



# 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](mailto:agupta25@calstatela.edu)), Rahul Bhogale ([rbhogal@calstatela.edu](mailto:rbhogal@calstatela.edu)),  
Priyanka Thota ([pthota2@calstatela.edu](mailto:pthota2@calstatela.edu)), Prathushkumar Dathuri ([pdathur@calstatela.edu](mailto: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

```
-bash-4.2$ hdfs version
Hadoop 3.1.2
```

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

```
-bash-4.2$ yarn node -list -all
23/05/07 23:17:08 INFO client.RMProxy: Connecting to ResourceManager at bigdaimn0.sub03291929060.trainingvcn.oraclevcn.com/10.1.0.38:8050
23/05/07 23:17:08 INFO client.AHSProxy: Connecting to Application History server at bigdaiun0.sub03291929060.trainingvcn.oraclevcn.com/10.1.0.149:10200
Total Nodes:3
      Node-Id          Node-State Node-Http-Address      Number-of-Running-Containers
bigdaiwn2.sub03291929060.trainingvcn.oraclevcn.com:45454      RUNNING bigdaiwn2.sub03291929060.trainingvcn.oraclevcn.com:8042      4
bigdaiwn1.sub03291929060.trainingvcn.oraclevcn.com:45454      RUNNING bigdaiwn1.sub03291929060.trainingvcn.oraclevcn.com:8042      2
bigdaiwn0.sub03291929060.trainingvcn.oraclevcn.com:45454      RUNNING bigdaiwn0.sub03291929060.trainingvcn.oraclevcn.com:8042      3
```

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

```
-bash-4.2$ hdfs dfs -df -h
Filesystem                                         Size   Used  Available  Use%
hdfs://bigdaimn0.sub03291929060.trainingvcn.oraclevcn.com:8020  390.7 G  344.4 G   44.0 G   88%
```

## 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]

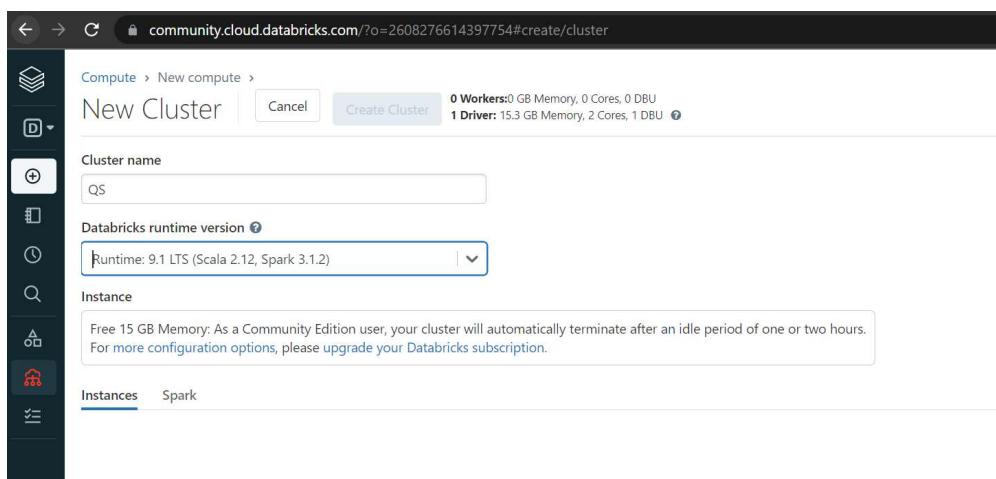
# Save the selected rows as a new CSV file
df_sample.to_csv('benefits1.csv', index=False)
df_sample
```

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 98185MI0180012-05 Routine Eye Exam (Adult)      NaN      NaN .
933912       2017       MI    37651     SERFF 2016-10-19 37651MI0150005-00 Dental Check-Up for Children      NaN      NaN .
209805       2017       MO    32753     HIOS 2017-01-19 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.

Compute >

QS 

Configuration Notebooks (1) Libraries Event log Spark UI Driver logs Metrics Apps Spark cluster UI - Master ▾

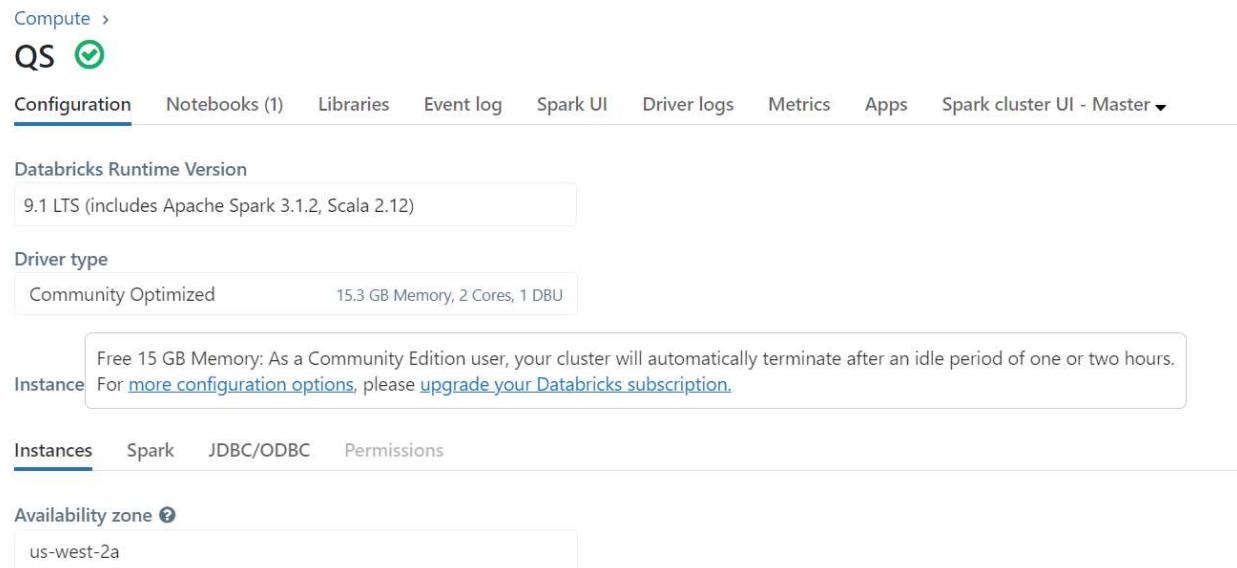
Databricks Runtime Version  
9.1 LTS (includes Apache Spark 3.1.2, Scala 2.12)

Driver type  
Community Optimized 15.3 GB Memory, 2 Cores, 1 DBU

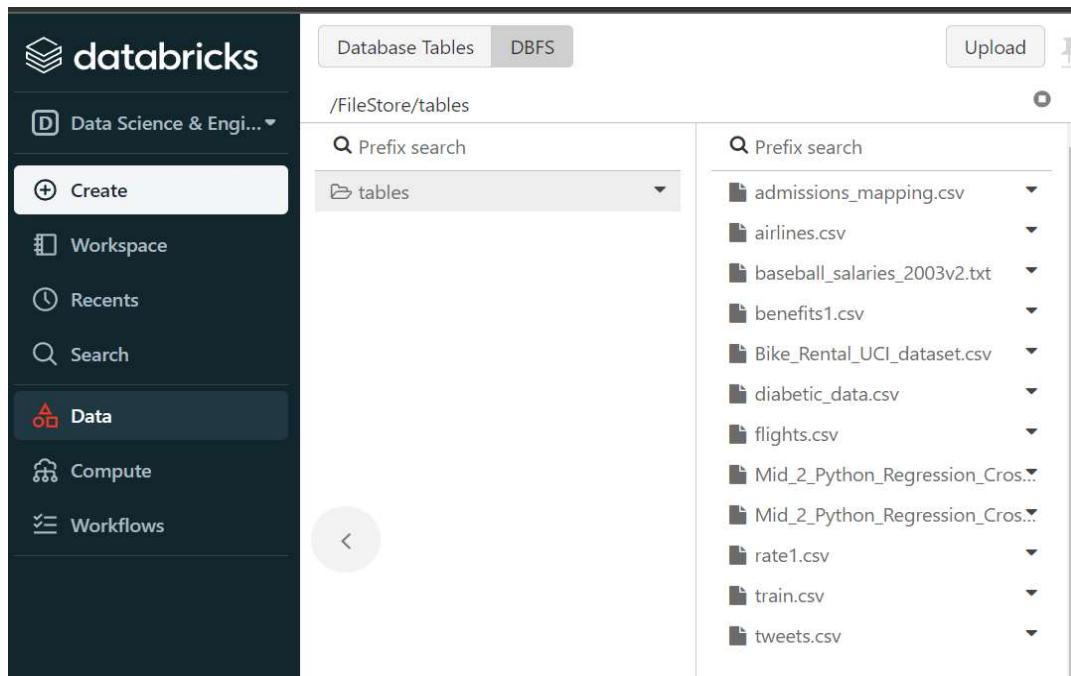
Free 15 GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of one or two hours.  
Instance For [more configuration options](#), please [upgrade your Databricks subscription](#).

Instances Spark JDBC/ODBC Permissions

Availability zone  us-west-2a



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



The screenshot shows the Databricks interface with the Data section selected. On the left, there's a sidebar with options like Create, Workspace, Recents, Search, Data (selected), Compute, and Workflows. The main area has tabs for Database Tables and DBFS. Under DBFS, the path /FileStore/tables is shown. A search bar labeled 'Prefix search' contains 'tables'. To the right is a list of files with dropdown arrows next to them, indicating they can be selected. The files listed are:

- admissions\_mapping.csv
- airlines.csv
- baseball\_salaries\_2003v2.txt
- benefits1.csv
- Bike\_Rental\_UCI\_dataset.csv
- diabetic\_data.csv
- flights.csv
- Mid\_2\_Python\_Regression\_Cros..
- rate1.csv
- train.csv
- tweets.csv

## Upload Data to DBFS

DBFS Target Directory [?](#)

/FileStore/tables

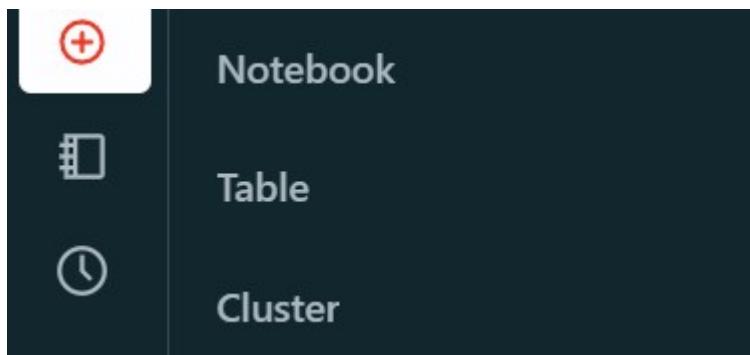
Files uploaded to DBFS are accessible by everyone who has access to this workspace. [Learn more](#)

Files [?](#)

Drop files to upload, or click to browse

[Done](#)

- 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.



## Create Notebook

X

Name

Default Language



Cluster



Cancel

Create

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

The screenshot shows a Databricks notebook interface. The top bar displays the project name "project1" and the language "Python". The cluster dropdown shows "qs". The main area shows a single command cell "Cmd 1" containing the number "1". The status bar at the bottom right indicates "Run all" and "Terminated".

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

The screenshot shows the same Databricks notebook interface after connecting to the "qs" cluster. The cluster dropdown now shows "QS". The command cell "Cmd 1" still contains the number "1". The status bar at the bottom right indicates "Run all" and "QS".

## Step 3: Import all the Libraries

Once the above notebook is created, copy the below code, and paste it in 1<sup>st</sup> 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
```

The screenshot shows the Google Colab interface. On the left is a sidebar with various icons for file operations. The main area contains a code cell titled 'Cmd 1'. The code is as follows:

```
1 # -*- coding: utf-8 -*-
2 """project.ipynb
3 
4 Automatically generated by Colaboratory.
5 
6 Original file is located at
7     https://colab.research.google.com/drive/1Ym0cL_Yy-Ajfsp4Go_FE26ph_qu0f5Bp
8 
9 ###Import all the Libraries
10 """
11 
12 from pyspark.sql.types import *
13 from pyspark.sql.functions import *
14 from pyspark.sql.functions import col, when, max
15 
16 from pyspark.ml import Pipeline
17 
18 from pyspark.ml.classification import DecisionTreeClassifier, LogisticRegression, RandomForestClassifier, LinearSVC, GBTClassifier, FMCClassifier
19 
20 from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer, MinMaxScaler
21 from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit, CrossValidator
22 from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClassificationEvaluator
23 
24 from pyspark.context import SparkContext
25 from pyspark.sql.session import SparkSession
26 
27 """##Create a Spark-Submit Session"""
28
```

A status bar at the top right indicates 'Running command 1. Go to command'.

## 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")
```

The screenshot shows the Google Colab interface with the completed PySpark CLI code in a code cell. The code is identical to the one provided in the previous step.

```
26 """
27 """##Create a Spark-Submit Session"""
28 
29 PYSPARK_CLI = True # conditional statement to run only at shell
30 if PYSPARK_CLI:
31     sc = SparkContext.getOrCreate()
32     spark = SparkSession(sc)
33 
34 # Limit the log
35 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)
```

```

Load the sample dataset from DBFS
Cmd 7
1 # Oracle BDCE
2 #csv = spark.read.csv('/user/agupta25/project/benefits1.csv', inferSchema=True, header=True)
3 # File location and type
4 file_location = "/FileStore/tables/benefits1.csv"
5 file_type = "csv"
6
7 # CSV options
8 infer_schema = "true"
9 first_row_is_header = "true"
10 delimiter = ","
11
12 df = spark.read.format(file_type) \
13 .option("inferSchema", infer_schema) \
14 .option("header", first_row_is_header) \
15 .option("sep", delimiter) \
16 .load(file_location)
17
18 display(df)

```

Table +

	BusinessYear	StateCode	IssuerId	SourceName	ImportDate	StandardComponentId	PlanId	BenefitName
1	2021	IL	36096	SERFF	2020-12-23 20:15:49	36096IL0990144	36096IL0990144-06	Hearing A
2	2021	TN	23552	HIOS	2020-09-18 03:20:24	23552TN0020016	23552TN0020016-05	Inpatient
3	2017	FL	36194	HIOS	2016-09-28 03:28:44	36194FL0030034	36194FL0030034-00	Inpatient
4	2020	GA	60224	HIOS	2019-08-14 03:20:21	60224GA0010003	60224GA0010003-03	Preventiv
5	2020	MI	77739	SERFF	2019-08-21 20:17:17	77739MI0070009	77739MI0070009-03	Infertility
6	2018	MI	60829	SERFF	2017-08-16 20:15:58	60829MI0190010	60829MI0190010-01	Urgent C.
7								

↓ ▾ 6,344 rows | Truncated data ▾

Command complete

## 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()

```

**Prepare the Data**

```
Cmd 9
1 df = df.select('BusinessYear', 'StateCode', 'IssuerId', 'SourceName', 'IsEHB', 'QuantLimitOnSvc', 'Exclusions',
2 'EHBVarReason', col("IsCovered").alias("label"))
3 df.show()
```

BusinessYear	StateCode	IssuerId	SourceName	IsEHB	QuantLimitOnSvc	Exclusions	EHBVarReason	label
2021	IL	36096	SERFF	Yes	Yes	null Substantially Equal	Covered	
2021	TN	23552	HIOS	Yes	null	null	Covered	
2017	FL	36194	HIOS	Yes	No	null	Covered	
2020	GA	60224	HIOS	Yes	null	null	Covered	
2020	MI	77739	SERFF	Yes	null	null	Covered	
2018	MI	60829	SERFF	Yes	null	null	Covered	
2021	UT	68781	SERFF	null	null	null	null Not Covered	
2021	OK	40463	HIOS	Yes	null	null	null Covered	
2020	IL	20129	SERFF	Yes	null	null	null Covered	
2018	PA	16322	SERFF	null	null	null	null Not Covered	
2020	VA	69103	SERFF	Yes	null	null Substantially Equal	Covered	
2019	MT	30751	SERFF	Yes	null	null	Not EHB Not Covered	
2020	IA	93078	SERFF	Yes	null	null	null Covered	
2019	PA	75729	SERFF	Yes	null	null	null Covered	
2020	AZ	53901	HIOS	Yes	null	null	null Covered	
2018	AK	73836	HIOS	Yes	null	null	null Covered	
2017	TX	33602	HIOS	Yes	null	null	null Covered	

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)
```



The screenshot shows a Jupyter Notebook interface. On the left is a sidebar with icons for file operations. The main area has a title bar 'count the null values from prediction col'. Below it is a code cell labeled 'Cmd 12' containing the Python code from the previous block. To the right of the code cell is a status bar with 'Python' and other icons. The output cell below shows the result '2001' followed by 'Command complete'.

## 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()
```

Replace null or whitespace values with None. Later drop the values.

```
Cmd 14
1 from pyspark.sql.functions import when, col
2
3 # Replace empty strings or whitespace with null values
4 df = df.withColumn('label', when(col('label').isin('', ' '), None).otherwise(col('label')))
5
6 # Drop null values from label column
7 df = df.dropna(subset=['label'])
8 df.show()
9
10
```

BusinessYear	StateCode	IssuerId	SourceName	IsEHB	QuantLimitOnSvc	Exclusions	EHBVarReason	label
2021	IL	36996	SERF	Yes	Yes	null	Substantially Equal	Covered
2021	TN	23552	HIOS	Yes	null	null	null	Covered
2017	FL	36194	HIOS	Yes	No	null	null	Covered
2020	GA	60224	HIOS	Yes	null	null	null	Covered
2020	MI	77739	SERF	Yes	null	null	null	Covered
2018	MI	60829	SERF	Yes	null	null	null	Covered
2021	UT	68781	SERFF	null	null	null	null	Not Covered
2021	OK	40463	HIOS	Yes	null	null	null	Covered

## 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()
```

Take Max of all the other columns in dataset having null values

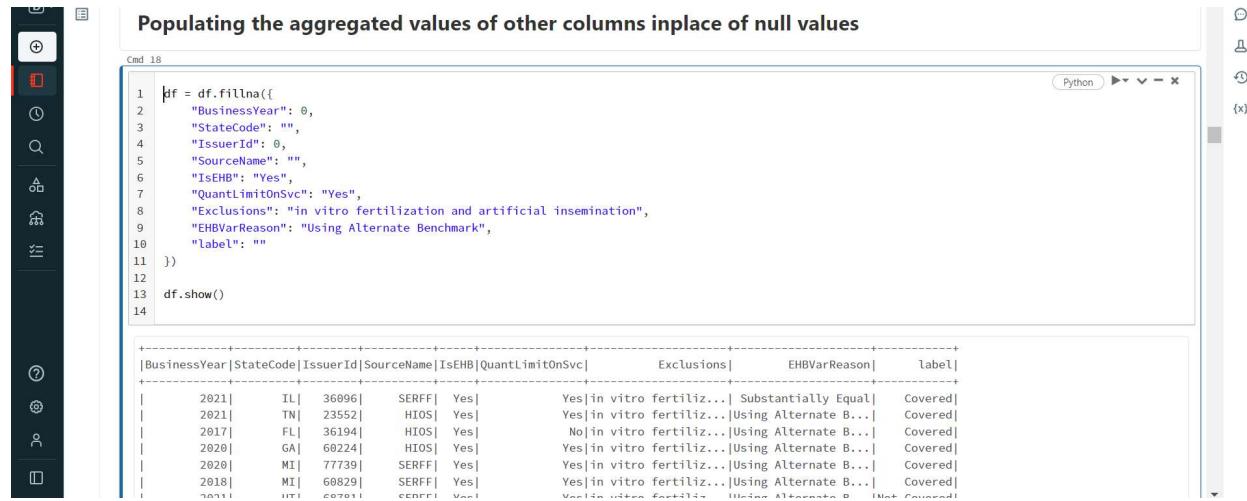
```
Cmd 15
1 df.agg({'IsEHB': 'max', 'QuantLimitOnSvc': 'max', 'Exclusions': 'max', 'EHBVarReason': 'max'}).collect()
2
```

```
Out[13]: [Row(max(Exclusions)=in vitro fertilization and artificial insemination., max(QuantLimitOnSvc)=Yes, max(IsEHB)=Yes, max(EHBVarReason)=Using Alternate Benchmark)]
Command complete
```

## 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()
```



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Populating the aggregated values of other columns in place of null values
- Toolbar:** Includes icons for file operations, cell selection, and help.
- Code Cell:** Contains the Python code from the previous text block.
- Output Cell:** Displays the resulting DataFrame with the following schema and data:

BusinessYear	StateCode	IssuerId	SourceName	IsEHB	QuantLimitOnSvc	Exclusions	EHBVarReason	label
2021	IL	36096	SERFF	Yes	Yes	in vitro fertiliz...	Substantially Equal	Covered
2021	TN	23552	HIOS	Yes	Yes	in vitro fertiliz...	Using Alternate B...	Covered
2017	FL	36194	HIOS	Yes	No	in vitro fertiliz...	Using Alternate B...	Covered
2020	GA	60224	HIOS	Yes	Yes	in vitro fertiliz...	Using Alternate B...	Covered
2020	MI	77739	SERFF	Yes	Yes	in vitro fertiliz...	Using Alternate B...	Covered
2018	MI	60829	SERFF	Yes	Yes	in vitro fertiliz...	Using Alternate B...	Covered
2021	UT	20091	SERFF	Yes	Yes	in vitro fertiliz...	Using Alternate B...	Not Covered

## 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()
```

The screenshot shows a Jupyter Notebook interface with a cell titled "Convert the label into 0 and 1 for classification modelling and prediction." The cell contains the Python code provided above. The output of the cell is a DataFrame with columns: BusinessYear, StateCode, IssuerId, SourceName, IsEHB, QuantLimitOnSvc, Exclusions, EHBVarReason, and label. The data is as follows:

BusinessYear	StateCode	IssuerId	SourceName	IsEHB	QuantLimitOnSvc	Exclusions	EHBVarReason	label
2021	IL	36896	SERFF	Yes	Yes in vitro fertiliz...  Substantially Equal	1		
2021	TN	23552	HIOS	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2017	FL	36194	HIOS	Yes	No in vitro fertiliz... Using Alternate B...	1		
2020	GA	60224	HIOS	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2020	MI	77739	SERFF	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2018	MI	60829	SERFF	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2021	UT	68781	SERFF	Yes	Yes in vitro fertiliz... Using Alternate B...	0		
2021	OK	40463	HIOS	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2020	IL	20129	SERFF	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2018	PA	16322	SERFF	Yes	Yes in vitro fertiliz... Using Alternate B...	0		
2020	VA	69103	SERFF	Yes	Yes in vitro fertiliz...  Substantially Equal	1		
2019	MT	30751	SERFF	Yes	Yes in vitro fertiliz...  Not EHB	0		
2020	IA	93078	SERFF	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2019	PA	75729	SERFF	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2020	AZ	53901	HIOS	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2018	AK	73836	HIOS	Yes	Yes in vitro fertiliz... Using Alternate B...	1		
2017	TX	22607	HIOS	Yes	Yes in vitro fertiliz... Using Alternate B...	1		

## 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()
```

The screenshot shows a Jupyter Notebook interface with a cell titled "Shows the summary of dataset". The cell contains the Python code provided above. The output of the cell is a DataFrame with columns: summary, BusinessYear, StateCode, IssuerId, SourceName, IsEHB, QuantLimitOnSvc, Exclusions, EHBVarReason, and label. The data is as follows:

summary	BusinessYear	StateCode	IssuerId	SourceName	IsEHB	QuantLimitOnSvc	Exclusions	EHBVarReason	label
count	47999	47999	47999	47999 47999	47999	47999	47999	47999	47999
mean	2019.0697097856205	null 50793.196337423695	null  null	null  null	null	null	null	null	null
stddev	1.4978542860833765	null 26507.148720621153	null  null	null  null	null	null	null	null	0.8110168961853372
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			

## 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()
```

The screenshot shows a Jupyter Notebook cell titled "Shows the existing null values in dataset". The cell contains the Python code: `df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()`. The output displays a table with columns for BusinessYear, StateCode, IsUserId, SourceName, IsEHB, QuantLimitOnSvc, Exclusions, and EHBVarReason, all showing 0 null values. The status bar at the bottom right indicates "Command complete".

## 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)
```

The screenshot shows a Jupyter Notebook cell titled "Split the Data for training & testing". The cell contains the Python code for splitting the dataset and printing the counts of training and testing rows. The output shows "Training Rows: 33583 Testing Rows: 14416" and "Command complete".

## 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
1 strIdx_SC = StringIndexer(inputCol = "StateCode", outputCol = "SC",handleInvalid='keep')
2 strIdx_SN = StringIndexer(inputCol = "SourceName", outputCol = "SN",handleInvalid='keep')
3 strIdx_EHB = StringIndexer(inputCol = "IsEHB", outputCol = "EHB",handleInvalid='keep')
4 strIdx_QL = StringIndexer(inputCol = "QuantLimitOnSvc", outputCol = "QL",handleInvalid='keep')
5 strIdx_EX = StringIndexer(inputCol = "Exclusions", outputCol = "EX",handleInvalid='keep')
6 strIdx_EHBVR = StringIndexer(inputCol = "EHBVarReason", outputCol = "EHBVR",handleInvalid='keep')
7
8
9 # the following columns are categorical number such as ID so that it should be Category features
10 catVect = VectorAssembler(inputCols = ["SC", "BusinessYear", "IssuerId", "SN", "EHB", "QL","EHBVR"], outputCol="catFeatures")
11 catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol = "idxCatFeatures", handleInvalid="skip")
12
```

```

# Cat feature vector is normalized

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

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

```

Cmd 31
1 # cat feature vector is normalized
2
3 minMax = MinMaxScaler(inputCol = catIdx.getOutputCol(), outputCol="normFeatures")
4
5 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 classification_models=["Logistic Regression (LR)", "Decision Tree (DT)", "Random Forest (RT)", "Factorization Machine (FM)", "Gradient
7 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]]))

```

```

Cmd 32
1 # define list of models made from Train Validation Split or Cross Validation
2 model = []
3 pipeline = []

Command took 0.03 seconds -- by agupta25@calstatela.edu at 5/18/2023, 1:05:29 PM on QS

Cmd 33
1 # Pipeline process the series of transformation above, which is another transformation
2 for i in range(0,6):
3     pipeline.insert(i,Pipeline(stages=[strIdx_SC,strIdx_SN,strIdx_EHB,strIdx_QL,strIdx_EX,strIdx_EHBVR, catVect, catIdx,minMax, featVect, cls_mod[i]]))

Command took 0.04 seconds -- by agupta25@calstatela.edu at 5/18/2023, 1:05:30 PM on QS

```

## 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],

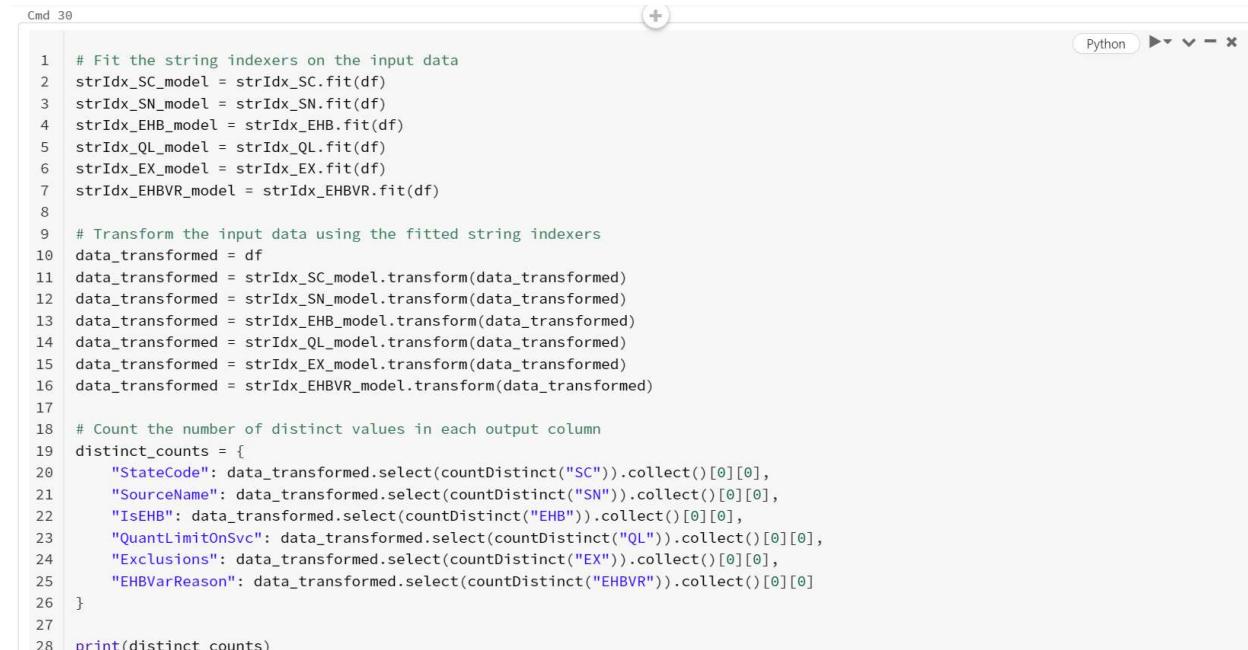
    "QuantLimitOnSvc":
data_transformed.select(countDistinct("QL")).collect()[0][0],

    "Exclusions":
data_transformed.select(countDistinct("EX")).collect()[0][0],

    "EHBVarReason":
data_transformed.select(countDistinct("EHBVR")).collect()[0][0]}

print(distinct_counts)

```



```

Cmd 38 + Python ▶▼ - ×
1 # Fit the string indexers on the input data
2 strIdx_SC_model = strIdx_SC.fit(df)
3 strIdx_SN_model = strIdx_SN.fit(df)
4 strIdx_EHB_model = strIdx_EHB.fit(df)
5 strIdx_QL_model = strIdx_QL.fit(df)
6 strIdx_EX_model = strIdx_EX.fit(df)
7 strIdx_EHBVR_model = strIdx_EHBVR.fit(df)
8
9 # Transform the input data using the fitted string indexers
10 data_transformed = df
11 data_transformed = strIdx_SC_model.transform(data_transformed)
12 data_transformed = strIdx_SN_model.transform(data_transformed)
13 data_transformed = strIdx_EHB_model.transform(data_transformed)
14 data_transformed = strIdx_QL_model.transform(data_transformed)
15 data_transformed = strIdx_EX_model.transform(data_transformed)
16 data_transformed = strIdx_EHBVR_model.transform(data_transformed)
17
18 # Count the number of distinct values in each output column
19 distinct_counts = {
20     "StateCode": data_transformed.select(countDistinct("SC")).collect()[0][0],
21     "SourceName": data_transformed.select(countDistinct("SN")).collect()[0][0],
22     "IsEHB": data_transformed.select(countDistinct("EHB")).collect()[0][0],
23     "QuantLimitOnSvc": data_transformed.select(countDistinct("QL")).collect()[0][0],
24     "Exclusions": data_transformed.select(countDistinct("EX")).collect()[0][0],
25     "EHBVarReason": data_transformed.select(countDistinct("EHBVR")).collect()[0][0]
26 }
27
28 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())

```



The screenshot shows a Jupyter Notebook interface with a code cell titled "Tune hyperparameters using ParamGrid". The code cell contains the following Python code:

```

1 paramGrid=[]
2
3 paramGrid.insert(0,(ParamGridBuilder() \
4     .addGrid(cls_mod[0].regParam, [0.01, 0.3]) \
5     .addGrid(cls_mod[0].elasticNetParam, [0.0, 0.5]) \
6     .addGrid(cls_mod[0].maxIter, [10,20]) \
7     .build()))
8
9
10 paramGrid.insert(1,ParamGridBuilder() \
11     .addGrid(cls_mod[1].maxBins, [64,128,256]) \
12     .addGrid(cls_mod[1].maxDepth, [2, 5, 10]) \
13     .addGrid(cls_mod[1].impurity, ["gini", "entropy"]) \
14     .addGrid(cls_mod[1].minInstancesPerNode, [1, 5, 10]) \
15     .build())
16
17
18 paramGrid.insert(2,ParamGridBuilder() \
19     .addGrid(cls_mod[2].numTrees, [50, 100, 150]) \
20     .addGrid(cls_mod[2].maxBins, [64,128,256]) \
21     .addGrid(cls_mod[2].maxDepth, [2, 5, 10]) \
22     .build())
23
24

```

The code cell is labeled "Cmd 35" and has a "Python" language selector. The notebook interface includes a toolbar on the left and a status bar at the bottom.

```

24 paramGrid.insert(3,ParamGridBuilder()\
25 .addGrid(cls_mod[3].regParam, [0.01, 0.1]) \
26 .addGrid(cls_mod[3].stepSize, [0.1,1])\
27 .addGrid(cls_mod[3].factorSize, [2,4])\
28 .build())
29
30
31
32 paramGrid.insert(4,ParamGridBuilder()\
33 .addGrid(cls_mod[4].maxDepth, [2, 5])\
34 .addGrid(cls_mod[4].maxIter, [10, 20])\
35 .addGrid(cls_mod[4].minInfoGain, [0.0])\
36 .build())
37
38
39 paramGrid.insert(5,ParamGridBuilder() \
40     .addGrid(cls_mod[5].regParam, [0.01, 0.5]) \
41     .addGrid(cls_mod[5].maxIter, [1, 5]) \
42     .addGrid(cls_mod[5].tol, [1e-4, 1e-3]) \
43     .addGrid(cls_mod[5].fitIntercept, [True, False]) \
44     .addGrid(cls_mod[5].standardization, [True, False]) \
45     .build())
46
47
48
49
50

```

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))

```

```
Used CrossValidator for modelling

Cmd 37
1 #cv=[]
2 K=3
3 for i in range(0,6):
4     cv.insert(i, CrossValidator(estimator=pipeline[i],
5                                evaluator=BinaryClassificationEvaluator(),
6                                estimatorParamMaps=paramGrid[i],
7                                numFolds=K))
8
9
10 #cv1 = CrossValidator(estimator=pipeline1, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid1, numFolds=K)
11 #cv2= CrossValidator(estimator=pipeline2, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid2, numFolds=K)
12 #cv3 = CrossValidator(estimator=pipeline3, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid3, numFolds=K)
13 #cv4 = CrossValidator(estimator=pipeline4, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid4, numFolds=K)
14 #cv5 = CrossValidator(estimator=pipeline5, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid5, numFolds=K)
15
16 #cv = TrainValidationSplit(estimator=pipeline, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid, trainRatio=0.8)
17

Command complete
```

## 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")
```

```

Cmd 39 Python ►▼▼—×
```

```

1 import time
2
3 start_time = []
4 end_time = []
5 computation_time = []
6
7 for i in range(0, 6):
8     start_time.insert(i, time.time())
9     model.insert(i, cv[i].fit(train))
10    # model1 = cv1.fit(train)
11    # model2 = cv2.fit(train)
12    # model3 = cv3.fit(train)
13    # model4 = cv4.fit(train)
14    # model5 = cv5.fit(train)
15    end_time.insert(i, time.time())
16    computation_time.insert(i, (end_time[i] - start_time[i]) / 60.0)
17    print("Computation time:", i, " ", computation_time[i], "minutes")
18
```

```

/databricks/spark/python/pyspark/ml/util.py:886: UserWarning: Cannot find mlflow module. To enable MLflow logging, install mlflow from PyPI.
  warnings.warn(_MLFLOW_INSTRUMENTATION_NO_MLFLOW_WARNING)
Computation time: 0 5.152020577589671 minutes
Computation time: 1 29.827600558598835 minutes
Computation time: 2 36.10357682307561 minutes
Computation time: 3 13.262092169125875 minutes
Computation time: 4 6.539898025989532 minutes
Computation time: 5 15.529444551467895 minutes
Command complete

```

## 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 pipeline 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
```

Python ▶ ▷ ▷ - ×

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						
0,7383... [38.0,0.0,0,73836.0... [1.0,0.0,... 0.709384... [1.0,0.0,0,0.709384... [[-1.5173015736635... [0.17985922014973... 1.0	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.													
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... [34.0,0.0,0,46944.0... [0.89473684210526... [0.89473684210526... [[-3.539492035324... [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... [34.0,0.0,0,46944.0... [0.89473684210526... [0.89473684210526... [[-3.539492035324... [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... [34.0,0.0,0,46944.0... [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... [34.0,0.0,0,46944.0... [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... [34.0,0.0,0,46944.0... [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
[1.0,0.0,0,0.709384...]	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

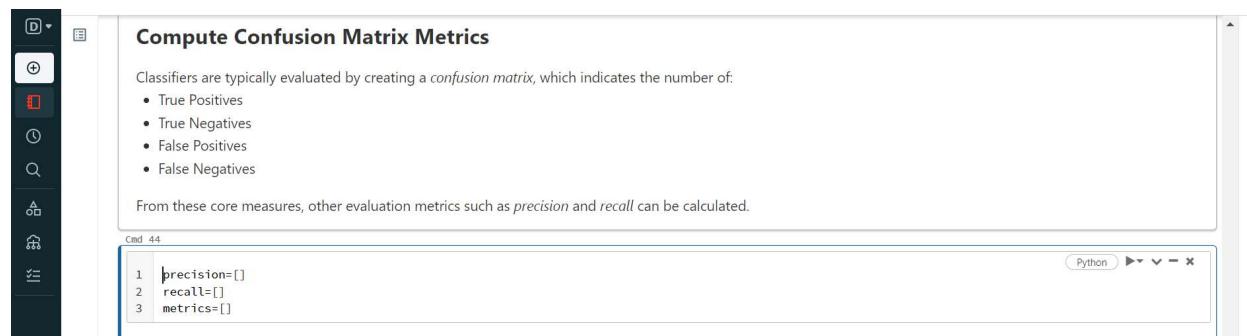
Classifiers are typically evaluated by creating a \*confusion matrix\*, which indicates the number of:

- 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 = []
```



The screenshot shows a Jupyter Notebook interface with a sidebar containing icons for file operations like new, open, save, and search. The main area has a title 'Compute Confusion Matrix Metrics'. Below it, a text block states: 'Classifiers are typically evaluated by creating a confusion matrix, which indicates the number of:' followed by a bulleted list of four items: 'True Positives', 'True Negatives', 'False Positives', and 'False Negatives'. Another text block below says: 'From these core measures, other evaluation metrics such as *precision* and *recall* can be calculated.' At the bottom of the notebook, there's a command cell labeled 'Cmd 44' containing Python code:

```
1 precision=[]
2 recall=[]
3 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

```

1 for i in range(0,6):
2     tp = float(predicted[i].filter("prediction== 1.0 AND truelabel == 1").count())
3     fp = float(predicted[i].filter("prediction== 1.0 AND truelabel == 0").count())
4     tn = float(predicted[i].filter("prediction== 0.0 AND truelabel == 0").count())
5     fn = float(predicted[i].filter("prediction==0.0 AND truelabel == 1").count())
6     precision.insert(i, tp / (tp + fp))
7     recall.insert(i, tp / (tp + fn))
8     metrics.insert(i, spark.createDataFrame([
9         {"TP": tp,
10        ("FP", fp),
11        ("TN", tn),
12        ("FN", fn),
13        ("Precision", tp / (tp + fp)),
14        ("Recall", tp / (tp + fn))], [{"metric": "value"})))
15     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

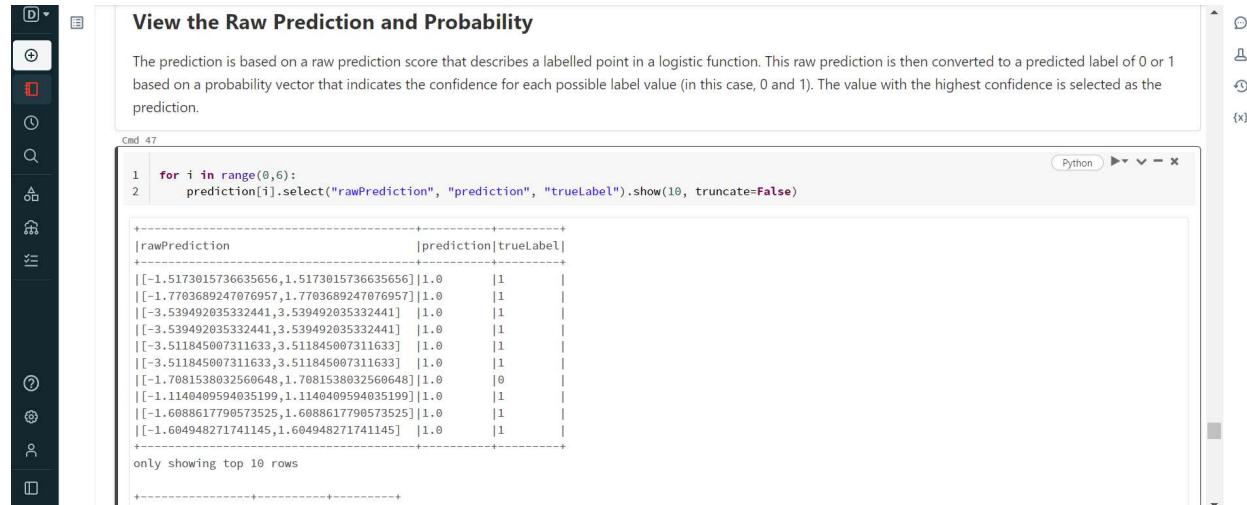
## Step 23: View the Raw Prediction and Probability

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)
```



```
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 predicted label of 0 or 1 based on a probability vector that indicates the confidence for each possible label value (in this case, 0 and 1). The value with the highest confidence is selected as the prediction.

Cmd 47
Python ▶ v - x

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

+-----+-----+-----+
|rawPrediction |prediction|trueLabel|
+-----+-----+-----+
|[ -1.5173015736635656, 1.5173015736635656] |1.0 |1 |
|[ -1.7703689247076957, 1.7703689247076957] |1.0 |1 |
|[ -3.539492035332441, 3.539492035332441] |1.0 |1 |
|[ -3.539492035332441, 3.539492035332441] |1.0 |1 |
|[ -3.511845007311633, 3.511845007311633] |1.0 |1 |
|[ -3.511845007311633, 3.511845007311633] |1.0 |1 |
|[ -1.7081538032566648, 1.7081538032566648] |1.0 |0 |
|[ -1.1140409594035199, 1.1140409594035199] |1.0 |1 |
|[ -1.6088617790573525, 1.6088617790573525] |1.0 |1 |
|[ -1.604948271741145, 1.604948271741145] |1.0 |1 |
+-----+-----+-----+
only showing top 10 rows
+-----+-----+-----+
```

## 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.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

**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.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

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.

$$\text{F1 Score} = 2 * ((\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall}))$$

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 = {:.3f}".format(auc_roc))

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

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

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

# accuracy

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

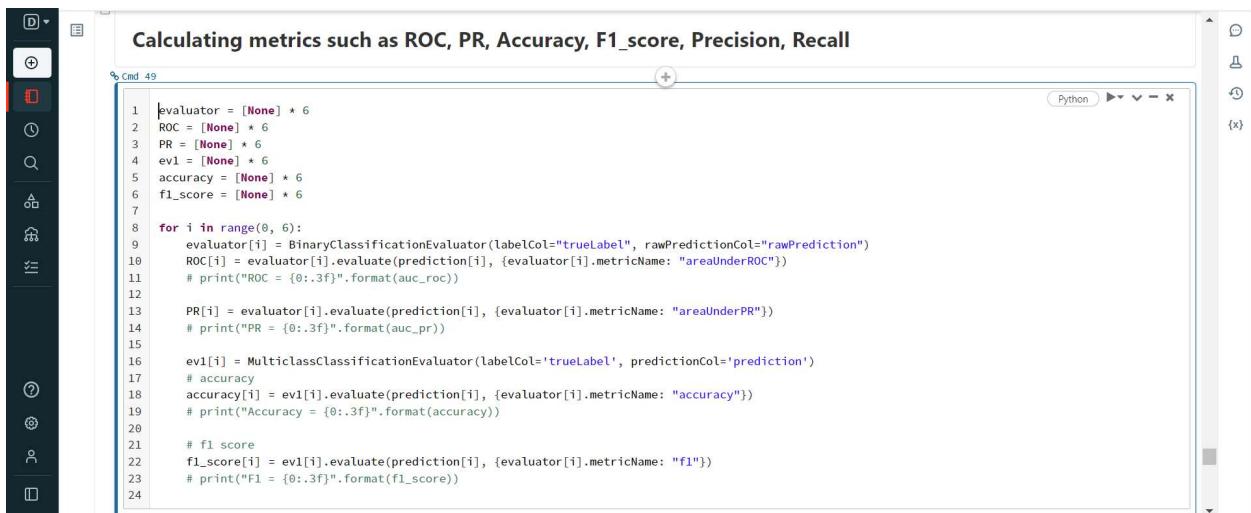
# print("Accuracy = {:.3f}".format(accuracy))

# f1 score

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

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

```



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code. The title bar of the cell reads "Calculating metrics such as ROC, PR, Accuracy, F1\_score, Precision, Recall". The code cell itself contains the following Python code:

```

1 evaluator = [None] * 6
2 ROC = [None] * 6
3 PR = [None] * 6
4 ev1 = [None] * 6
5 accuracy = [None] * 6
6 f1_score = [None] * 6
7
8 for i in range(0, 6):
9     evaluator[i] = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="rawPrediction")
10    ROC[i] = evaluator[i].evaluate(prediction[i], {evaluator[i].metricName: "areaUnderROC"})
11    # print("ROC = {:.3f}".format(auc_roc))
12
13    PR[i] = evaluator[i].evaluate(prediction[i], {evaluator[i].metricName: "areaUnderPR"})
14    # print("PR = {:.3f}".format(auc_pr))
15
16    ev1[i] = MulticlassClassificationEvaluator(labelCol='trueLabel', predictionCol='prediction')
17
18    # accuracy
19    accuracy[i] = ev1[i].evaluate(prediction[i], {evaluator[i].metricName: "accuracy"})
20    # print("Accuracy = {:.3f}".format(accuracy))
21
22    # f1 score
23    f1_score[i] = ev1[i].evaluate(prediction[i], {evaluator[i].metricName: "f1"})
24    # print("F1 = {:.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)
```

### Comparing all metrics at one place

Cmd 51

```
1 import pandas as pd
2
3 results = {
4     'Model': classification_models,
5     'Computation Time (min)': computation_time,
6     'ROC': ROC,
7     'PR': PR,
8     'Accuracy': accuracy,
9     'F1 Score': f1_score,
10    'Precision': precision,
11    'Recall': recall
12 }
13
14 df_results = pd.DataFrame.from_dict(results)
15 df_results = df_results.set_index('Model').transpose()
16
17 print(df_results)
18
```

Markdown  Python 

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	Gradient 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 GBTCClassifier 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))

```

Cmd 53

**Calculate the Feature importance of the best Model - GBT Classifier.**

Cmd 54

```

1 # Access feature importance of the best GBT model
2 bestModel = model[4].bestModel
3 gbtModel = bestModel.stages[-1] # Assuming GBTCClassifier is the last stage in the pipeline
4
5 featureImportances = gbtModel.featureImportances
6 # Create a list of tuples containing feature names and importances
7 importance_tuples = [(feature, importance) for feature, importance in zip(df.columns, featureImportances)]
8
9 # Sort the feature importances based on importance values in ascending order
10 sorted_importances = sorted(importance_tuples, key=lambda x: x[1], reverse=True)
11
12 # Print the feature importances in ascending order
13 print("Feature Importance:")
14 for feature, importance in sorted_importances:
15     print("{}: {}".format(feature, importance))

Feature Importance:
Exclusions: 0.555733072247023
BusinessYear: 0.1628008478674989
IssuerId: 0.1319338167516896
QuantLimitOnSvc: 0.12203959542332579
SourceName: 0.015728677623409574
StateCode: 0.011763990087053302
IsEHB: 0.0

```

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.

The screenshot shows a Jupyter Notebook interface with two code cells and one table cell.

**Cmd 6:** Load the sample dataset from DBFS

```
# Oracle BDCE
csv = spark.read.csv('/user/agupta25/project/benefit.csv', inferSchema=True, header=True)
# File location and type
```

**Cmd 7:**

```
# Oracle BDCE
csv = spark.read.csv('/user/agupta25/project/benefit.csv', inferSchema=True, header=True)
# File location and type
```

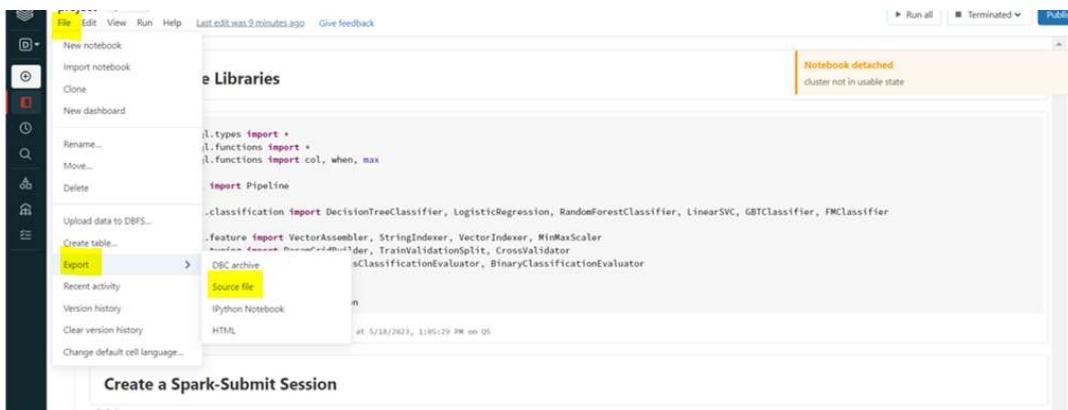
**Table:**

	BusinessYear	StateCode	IssuerId	SourceName	ImportDate	StandardComponentId	PlanId	BenefitName
1	2021	IL	36096	SERFF	2020-12-23 20:15:49	36096IL0990144	36096IL0990144-06	Hearing Aids
2	2021	TN	23552	HIOS	2020-09-18 03:20:24	23552TN0020016	23552TN0020016-05	Inpatient Physician and Surgical Services
3	2017	FL	36194	HIOS	2016-09-28 03:28:44	36194FL0030034	36194FL0030034-00	Inpatient Hospital Services (e.g., Hospital Stay)
4	2020	GA	60224	HIOS	2019-08-14 03:20:21	60224GA0010003	60224GA0010003-03	Preventive Care/Screening/Immunization
5	2020	MI	77739	SERFF	2019-08-21 20:17:17	77739MI0070009	77739MI0070009-03	Infertility Treatment
6	2018	MI	60829	SERFF	2017-08-16 20:15:58	60829MI0190010	60829MI0190010-01	Urgent Care Centers or Facilities
7	2021	UT	68781	SERFF	2020-12-29 20:15:44	68781UT0200008	68781UT0200008-03	Treatment for Temporomandibular Joint Disorders

6,344 rows | Truncated data | 21.91 seconds runtime | Refreshed 9 hours ago

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 "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\benefit 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 agupta25@144.24.53.159
Username: agupta25
Password: agupta25
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/
```

- 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!/org/slf4j/impl/Log4jLoggerFactory.class]
SLF4J: Found binding in [jar:file:/usr/odh/1.1.5/hadoop/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/Log4jLoggerFactory.class]
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	"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	Gradient 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](https://github.com/rahulbhogale/5560) [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](https://github.com/rahulbhogale/5560)