1. Group's information

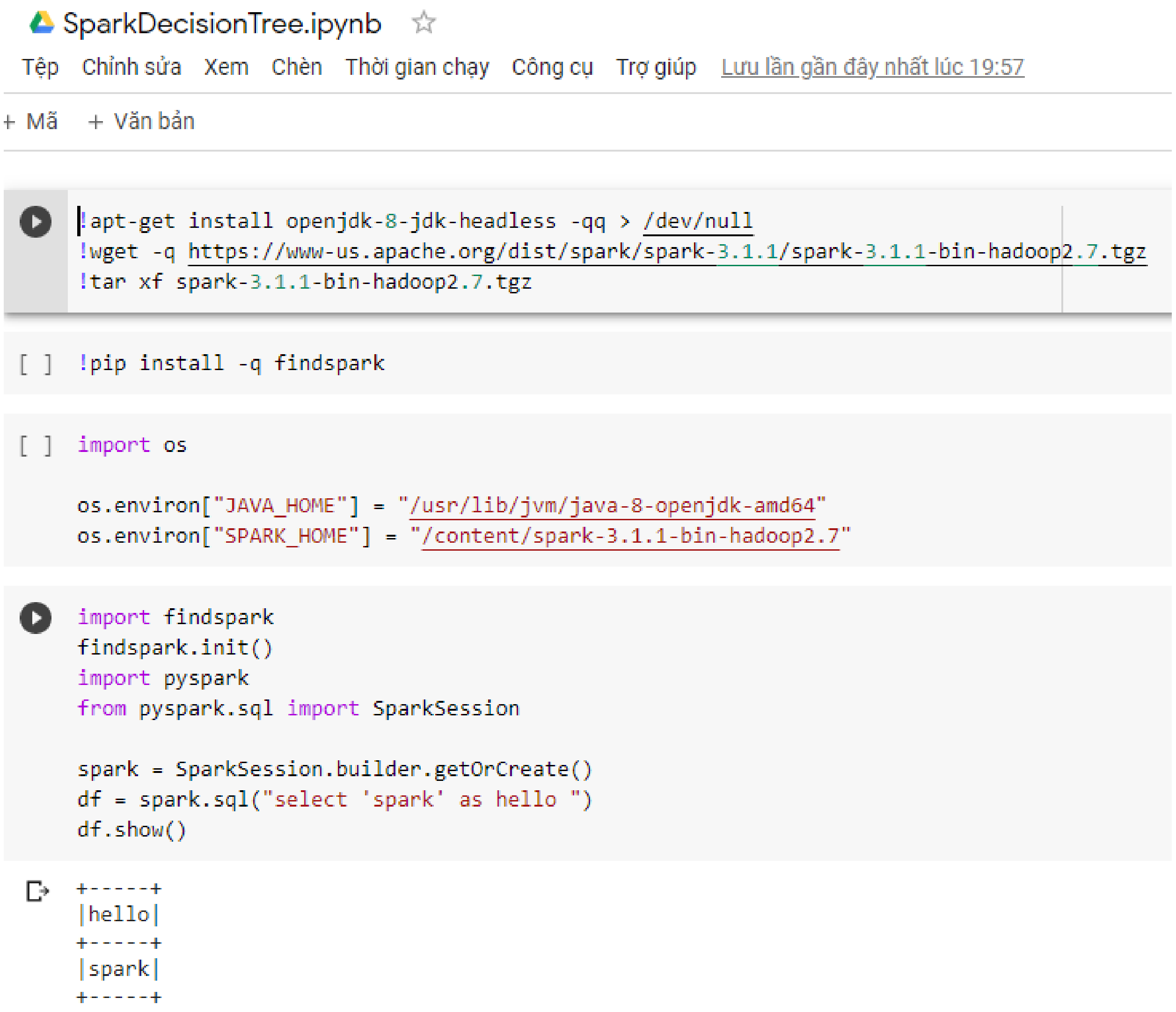
Name	Student ID
Trần Đại Chí	18127070
Phan Quang Đại	18127073
Phan Tấn Đạt	18127078
Trần Minh Quang	18127192

2. Group's result work

- Our team have referenced the solution and read the code in the Spark's document to understand and implement again

- Requirement 1:

+ Decision Tree


SparkDecisionTree.ipynb
Tệp Chỉnh sửa Xem Chèn Thời gian chạy Công cụ Trợ giúp Lưu lần gần đây nhất lúc 19:57
+ Mã + Văn bản

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-hadoop2.7.tgz
!tar xf spark-3.1.1-bin-hadoop2.7.tgz
```

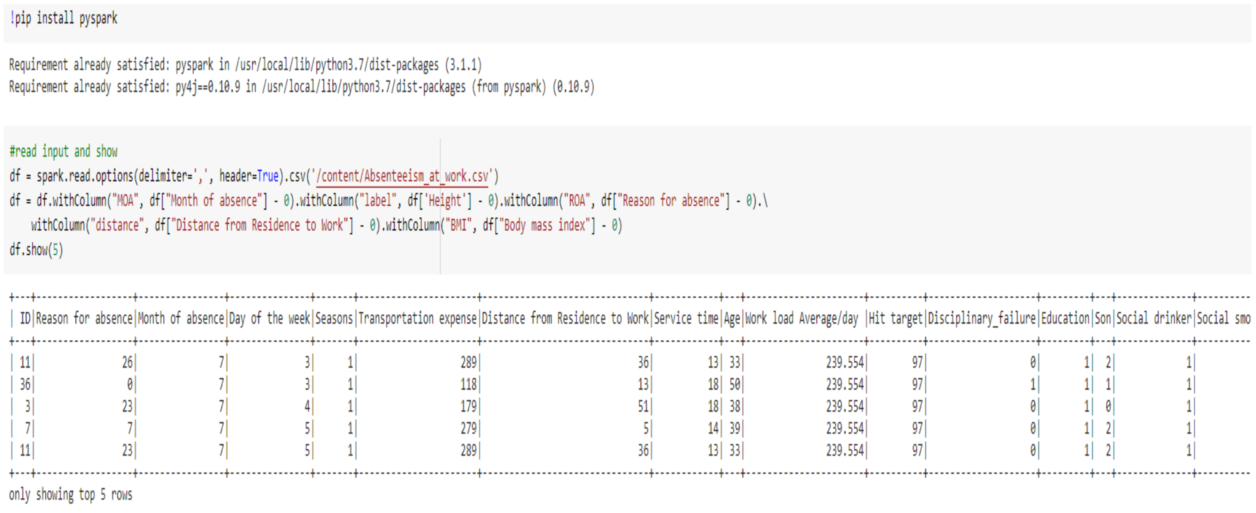
```
!pip install -q findspark
```

```
import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.1-bin-hadoop2.7"
```

```
import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
df = spark.sql("select 'spark' as hello ")
df.show()
```

```
+-----+
|hello|
+-----+
|spark|
+-----+
```

First, we will create a file name SparkDecisionTree.ipynb in google colab, then install Spark and define path for JAVA_HOME as well as SPARK_HOME and if our program can print hello world as the end of the picture, it means our spark have been installed successfully in google colab



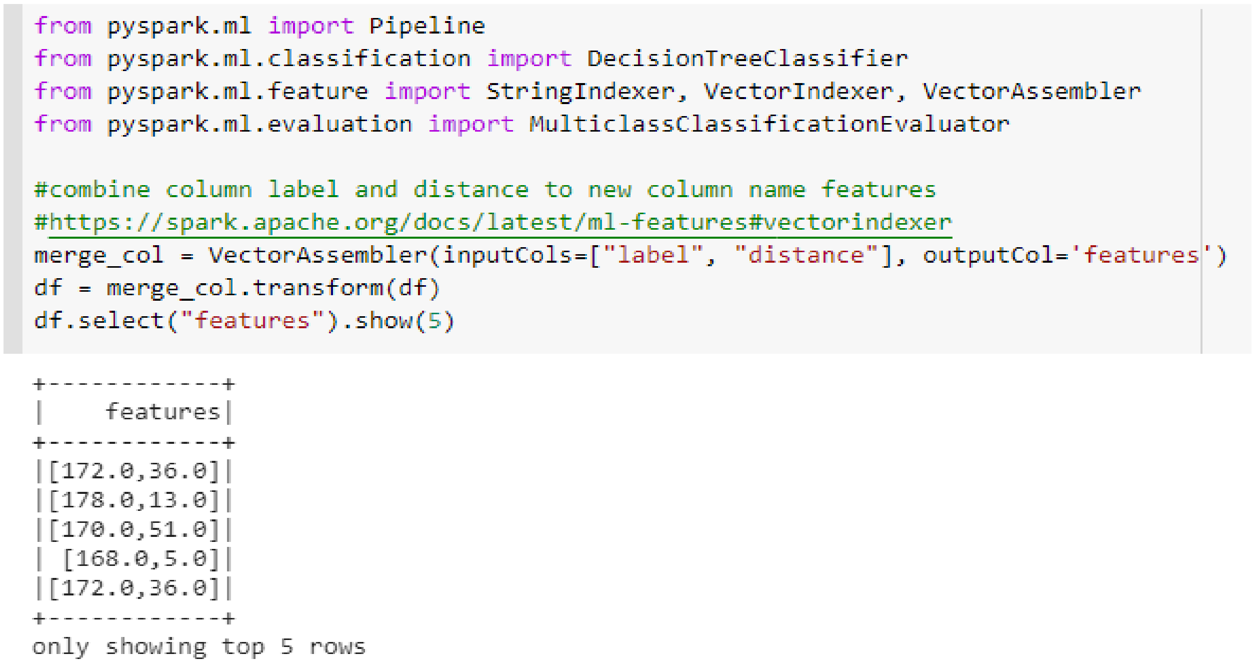
```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark) (0.10.9)
```

```
#read input and show
df = spark.read.options(delimiter=',', header=True).csv('/content/Absenteeism_at_work.csv')
df = df.withColumn("MOA", df["Month of absence"] - 0).withColumn("label", df['Height'] - 0).withColumn("ROA", df["Reason for absence"] - 0).\
    withColumn("distance", df["Distance from Residence to Work"] - 0).withColumn("BMI", df["Body mass index"] - 0)
df.show(5)
```

```
+---+------------------+----------------+---------------+-------+----------------------+-------------------------------+------------+---+----------------------+----------+--------------------+---------+---+--------------+----------
| ID|Reason for absence|Month of absence|Day of the week|Seasons|Transportation expense|Distance from Residence to Work|Service time|Age|Work load Average/day |Hit target|Disciplinary_failure|Education|Son|Social drinker|Social smo
+---+------------------+----------------+---------------+-------+----------------------+-------------------------------+------------+---+----------------------+----------+--------------------+---------+---+--------------+----------
| 11|                26|               7|              3|      1|                   289|                             36|          13| 33|               239.554|        97|                   0|        1|  2|             1|
| 36|                 0|               7|              3|      1|                   118|                             13|          18| 50|               239.554|        97|                   1|        1|  1|             1|
|  3|                23|               7|              4|      1|                   179|                             51|          18| 38|               239.554|        97|                   0|        1|  0|             1|
|  7|                 7|               7|              5|      1|                   279|                              5|          14| 39|               239.554|        97|                   0|        1|  2|             1|
| 11|                23|               7|              5|      1|                   289|                             36|          13| 33|               239.554|        97|                   0|        1|  2|             1|
+---+------------------+----------------+---------------+-------+----------------------+-------------------------------+------------+---+----------------------+----------+--------------------+---------+---+--------------+----------
only showing top 5 rows
```

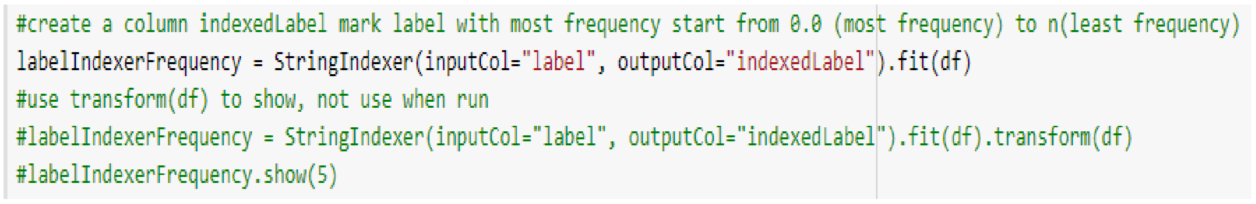
Next, we will install pyspark library and read our data, choose ‘Height’ as label and convert columns ‘Month of absence’, ‘Reason for absence’, ‘Distance from Residence to Work’, ‘Body mass index’, ‘Height’ from type of string to double



```
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer, VectorAssembler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

#combine column label and distance to new column name features
#https://spark.apache.org/docs/latest/ml-features#vectorindexer
merge_col = VectorAssembler(inputCols=["label", "distance"], outputCol='features')
df = merge_col.transform(df)
df.select("features").show(5)
```

```
+------------+
|    features|
+------------+
|[172.0,36.0]|
|[178.0,13.0]|
|[170.0,51.0]|
| [168.0,5.0]|
|[172.0,36.0]|
+------------+
only showing top 5 rows
```

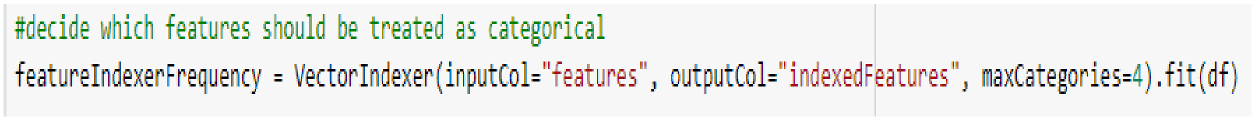
We will use VectorAssembler to merge two column ‘label’ and ‘distance’ to new column name ‘features’ and because VectorAssembler just accept data type of float/double, so we must convert our column from data type of string to double as previous step



```
#create a column indexedLabel mark label with most frequency start from 0.0 (most frequency) to n(least frequency)
labelIndexerFrequency = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(df)
#use transform(df) to show, not use when run
#labelIndexerFrequency = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(df).transform(df)
#labelIndexerFrequency.show(5)
```

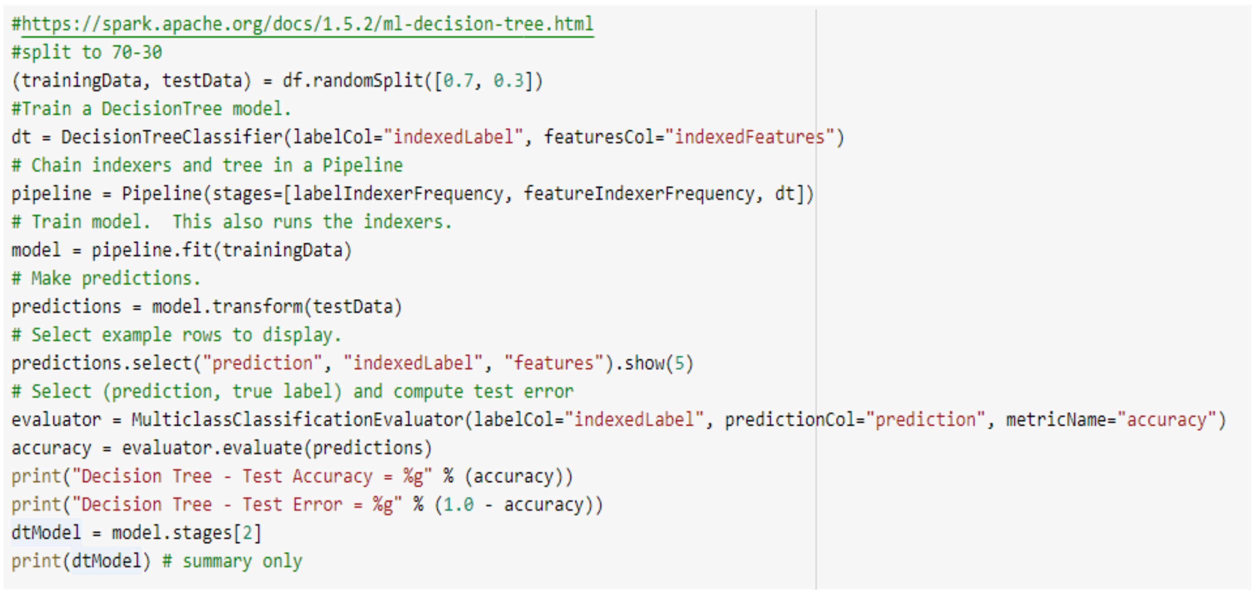
We will use StringIndexer function to mark data with most frequency start from 0.0 (most frequency) to n (least frequency). Example: our label column includes 3 characters A, 2 characters B and 1 character C, then our indexLabel will be as bellow:

Label	indexedLabel
A	0.0
A	0.0
A	0.0
B	1.0
B	1.0
C	2.0



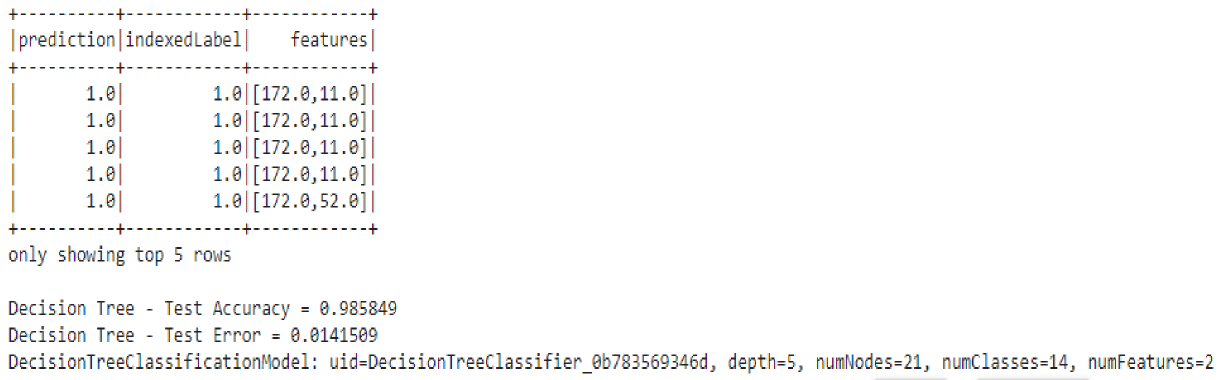
```
#decide which features should be treated as categorical
featureIndexerFrequency = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(df)
```

Next, we will decide which features should be treated as categorical by taking an input column name ‘features’ of type Vector and a parameter maxCategories to decide which features should be categorical based on the number of distinct values. This function will be helpful for improving performance of Decision Tree algorithm



```
#https://spark.apache.org/docs/1.5.2/ml-decision-tree.html
#split to 70-30
(trainingData, testData) = df.randomSplit([0.7, 0.3])
#Train a DecisionTree model.
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexerFrequency, featureIndexerFrequency, dt])
# Train model.  This also runs the indexers.
model = pipeline.fit(trainingData)
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("prediction", "indexedLabel", "features").show(5)
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Decision Tree - Test Accuracy = %g" % (accuracy))
print("Decision Tree - Test Error = %g" % (1.0 - accuracy))
dtModel = model.stages[2]
print(dtModel) # summary only
```

Finally, we split trainingData and testData to 70 and 30, or you can choose another ratio if you want, use DecisionTreeClassifier function to train a DecisionTree model, chain indexers and tree in a pipeline and use pipeline to fit the trainingData, then make the prediction and use MulticlassClassificationEvaluator function to make the evaluation for multiclass classification

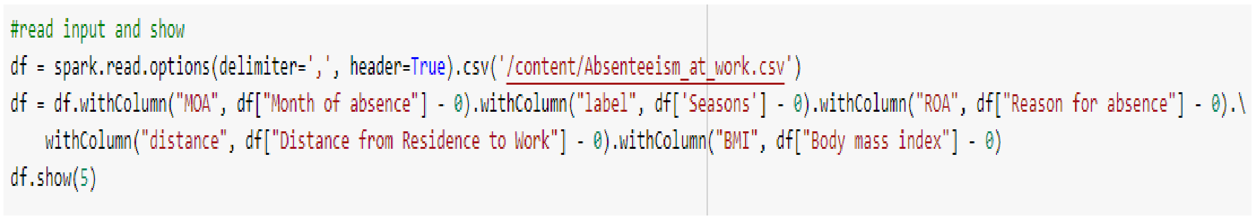


```
+----------+-----------+------------+
|prediction|indexedLabel|    features|
+----------+-----------+------------+
|       1.0|        1.0|[172.0,11.0]|
|       1.0|        1.0|[172.0,11.0]|
|       1.0|        1.0|[172.0,11.0]|
|       1.0|        1.0|[172.0,11.0]|
|       1.0|        1.0|[172.0,52.0]|
+----------+-----------+------------+
only showing top 5 rows

Decision Tree - Test Accuracy = 0.985849
Decision Tree - Test Error = 0.0141509
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_0b783569346d, depth=5, numNodes=21, numClasses=14, numFeatures=2
```

We can see that with this dataset, our algorithm gives a pretty high accuracy rate of 98.5%

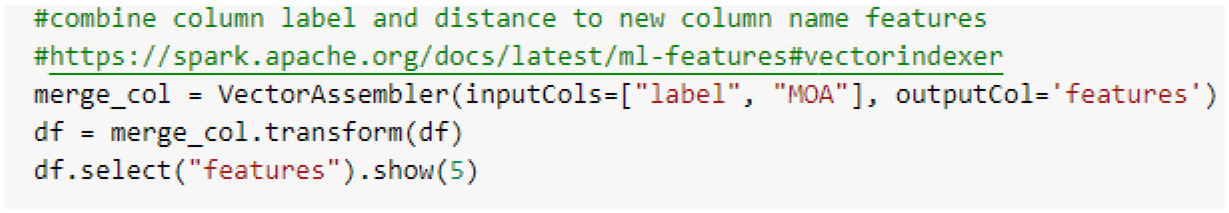
+ Naïve Bayes

First, we create a new file name SparkNaiveBayes.ipynb and install Spark, define path for JAVA_HOME, SPARK_HOME and test if Spark have been installed successfully in google colab follow the same steps as the previous algorithm we did

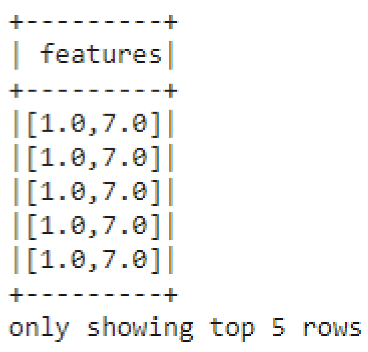


```
#read input and show
df = spark.read.options(delimiter=',', header=True).csv('/content/Absenteeism_at_work.csv')
df = df.withColumn("MOA", df["Month of absence"] - 0).withColumn("label", df['Seasons'] - 0).withColumn("ROA", df["Reason for absence"] - 0).\
   withColumn("distance", df["Distance from Residence to Work"] - 0).withColumn("BMI", df["Body mass index"] - 0)
df.show(5)
```

Next, we will choose ‘Seasons’ as label instead of ‘Height’

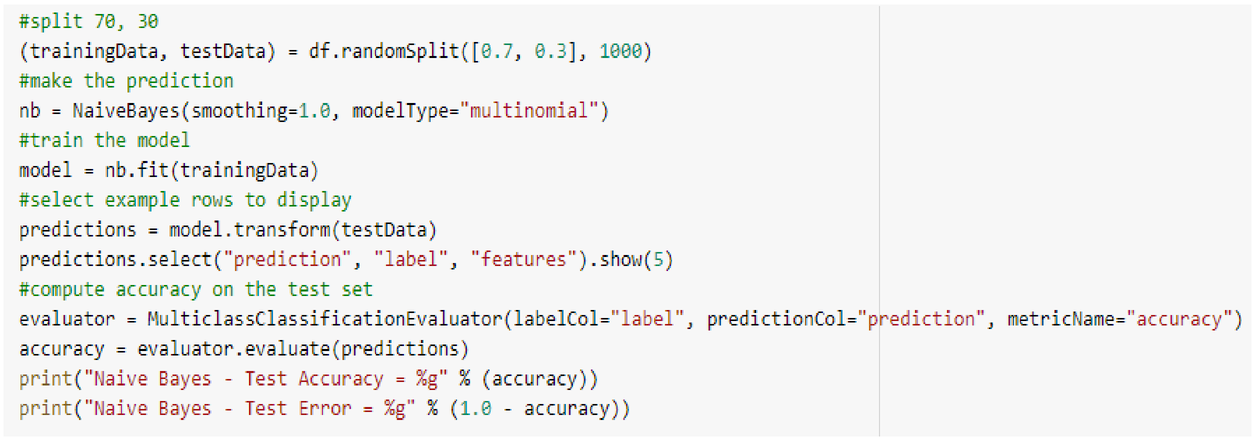


```
#combine column label and distance to new column name features
#https://spark.apache.org/docs/latest/ml-features#vectorindexer
merge_col = VectorAssembler(inputCols=["label", "MOA"], outputCol='features')
df = merge_col.transform(df)
df.select("features").show(5)
```



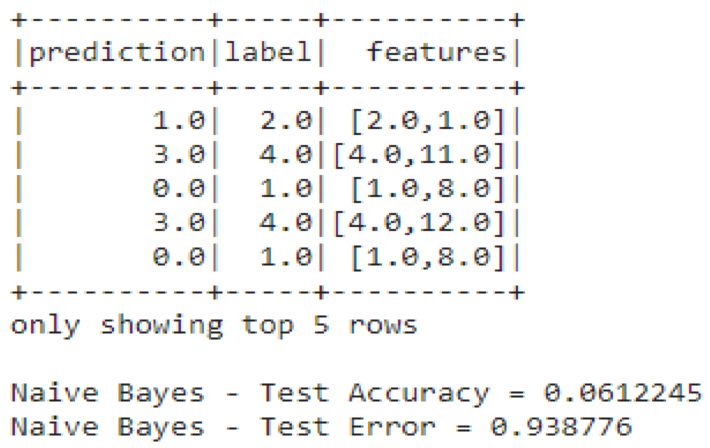
```
+---------+
| features|
+---------+
|[1.0,7.0]|
|[1.0,7.0]|
|[1.0,7.0]|
|[1.0,7.0]|
|[1.0,7.0]|
+---------+
only showing top 5 rows
```

Then. We will combine column ‘label’ and ‘MOA’ to new column name ‘features’



```
#split 70, 30
(trainingData, testData) = df.randomSplit([0.7, 0.3], 1000)
#make the prediction
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
#train the model
model = nb.fit(trainingData)
#select example rows to display
predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)
#compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Naive Bayes - Test Accuracy = %g" % (accuracy))
print("Naive Bayes - Test Error = %g" % (1.0 - accuracy))
```

Finally, we will split trainingData and testData with the ratio 70:30 as previous algorithm, use NaiveBayes function to train the NaiveBayes model with modelType=”multinomial” (Spark also support modelType Bernoulli) and apply smoothing default to 1.0, then make the prediction and make the evaluation for multiclass classification

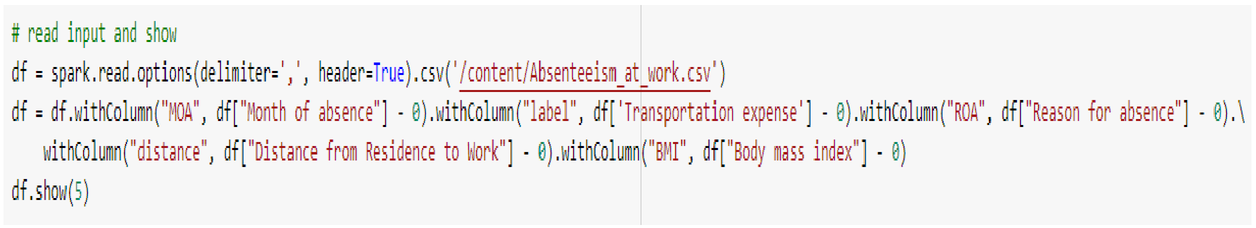


```
+----------+-----+----------+
|prediction|label|  features|
+----------+-----+----------+
|       1.0|  2.0| [2.0,1.0]|
|       3.0|  4.0|[4.0,11.0]|
|       0.0|  1.0| [1.0,8.0]|
|       3.0|  4.0|[4.0,12.0]|
|       0.0|  1.0| [1.0,8.0]|
+----------+-----+----------+
only showing top 5 rows

Naive Bayes - Test Accuracy = 0.0612245
Naive Bayes - Test Error = 0.938776
```

As we can see, our algorithm gives a bad accuracy with the same dataset

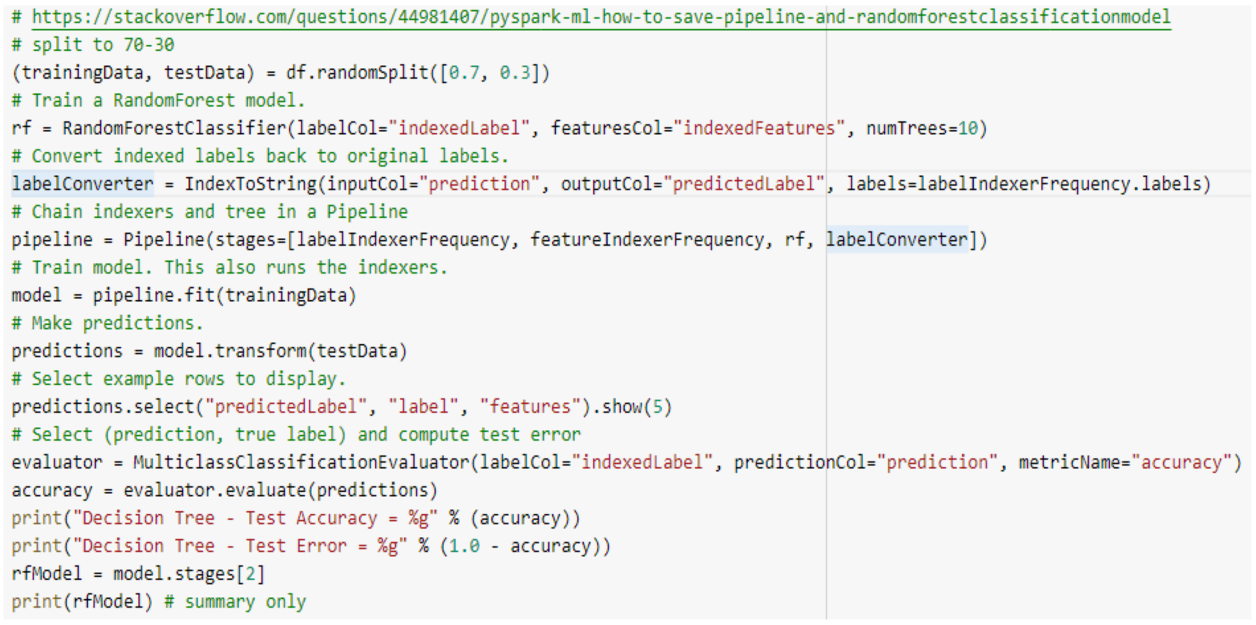
+ Random Forest

First, we create a new file name SparkRandomForest.ipynb and install Spark, define path for JAVA_HOME, SPARK_HOME and test if Spark have been installed successfully in google colab follow the same steps as the previous algorithm we did



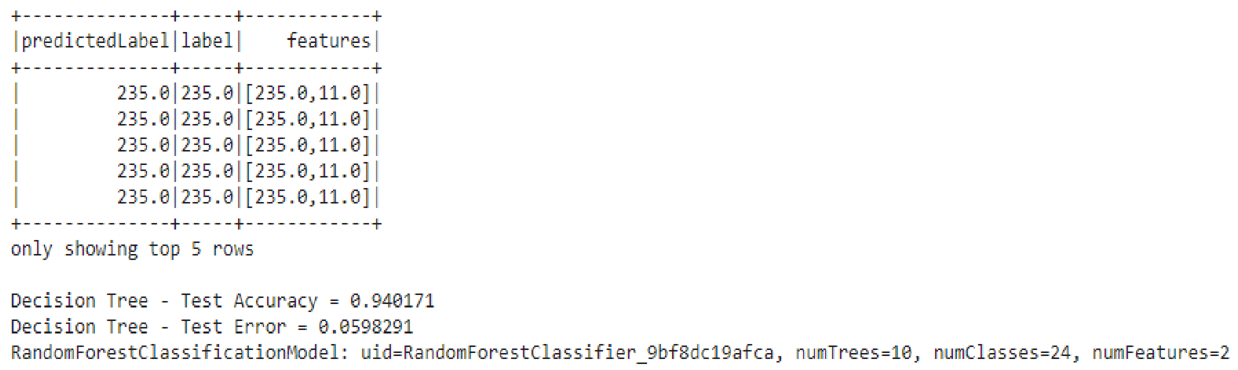
```
# read input and show
df = spark.read.options(delimiter=',', header=True).csv('/content/Absenteeism_at_work.csv')
df = df.withColumn("MOA", df["Month of absence"] - 0).withColumn("label", df['Transportation expense'] - 0).withColumn("ROA", df["Reason for absence"] - 0).\
    withColumn("distance", df["Distance from Residence to Work"] - 0).withColumn("BMI", df["Body mass index"] - 0)
df.show(5)
```

Next, we will choose ‘Transportation expense’ as label



```
# https://stackoverflow.com/questions/44981407/pyspark-ml-how-to-save-pipeline-and-randomforestclassificationmodel
# split to 70-30
(trainingData, testData) = df.randomSplit([0.7, 0.3])
# Train a RandomForest model.
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel", labels=labelIndexerFrequency.labels)
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexerFrequency, featureIndexerFrequency, rf, labelConverter])
# Train model. This also runs the indexers.
model = pipeline.fit(trainingData)
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("predictedLabel", "label", "features").show(5)
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Decision Tree - Test Accuracy = %g" % (accuracy))
print("Decision Tree - Test Error = %g" % (1.0 - accuracy))
rfModel = model.stages[2]
print(rfModel) # summary only
```

Finally, we also merge two column ‘label’ and ‘distance’ to a new column name ‘features’ and use two functions StringIndexer and VectorIndexer similarly as we implemented in Decision Tree algorithm. But in final step, after using RandomForestClassifider function to train a RandomForest model, we will use IndexToString function to convert index labels back to original label, add it to Pipeline, make the prediction and make the evaluation for multiclass classification



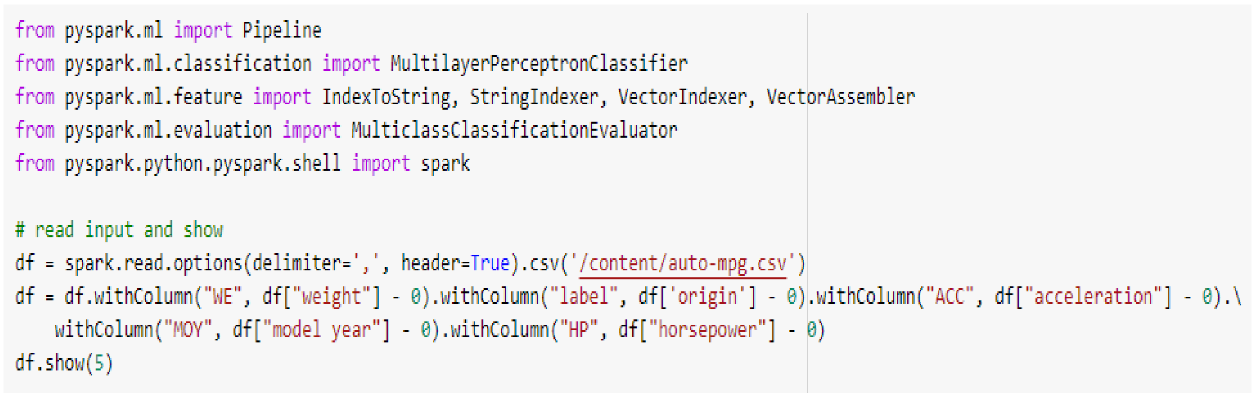
```
+--------------+-----+------------+
|predictedLabel|label|    features|
+--------------+-----+------------+
|         235.0|235.0|[235.0,11.0]|
|         235.0|235.0|[235.0,11.0]|
|         235.0|235.0|[235.0,11.0]|
|         235.0|235.0|[235.0,11.0]|
|         235.0|235.0|[235.0,11.0]|
+--------------+-----+------------+
only showing top 5 rows

Decision Tree - Test Accuracy = 0.940171
Decision Tree - Test Error = 0.0598291
RandomForestClassificationModel: uid=RandomForestClassifier_9bf8dc19afca, numTrees=10, numClasses=24, numFeatures=2
```

As we can see, our algorithm produces a fairly high accuracy result on the same dataset, just a little accuracy less than the Decision Tree algorithm

- Requirement 2:

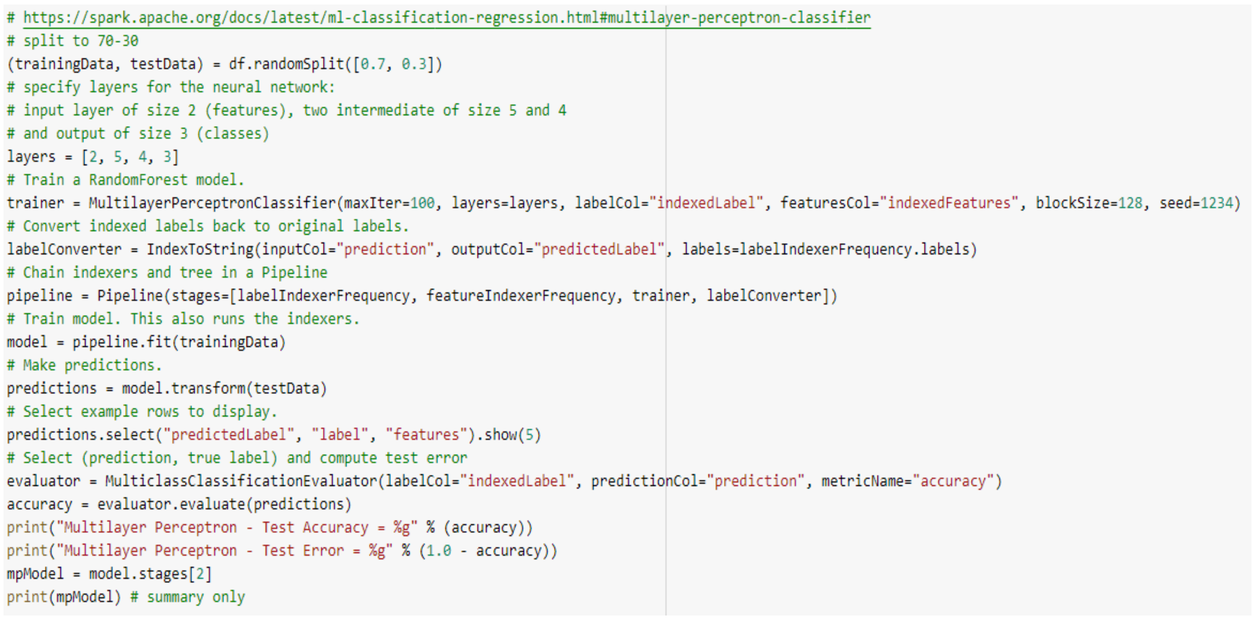
+ Spark Multilayer Perceptron

First, we create a new file name SparkMultilayerPerceptron.ipynb and install Spark, define path for JAVA_HOME, SPARK_HOME and test if Spark have been installed successfully in google colab follow the same steps as the previous algorithm we did. Then, we will use new dataset name cars.csv, this is the dataset about car and some its features such as model year, acceleration, weight…



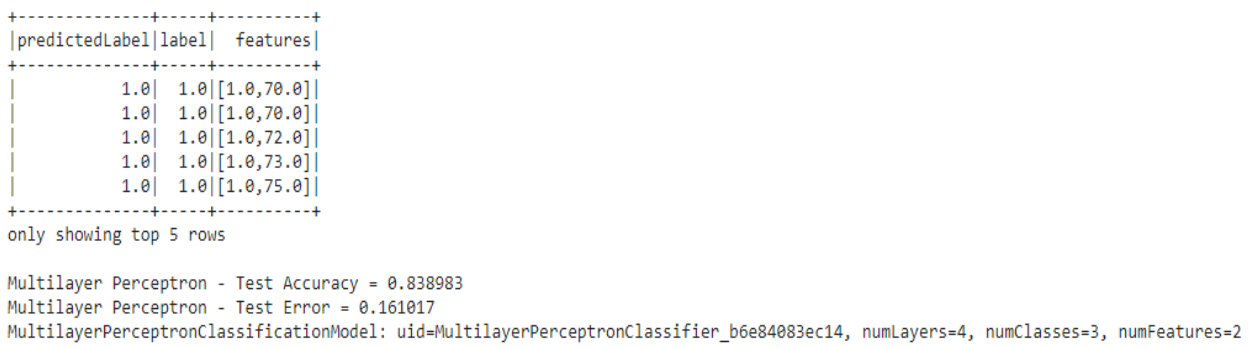
```python
from pyspark.ml import Pipeline
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer, VectorAssembler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.python.pyspark.shell import spark

# read input and show
df = spark.read.options(delimiter=',', header=True).csv('/content/auto-mpg.csv')
df = df.withColumn("WE", df["weight"] - 0).withColumn("label", df['origin'] - 0).withColumn("ACC", df["acceleration"] - 0).\
    withColumn("MOY", df["model year"] - 0).withColumn("HP", df["horsepower"] - 0)
df.show(5)
```

We also read the dataset and change columns from data type of string to double and merge two column ‘label’ and ‘MOY’ to new column name ‘features’ and use two functions StringIndexer and VectorIndexer similarly as we implemented in Decision Tree algorithm



```python
# https://spark.apache.org/docs/latest/ml-classification-regression.html#multilayer-perceptron-classifier
# split to 70-30
(trainingData, testData) = df.randomSplit([0.7, 0.3])
# specify layers for the neural network:
# input layer of size 2 (features), two intermediate of size 5 and 4
# and output of size 3 (classes)
layers = [2, 5, 4, 3]
# Train a RandomForest model.
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, labelCol="indexedLabel", featuresCol="indexedFeatures", blockSize=128, seed=1234)
# Convert indexed labels back to original labels.
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel", labels=labelIndexerFrequency.labels)
# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexerFrequency, featureIndexerFrequency, trainer, labelConverter])
# Train model. This also runs the indexers.
model = pipeline.fit(trainingData)
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select("predictedLabel", "label", "features").show(5)
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Multilayer Perceptron - Test Accuracy = %g" % (accuracy))
print("Multilayer Perceptron - Test Error = %g" % (1.0 - accuracy))
mpModel = model.stages[2]
print(mpModel) # summary only
```

Finally, we also split trainingData and testData with the ratio 70:30 and specify layer for the neural network, because we merge two columns to new column name ‘features’, so input layer will have size of 2, we define two intermediate of size 5 and 4 and output of size 3 correspond to our classes. We will use MultilayerPerceptronClassifier to train Multi Layer Perceptron model with default maxIter = 100, blockSize = 128 and seed = 1234, we can change these parameters to make suitable to our dataset, then do the rest of the steps in a similar way to the Random Forest algorithm we just did before

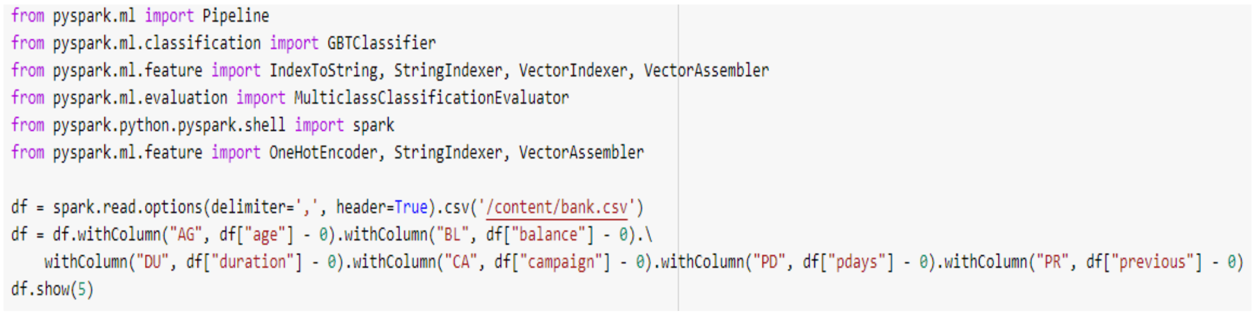


```
+--------------+-----+----------+
|predictedLabel|label|  features|
+--------------+-----+----------+
|           1.0|  1.0|[1.0,70.0]|
|           1.0|  1.0|[1.0,70.0]|
|           1.0|  1.0|[1.0,72.0]|
|           1.0|  1.0|[1.0,73.0]|
|           1.0|  1.0|[1.0,75.0]|
+--------------+-----+----------+
only showing top 5 rows

Multilayer Perceptron - Test Accuracy = 0.838983
Multilayer Perceptron - Test Error = 0.161017
MultilayerPerceptronClassificationModel: uid=MultilayerPerceptronClassifier_b6e84083ec14, numLayers=4, numClasses=3, numFeatures=2
```

As we can see, the result is not very high and maybe this dataset does not match our algorithm or the label we have chosen before is not really the best label

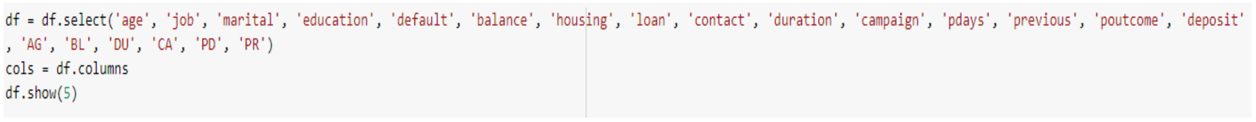
+ Gradient Boosted Tree

First, we create a new file name SparkGradientBoostedTree.ipynb and install Spark, define path for JAVA_HOME, SPARK_HOME and test if Spark have been installed successfully in google colab follow the same steps as the previous algorithm we did. Then, we will use new dataset name bank.csv, this is the dataset about information of a people such as job, age, martial, balance, loan…

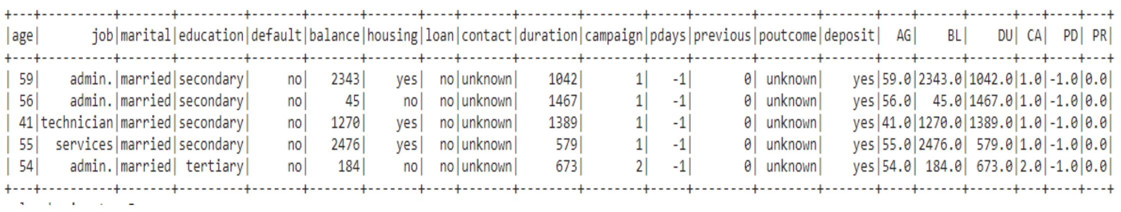


```python
from pyspark.ml import Pipeline
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer, VectorAssembler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.python.pyspark.shell import spark
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler

df = spark.read.options(delimiter=',', header=True).csv('/content/bank.csv')
df = df.withColumn("AG", df["age"] - 0).withColumn("BL", df["balance"] - 0).\
    withColumn("DU", df["duration"] - 0).withColumn("CA", df["campaign"] - 0).withColumn("PD", df["pdays"] - 0).withColumn("PR", df["previous"] - 0)
df.show(5)
```

Next, we will read the dataset and convert columns from data type of string to int



```python
df = df.select('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'deposit'
, 'AG', 'BL', 'DU', 'CA', 'PD', 'PR')
cols = df.columns
df.show(5)
```



```
+---+----------+-------+---------+-------+-------+-------+----+-------+--------+--------+-----+--------+--------+-------+----+------+------+---+----+---+
|age|       job|marital|education|default|balance|housing|loan|contact|duration|campaign|pdays|previous|poutcome|deposit|  AG|    BL|    DU| CA|  PD| PR|
+---+----------+-------+---------+-------+-------+-------+----+-------+--------+--------+-----+--------+--------+-------+----+------+------+---+----+---+
| 59|    admin.|married|secondary|     no|   2343|    yes|  no|unknown|    1042|       1|   -1|       0| unknown|    yes|59.0|2343.0|1042.0|1.0|-1.0|0.0|
| 56|    admin.|married|secondary|     no|     45|     no|  no|unknown|    1467|       1|   -1|       0| unknown|    yes|56.0|  45.0|1467.0|1.0|-1.0|0.0|
| 41|technician|married|secondary|     no|   1270|    yes|  no|unknown|    1389|       1|   -1|       0| unknown|    yes|41.0|1270.0|1389.0|1.0|-1.0|0.0|
| 55|  services|married|secondary|     no|   2476|    yes|  no|unknown|     579|       1|   -1|       0| unknown|    yes|55.0|2476.0| 579.0|1.0|-1.0|0.0|
| 54|    admin.|married| tertiary|     no|    184|     no|  no|unknown|     673|       2|   -1|       0| unknown|    yes|54.0| 184.0| 673.0|2.0|-1.0|0.0|
+---+----------+-------+---------+-------+-------+-------+----+-------+--------+--------+-----+--------+--------+-------+----+------+------+---+----+---+
```

Because two columns ‘day’ and ‘month’ are not necessary, we will remove them

```
categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'poutcome']
stages = []
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol='indexed' + categoricalCol)
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]
labelIndexerFrequency = StringIndexer(inputCol = 'deposit', outputCol = 'label')
stages += [labelIndexerFrequency]
numericCols = ['AG', 'BL', 'DU', 'CA', 'PD', 'PR']
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

We will use StringIndexer to take categoryIndex with its frequency from 0.0 (most frequency) to n (least frequency) as we have explained before, then convert the indexed categories into one-hot encoded variables and so the result will have the binary vectors appended to the end of each row. We re-use StringIndexer to encode our labels to label indices and use VectorAssembler to combine multi columns into a new column name ‘features’

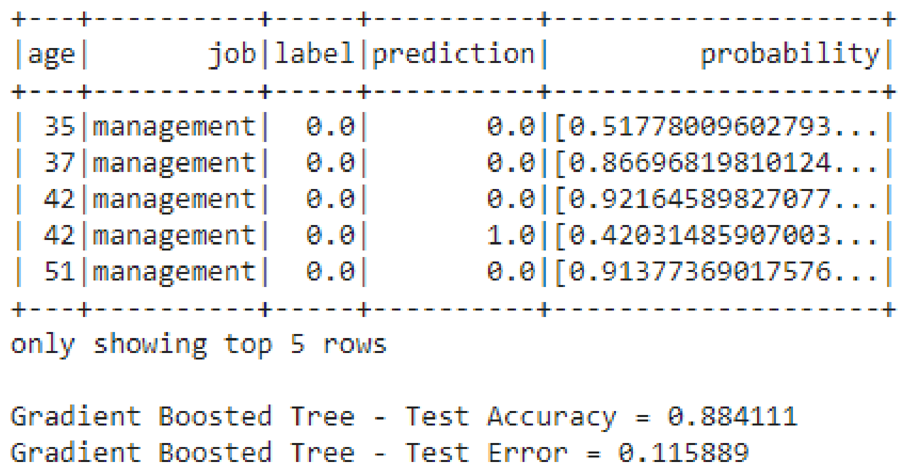
```
from pyspark.ml import Pipeline

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
selectedCols = ['label', 'features'] + cols
df = df.select(selectedCols)
df.show(5)
```

```
+-----+--------------------+---+----------+-------+---------+-------+-------+-------+----+-------+--------+--------+-----+--------+--------+-------+----+------+------+---+----+---+
|label|            features|age|       job|marital|education|default|balance|housing|loan|contact|duration|campaign|pdays|previous|poutcome|deposit|  AG|    BL|    DU| CA|  PD| PR|
+-----+--------------------+---+----------+-------+---------+-------+-------+-------+----+-------+--------+--------+-----+--------+--------+-------+----+------+------+---+----+---+
|  1.0|(30,[3,11,13,16,1...| 59|    admin.|married|secondary|     no|   2343|    yes|  no|unknown|    1042|       1|   -1|       0| unknown|    yes|59.0|2343.0|1042.0|1.0|-1.0|0.0|
|  1.0|(30,[3,11,13,16,1...| 56|    admin.|married|secondary|     no|     45|     no|  no|unknown|    1467|       1|   -1|       0| unknown|    yes|56.0|  45.0|1467.0|1.0|-1.0|0.0|
|  1.0|(30,[2,11,13,16,1...| 41|technician|married|secondary|     no|   1270|    yes|  no|unknown|    1389|       1|   -1|       0| unknown|    yes|41.0|1270.0|1389.0|1.0|-1.0|0.0|
|  1.0|(30,[4,11,13,16,1...| 55|  services|married|secondary|     no|   2476|    yes|  no|unknown|     579|       1|   -1|       0| unknown|    yes|55.0|2476.0| 579.0|1.0|-1.0|0.0|
|  1.0|(30,[3,11,14,16,1...| 54|    admin.|married| tertiary|     no|    184|     no|  no|unknown|     673|       2|   -1|       0| unknown|    yes|54.0| 184.0| 673.0|2.0|-1.0|0.0|
+-----+--------------------+---+----------+-------+---------+-------+-------+-------+----+-------+--------+--------+-----+--------+--------+-------+----+------+------+---+----+---+
only showing top 5 rows
```

After that, we use pipeline to chain transformers and estimators together to specify our machine learning workflow and its stages is defined as an ordered array

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-classifier
# split to 70-30
(trainingData, testData) = df.randomSplit([0.7, 0.3], seed=2021)
# Train a Gradient Boosted Tree model
gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)
# Train model. This also runs the indexers.
model = gbt.fit(trainingData)
# Make predictions.
predictions = model.transform(testData)
# Select example rows to display.
predictions.select('age', 'job', 'label', 'prediction', 'probability').show(5)
evaluator = BinaryClassificationEvaluator()
accuracy = evaluator.evaluate(predictions)
print("Gradient Boosted Tree - Test Accuracy = %g" % (accuracy))
print("Gradient Boosted Tree - Test Error = %g" % (1.0 - accuracy))
```

Finally, we use GBTClassifier function to train our Gradient Boosted Tree model with default maxIter=10 and because our algorithm just works on binary dataset, so we will use BinaryClassifcationEvaluator function to make the evaluation instead using MulticlassClassificationEvaluator



```
+---+----------+-----+----------+--------------------+
|age|       job|label|prediction|         probability|
+---+----------+-----+----------+--------------------+
| 35|management|  0.0|       0.0|[0.51778009602793...|
| 37|management|  0.0|       0.0|[0.86696819810124...|
| 42|management|  0.0|       0.0|[0.92164589827077...|
| 42|management|  0.0|       1.0|[0.42031485907003...|
| 51|management|  0.0|       0.0|[0.91377369017576...|
+---+----------+-----+----------+--------------------+
only showing top 5 rows

Gradient Boosted Tree - Test Accuracy = 0.884111
Gradient Boosted Tree - Test Error = 0.115889
```

As we can see, our algorithm gives no bad accuracy on this dataset

3. **Self-evaluation**: The team finished the job fully but the two supported classification algorithms in the Spark were the Gradient Boosted Tree and the Multi Layer Perceptron which were new, so the team had a bit of difficulty learning to run these two algorithms successfully
4. **References**:

- Code reference:

[1]: https://github.com/Ruthvicp/CS5590_BigDataProgramming/wiki/Lab-Assignment-4----Spark-MLlib-classification-algorithms,-word-count-on-twitter-streaming

[2]: https://www.analyticsvidhya.com/blog/2020/11/a-must-read-guide-on-how-to-work-with-pyspark-on-google-colab-for-data-scientists/

[3]: https://spark.apache.org/docs/latest/ml-features#vectorassembler

[4]: https://spark.apache.org/docs/latest/ml-features#stringindexer

[5]: https://spark.apache.org/docs/latest/ml-features#vectorindexer

[6]: https://spark.apache.org/docs/1.5.2/ml-decision-tree.html

[7]: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3741049972324885/3783546674231736/4413065072037724/latest.html

[8]: https://stackoverflow.com/questions/44981407/pyspark-ml-how-to-save-pipeline-and-randomforestclassificationmodel

[9]: https://spark.apache.org/docs/latest/ml-classification-regression.html#multilayer-perceptron-classifier

[10]: https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-classifier

[11]: https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa

- Dataset:

[1]: https://drive.google.com/drive/folders/1jMAplUtpzPvnKdnzUSKMiTTTsPNPlAX0

[2]: https://www.kaggle.com/rouseguy/bankbalanced/data