.0|
|Precision|0.8669003057191857|
|   Recall|0.9935053836950949|
+---------+------------------+
```

Command complete

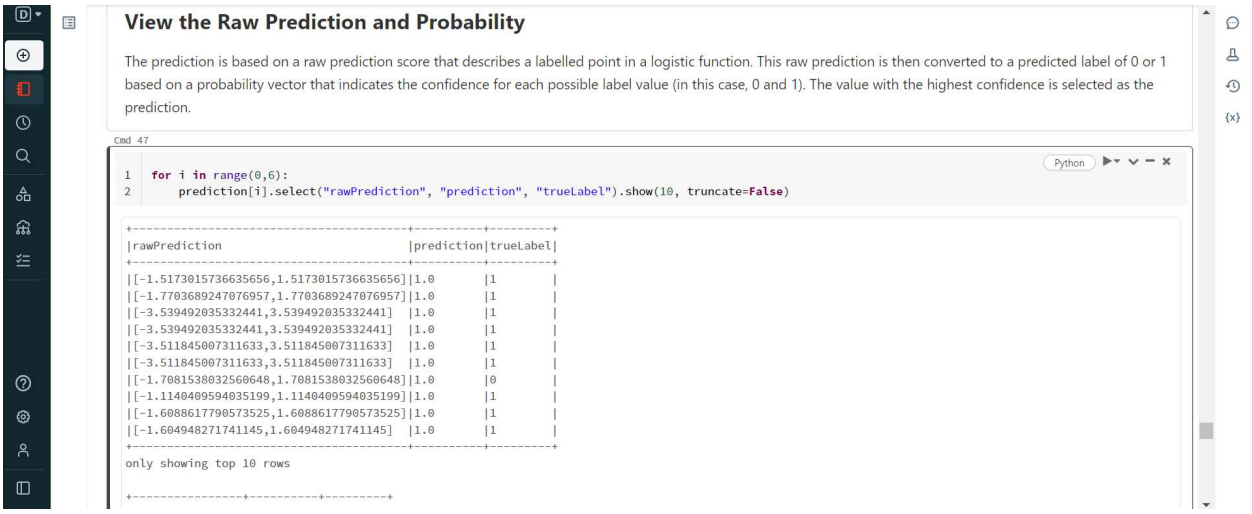
# Step 23: View the Raw Prediction and Probability

The prediction is based on a raw prediction score that describes a labelled point in a logistic function. This raw prediction is then converted to a prediction label of 0 or 1 based on a probability vector that indicates the confidence for each label value (in this case, 0 and 1).

The value with the highest confidence is selected as the prediction.

Run the below code in the next line and observe the equivalent results.

```
for i in range(0,6):
    prediction[i].select("rawPrediction", "prediction", "trueLabel").show(10, truncate=False)
```




# Step24: Calculating metrics such as ROC, PR, Accuracy, F1_score, Precision, Recall

Here, we are calculating different metrics as shown below.

**ROC:** ROC is a graphical representation of the performance of a classification model. It plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. In simple terms, it helps assess the trade-off between sensitivity (true positive rate) and specificity (true negative rate) of the model.

**PR:** PR is another evaluation metric for classification models. It plots the precision (positive predictive value) against the recall (true positive rate) at different classification thresholds. It is particularly useful when dealing with imbalanced datasets, where the focus is on correctly identifying positive cases.

**Precision:** Precision measures the proportion of correctly predicted positive instances out of the total instances predicted as positive. In other words, it quantifies the accuracy of positive predictions.

Precision = TP / (TP + FP)

**Recall:** Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances out of all actual positive instances. It quantifies the model's ability to identify positive cases.

Recall = TP / (TP + FN)

The above 4 metrics are evaluated using BinaryClassification Evaluator

**Accuracy:** Accuracy is a widely used evaluation metric that measures the overall correctness of the model's predictions. It calculates the ratio of correctly classified instances (both true positives and true negatives) to the total number of instances. However, accuracy may not be suitable for imbalanced datasets, as it can be misleading when the classes have different proportions.

**F1 Score:** The F1 score is a metric that combines precision and recall into a single value. It is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. The F1 score is commonly used when there is an uneven class distribution in the dataset.

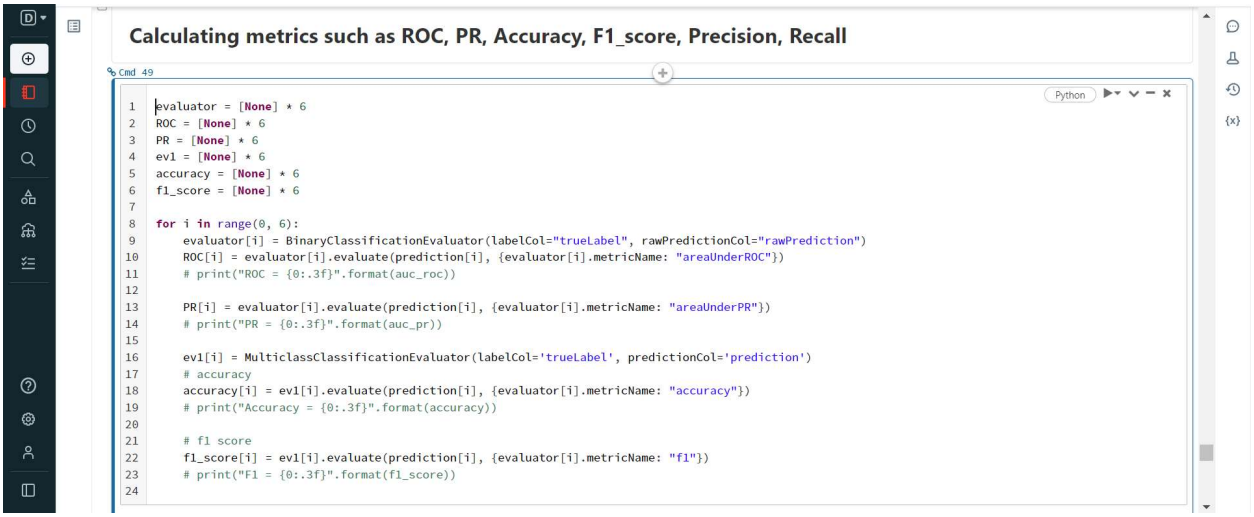
F1 Score = 2 * ((Precision * Recall) / (Precision + Recall))

The above 2 metrics are evaluated using MultiClassification Evaluator

This is accomplished by using the below code. Run it in the next cell and observe the following result.

```
evaluator = [None] * 6

ROC = [None] * 6

PR = [None] * 6

ev1 = [None] * 6

accuracy = [None] * 6

f1_score = [None] * 6

for i in range(0, 6):
```

```
    evaluator[i] = BinaryClassificationEvaluator(labelCol="trueLabel",
rawPredictionCol="rawPrediction")

    ROC[i] = evaluator[i].evaluate(prediction[i],
{evaluator[i].metricName: "areaUnderROC"})

    # print("ROC = {0:.3f}".format(auc_roc))

    PR[i] = evaluator[i].evaluate(prediction[i],
{evaluator[i].metricName: "areaUnderPR"})

    # print("PR = {0:.3f}".format(auc_pr))

    ev1[i] = MulticlassClassificationEvaluator(labelCol='trueLabel',
predictionCol='prediction')

    # accuracy

    accuracy[i] = ev1[i].evaluate(prediction[i],
{evaluator[i].metricName: "accuracy"})

    # print("Accuracy = {0:.3f}".format(accuracy))

    # f1 score

    f1_score[i] = ev1[i].evaluate(prediction[i],
{evaluator[i].metricName: "f1"})

    # print("F1 = {0:.3f}".format(f1_score))
```



# Step 25: Comparing all metrics at one place

Here, we are comparing all the above metrics calculated for 6 different algorithms are observing which model performed well as per prediction statement. ***"Are health Insurance Plans covering adult dental services or not."***

```
import pandas as pd

results = {
    'Model': classification_models,
    'Computation Time (min)': computation_time,
    'ROC': ROC,
    'PR': PR,
    'Accuracy': accuracy,
    'F1 Score': f1_score,
    'Precision': precision,
    'Recall': recall
}
df_results = pd.DataFrame.from_dict(results)
df_results = df_results.set_index('Model').transpose()
print(df_results)
```

| Model | Logistic Regression (LR) | Decision Tree (DT) | \ |
|---|---|---|---|
| Computation Time (min) | 5.152021 | 29.827601 | |
| ROC | 0.620981 | 0.648268 | |
| PR | 0.879197 | 0.847665 | |
| Accuracy | 0.727362 | 0.845612 | |
| F1 Score | 0.727362 | 0.845612 | |
| Precision | 0.811754 | 0.866900 | |
| Recall | 0.999744 | 0.993505 | |

| Model | Random Forest (RT) | Factorization Machine (FM) | \ |
|---|---|---|---|
| Computation Time (min) | 36.103577 | 13.262092 | |
| ROC | 0.846004 | 0.633083 | |
| PR | 0.958525 | 0.882931 | |
| Accuracy | 0.834552 | 0.727156 | |
| F1 Score | 0.834552 | 0.727156 | |
| Precision | 0.858991 | 0.811676 | |
| Recall | 0.998462 | 0.999231 | |

| Model | Gradiest Boost (GBT) | Support Vector Machine (SVM) |
|---|---|---|
| Computation Time (min) | 6.539898 | 15.529445 |
| ROC | 0.828126 | 0.601684 |
| PR | 0.953801 | 0.870743 |
| Accuracy | 0.824062 | 0.727465 |
| F1 Score | 0.824062 | 0.727465 |
| Precision | 0.852945 | 0.811793 |
| Recall | 0.999744 | 1.000000 |

From the above tables, we come to conclusion that GBT is the best model with highest precision of 85% with least computation time 6.5 mins.

| **Models** | **Computation Time** | **Precision** | **Recall** |
|---|---|---|---|
| Logistic Regression (LR) | 5 mins | 0.81 | 0.99 |
| Decision Tree (DT) | 30 mins | 0.87 | 0.99 |
| Random Forrest (RF) | 36 mins | 0.86 | 0.99 |
| Factorization Machine (FM) | 13 mins | 0.81 | 0.99 |
| Gradient Boost (GBT) | 6.5 mins | 0.85 | 0.99 |
| Support Vector Machine (SVM) | 15.5 mins | 0.81 | 1.00 |

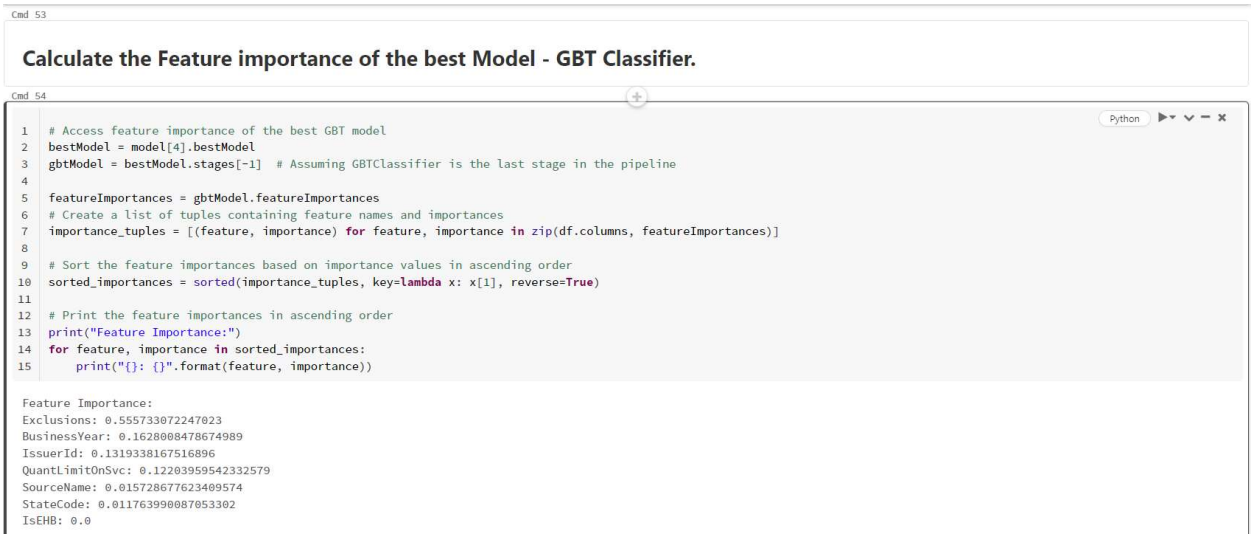
# Step 26: Calculating Feature Importance of the best model – GBT Classifier.

Now, we are ranking the importance of features of df columns and seeing which columns are impacting the most for predicting whether the dental services are covered or not in health insurance plans.

Run the below code in the next cell and observe the results.

```
# Access feature importance of the best GBT model
```

```
bestModel = model[4].bestModel

gbtModel = bestModel.stages[-1]  # Assuming GBTClassifier is the last stage in the pipeline


featureImportances = gbtModel.featureImportances

# Create a list of tuples containing feature names and importance's

importance_tuples = [(feature, importance) for feature, importance in zip(df.columns, featureImportances)]


# Sort the feature importance's based on importance values in ascending order

sorted_importances = sorted(importance_tuples, key=lambda x: x[1], reverse=True)


# Print the feature importance's in ascending order

print("Feature Importance:")

for feature, importance in sorted_importances:

    print("{}: {}".format(feature, importance))
```




From the above results, ***Exclusion is having the highest impact with 0.55*** compared to isEHB which is redundant column.

# PART -2

The objective is to perform the same analysis on the actual dataset i.e., benefit.csv with 5 million rows.

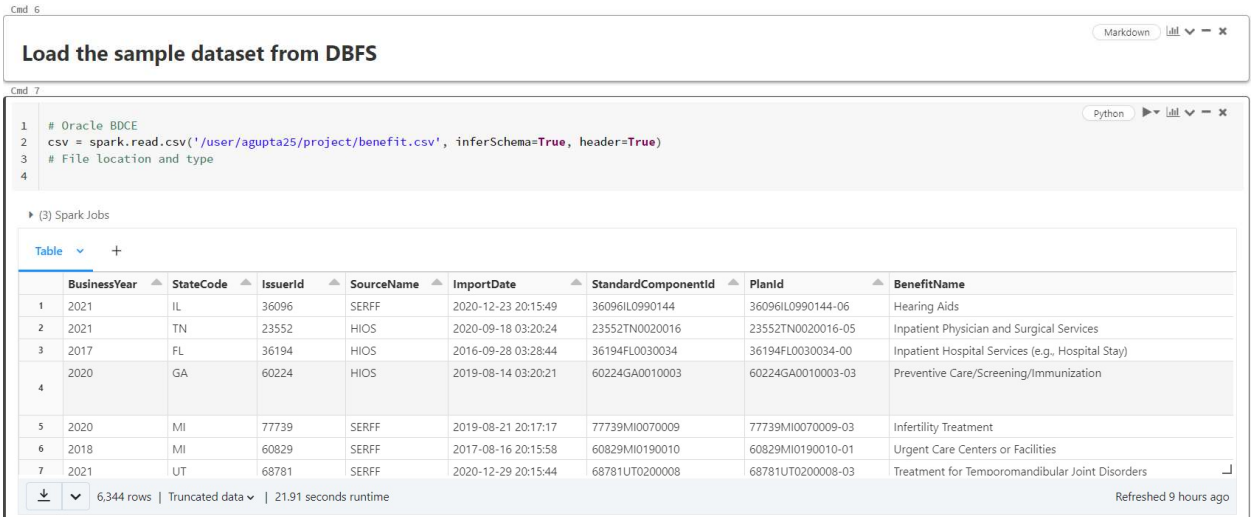
## Step 1: Download the source file in .py

Modify the Step 5 of Part1 , replace the code by below as show.

Replace the @username with your system name and file path as @your_file_path where benefits.csv is stored in your HDFS system.

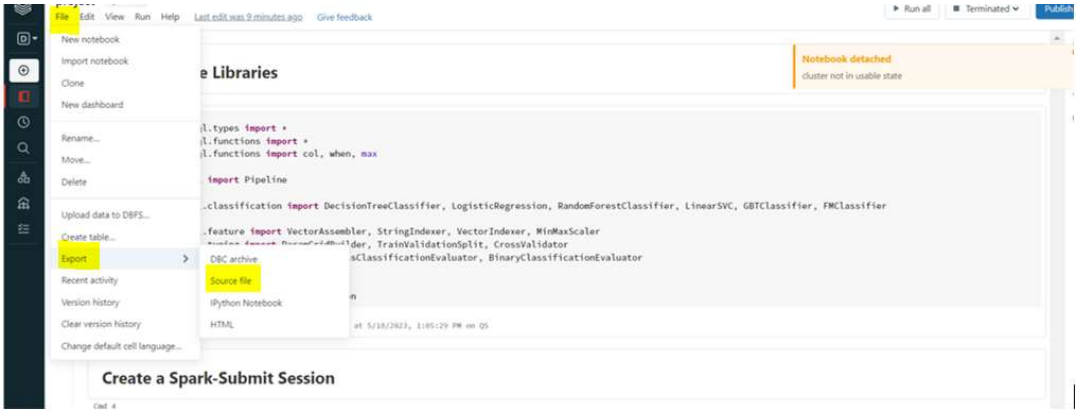
```
# Oracle BDCE

csv = spark.read.csv('/user/@username/@your_file_path/benefit.csv', inferSchema=True, header=True)

# File location and type
```

This step is performed to run this file in PySpark_CLI.




Now, download the file with .py extension as shown from the figure below. File >> Export >> Source File.

Save the file as `py_cli.py`




# Step 2: Upload the files in Linux server

## A. Upload the dataset in HDFS environment

To upload the files from the local system to linux, open a new terminal of Git Bash, and copy the below code and run it.

```
$ scp "@yourfilepath"\benefit.csv @username@ip_address
```

```
$ scp "C:\Users\agupta25\Desktop\5560 project\benefit.csv" agupta25@144.24.53.159:~/
```

```
AD+agupta25@STU-PF3034XS MINGW64 ~
$ scp "C:\Users\agupta25\Desktop\5560 project\benefit.csv" agupta25@144.24.53.15
9:~/
agupta25@144.24.53.159's password:
C:\Users\agupta25\Desktop\5560 project\benefi 100% 2003MB   1.3MB/s   25:15
```

The above code is used to upload the actual dataset to Linux and from there to HDFS.

If you want to view the list of files in your Linux server, open another new Git Bash Terminal and connect to your Linux server using the below code.

```
$ ssh @username@ip_address
```

```
Username password:
```

```
Ex: $ ssh agupta25@144.24.53.159
```

```
$ ssh agupta25@144.24.53.159
agupta25@144.24.53.159's password:
Last login: Sun May  7 00:46:21 2023 from 035-150-145-141.res.spectrum.com
```

```
$ ls
```

Now we need to ***move/copy the benefit.csv file*** from ***Linux to HDFS***, run the below code in the same Git Bash terminal after `ls` command.

1. Create a hdfs directory as ***'project'***
   ```
   $ hdfs dfs -mkdir project/
   ```
2. Move / Copy the ***benefit.csv file into project folder***
   ```
   $ hdfs dfs -put benefit.csv project/
   ```

```
-bash-4.2$ ls
airlines.csv  Classification_Pipeline_CV_Tune.py
benefit.csv   customers.csv
-bash-4.2$ hdfs dfs -put benefit.csv project/
```

3. Once we move the file into project directory / folder, we can view the files in it using the below code.

   `$ hdfs dfs -ls project/`

```
-bash-4.2$ hdfs dfs -ls project/
Found 5 items
-rw-r--r--   3 agupta25 hdfs 2100070159 2023-05-07 03:35 project/benefit.csv
```

## B. Upload the source file .py in Linux Server

Upload the downloaded file from databricks with .py with `py_cli.py` by using the below code.

`$ scp "@yourfilepath\py_cli_fin.py" @username@ip_address`

`Ex: $ scp "C:\Users\agupta25\Desktop\5560 project\py_cli_fin.py" agupta25@144.24.53.159:~/`

```
AD+agupta25@STU-PF3034XS MINGW64 ~
$ scp "C:\Users\agupta25\Desktop\5560 project\py_cli.py" agupta25@144.24.53.159:
~/
agupta25@144.24.53.159's password:
C:\Users\agupta25\Desktop\5560 project\py_cli 100%   13KB 123.1KB/s   00:00
```

Similarly, you can view this file in Linux server using the below code.

`$ ls`

```
-bash-4.2$ ls
airlines.csv                    project_CLI_sample.py
Classification_Pipeline_CV_Tune.py  python_clustering_db.py
customers.csv                   Python_Pipeline_databricks_sol.py
flights.csv                     python_recommendation_db.py
py_cli.py                       ratings.csv
lr.py                           svm.py
movies.csv                      tweets.csv
```

# Step 3: Run the file using Spark – Submit in PySpark CLI:

Once, the `py_cli.py` file gets uploaded on Linux server, to observe the results of the following file which are like data bricks metric evaluation, run the following code in next line.

`$ spark-submit py_cli.py`

```
-bash-4.2$ spark-submit py_cli.py
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/odh/1.1.5/spark/jars/slf4j-log4j12-1.7.30.jar!/o
SLF4J: Found binding in [jar:file:/usr/odh/1.1.5/hadoop/lib/slf4j-log4j12-1.7.25.jar!/o
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
23/05/20 16:42:41 INFO SparkContext: Running Spark version 3.0.2
23/05/20 16:42:41 INFO ResourceUtils: ==============================================
23/05/20 16:42:41 INFO ResourceUtils: Resources for spark.driver:

23/05/20 16:42:41 INFO ResourceUtils: ==============================================
23/05/20 16:42:41 INFO SparkContext: Submitted application: fm.py
23/05/20 16:42:41 INFO SecurityManager: Changing view acls to: agupta25
```

| summary | BusinessYear | StateCode | IssuerId | SourceName | IsEHB | QuantLimitOnSvc | ExclusionS | EHBVarReason | label |
|---|---|---|---|---|---|---|---|---|---|
| count | 5415910 | 5415910 | 5415910 | 5415910 | 5415910 | 5415910 | 5415910 | 5415910 | 5415910 |
| mean | 2019.0615619535774 | null | 50884.65046538809 | null | null | null | null | null | 0.8108378831996839 |
| stddev | 1.4964511295812055 | null | 26459.569440657353 | null | null | null | null | null | 0.39163738162766454 |
| min | 2017 | AK | 10046 | HIOS | Yes | No | "Covered services... | Hospital Stay)""" | 0 |
| max | 2021 | WY | 99969 | SERFF | Yes | Yes | in vitro fertiliz... | Using Alternate B... | 1 |

The above figure shows the description / summary of actual dataset with 5 million rows.

For results to appear, it would take about 1-2 hours, just wait for a while and you could observe the comparable results as shown below.

| Model | Gradiest Boost (GBT) |
|---|---|
| Computation Time (min) | 39.656804 |
| ROC | 0.825390 |
| PR | 0.953203 |
| Accuracy | 0.822529 |
| F1 Score | 0.822529 |
| Precision | 0.851675 |
| Recall | 0.999873 |

| Model | Logistic Regression (LR) |
|---|---|
| Computation Time (min) | 20.546078 |
| ROC | 0.623813 |
| PR | 0.879924 |
| Accuracy | 0.726101 |
| F1 Score | 0.726101 |
| Precision | 0.810844 |
| Recall | 0.999766 |

| Model | Factorization Machine (FM) |
|---|---|
| Computation Time (min) | 5.359796 |
| ROC | 0.629820 |
| PR | 0.881209 |
| Accuracy | 0.726084 |
| F1 Score | 0.726084 |
| Precision | 0.810869 |
| Recall | 0.999484 |

| Model | Support Vector Machine (SVM) |
|---|---|
| Computation Time (min) | 4.112850 |
| ROC | 0.595847 |
| PR | 0.870542 |
| Accuracy | 0.730997 |
| F1 Score | 0.730997 |
| Precision | 0.814332 |
| Recall | 1.000000 |

The best model is Gradient Boost with computation time - 40 minutes and accuracy with 85%

| **Models** | **Computation Time** | **Precision** | **Recall** |
|---|---|---|---|
| Logistic Regression (LR) | 20.5 mins | 0.81 | 0.99 |
| Gradient Boost (GBT) | 40 mins | 0.85 | 0.99 |
| Factorization Machine (FM) | 5 mins | 0.81 | 0.99 |
| Support Vector Machine (SVM) | 4 mins | 0.81 | 1.00 |

You could download the source files from 5560/codes at main · rahulbhogale/5560 · GitHub [6]

# References


1. Data sources - https://www.cms.gov/cciio/resources/data-resources/marketplace-puf

2. GitHub - https://github.com/rahulbhogale

3. References:

   [1] Kaggle : https://www.kaggle.com/datasets/hhs/health-insurance-marketplace

   [2] PySpark documentation:

   https://spark.apache.org/docs/latest/api/python/reference/pyspark.ml.html#classification

   [3] GitBash: https://gitforwindows.org/

   [4] Databricks Community Edition: https://community.cloud.databricks.com/login.html

   [5] Jupyter Notebooks: https://jupyter.org/

   [6] Source codes: 5560/codes at main · rahulbhogale/5560 · GitHub