PySpark-Big Data

July 25, 2022

1 Part i: Medical_Analysis Project

[1]:
```
# starting a SparkSession and creating a spark instance
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

[1]:
```
'C:\\spark-3.0.3-bin-hadoop2.7'
```

[2]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Big Data Analytics').getOrCreate()
print ("*** Application name: " + spark.sparkContext.appName + " / Version: " +␣
 ↪spark.version + " ***")
```

```
*** Application name: Big Data Analytics / Version: 3.0.3 ***
```

[3]:
```
from itertools import chain
from pyspark.sql.functions import count, mean, when, lit, create_map,␣
 ↪regexp_extract
```

1. Load the data file into a Spark DataFrame (1st DataFrame,df1). Describing the structure of the DataFrame.

[4]:
```
df1 = spark.read.option('header','true').csv('D:\\Datascience\\Medical_info.
 ↪csv', inferSchema=True)
```

[5]:
```
df1.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- BMI: double (nullable = true)
 |-- PSA: string (nullable = true)
 |-- TG: integer (nullable = true)
 |-- Cholesterol: string (nullable = true)
 |-- LDLChole: integer (nullable = true)
 |-- HDLChole: integer (nullable = true)
```

```
 |-- Glucose: string (nullable = true)
 |-- Testosterone: double (nullable = true)
 |-- BP_1: integer (nullable = true)
```

[6]:
```
# Showing the Dataset
df1.show(4)
```

```
+--------+----+------+----+----+----------+--------+--------+-------+----------
--+----+
|      id| age|   BMI| PSA|
TG|Cholesterol|LDLChole|HDLChole|Glucose|Testosterone|BP_1|
+--------+----+------+----+----+----------+--------+--------+-------+----------
--+----+
|    null|null|  null|null|null|       null|    null|    null|   null|
null|null|
|19782173|  59|28.378|0.34| 204|        196|     132|      49|     92|
7.7|   1|
|    null|null|  null|null|null|       null|    null|    null|   null|
null|null|
|32613511|  59|24.968|   1| 147|        181|     129|      34|     96|
4.09|   1|
+--------+----+------+----+----+----------+--------+--------+-------+----------
--+----+
only showing top 4 rows
```

2. Create a new DataFrame (2 nd DataFrame) by removing all the rows with null/missing values in the 1 st DataFrame and calculate the number of rows removed.

[7]:
```
# making a new dataframe and Removing  rows with null values
df2 = df1.na.drop(how="all")
```

[8]:
```
# displaying the Values
df2.show(4)
```

```
+--------+---+------+----+---+----------+--------+--------+-------+-----------
+----+
|      id|age|   BMI| PSA|
TG|Cholesterol|LDLChole|HDLChole|Glucose|Testosterone|BP_1|
+--------+---+------+----+---+----------+--------+--------+-------+-----------
+----+
|19782173| 59|28.378|0.34|204|        196|     132|      49|     92|
7.7|   1|
|32613511| 59|24.968|   1|147|        181|     129|      34|     96|
4.09|   1|
|32723850| 48|31.307|0.62|155|        185|     127|      41|    139|
4.5|   1|
|22913531| 47|27.837|0.38|488|        254|     158|      55|    250|
```

```
5.3|   2|
+--------+---+------+----+---+-----------+--------+--------+-------+------------
+----+
only showing top 4 rows
```

[9]:
```
# number of rows in the after removing missing values
print('Number of rows before removing missing values: \t', df1.count())

# number of rows after before removing missing values
print('Number of rows after removing missing values: \t', df2.count())

# number of rows after before removing missing values
print('Number of rows removed: \t', df1.count()-df2.count())
```

```
Number of rows before removing missing values:   13934
Number of rows after removing missing values:    6967
Number of rows removed:          6967
```

3. Calculate summary statistics of the 'age' feature in the 2nd DataFrame, including its min value, maxvalue, mean value, median value, variance and standard deviation. Generate a histogram for the 'age' feature and describe the distribution of the feature.

[10]:
```
from pyspark.sql.functions import variance
df2.select('Age').summary('min','max','mean','50%','stddev').show()
df2.select(variance("Age")).show() # variance of feature Age'
```

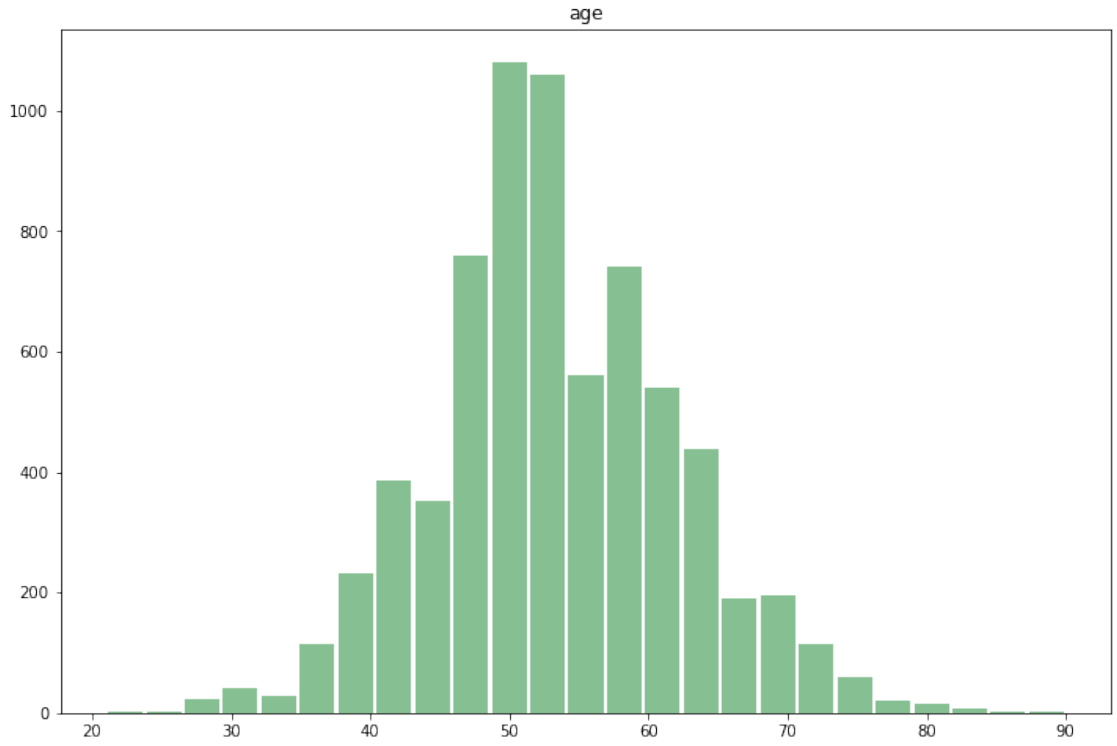
```
+-------+-----------------+
|summary|              Age|
+-------+-----------------+
|    min|               21|
|    max|               90|
|   mean|53.33156308310607|
|    50%|               53|
| stddev|8.715031757570447|
+-------+-----------------+

+-----------------+
|    var_samp(Age)|
+-----------------+
|75.95177853546144|
+-----------------+
```

[11]:
```
import pandas as pd
```

[12]:
```
df2 = df2.toPandas()
```

[13]:
```
#  Visualization of Data ( second dataframe)
df2.hist(column='age', bins=25, grid=False, figsize=(12,8), color='#86bf91',␣
 ↪zorder=2, rwidth=0.9)
```

[13]:
```
array([[<AxesSubplot:title={'center':'age'}>]], dtype=object)
```



[14]:
```
df2 = spark.read.option('header','true').csv('D:\\Datascience\\Medical_info.
 ↪csv', inferSchema=True)
type(df2)
```

[14]:
```
pyspark.sql.dataframe.DataFrame
```

[ ]:
```

```

4. Display the quartile info of the 'BMI' feature in the 2nd DataFrame. Generate a boxplot for the 'BMI'feature and discuss the distribution of the feature based on the boxplot.

[15]:
```
# Displaying the quartile info BMI
df2.select('BMI').summary('25%','50%','75%').show()
```

```
+-------+------+
|summary|   BMI|
+-------+------+
|    25%|22.976|
```

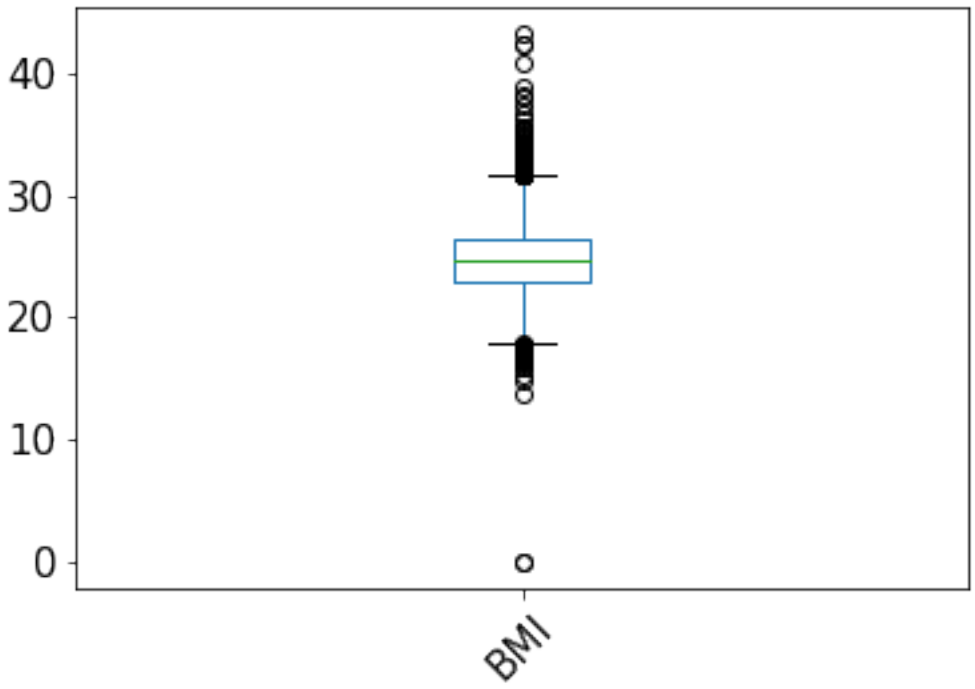
```
|    50%|24.696|
|    75%|26.435|
+-------+------+
```

[16]:
```
# Data visualization-Boxplot
import pandas as pd
```

[17]:
```
df2 = df2.toPandas()
```

[18]:
```
df2.boxplot(column='BMI',grid=False, rot=45,fontsize=15)
```

[18]: <AxesSubplot:>



[19]:
```
df2 = spark.read.option('header','true').csv('D:\\Datascience\\Medical_info.
 ↪csv', inferSchema=True)
type(df2)
```

[19]: pyspark.sql.dataframe.DataFrame

5. Use Spark DataFrame API (i.e., expression methods) to count the number of rows where 'age' is greater than 50 and 'BP_1' equals 1.

[20]:
```
# Creating  temporary table of spark Dataframe to view Age and BP_1
df2.createOrReplaceTempView("Expression")
```

```
spark.sql("SELECT Age, BP_1 from EXpression")
```

[20]: DataFrame[Age: int, BP_1: int]

[21]:
```
spark.sql("SELECT Age from EXpression WHERE Age>50").count()
```

[21]: 4331

[22]:
```
spark.sql("SELECT BP_1 from EXpression WHERE BP_1=1").count()
```

[22]: 3900

[23]:
```
# number of rows where 'age' is greater than 50 and 'BP_1' equals 1
spark.sql("SELECT * from EXpression WHERE Age>50 AND BP_1=1").count()
```

[23]: 2182

6.Use the 'BP_1' feature in the 2 nd DataFrame as the target label, to build two classification models based on all other columns as predictors. Conduct performance evaluation for the two models and make conclusions

[24]:
```
df2.na.drop("all").show(5)
# Check for duplicates
print('Count of rows: {0}'.format(df2.count()))
print('Count of distinct rows: {0}'.format(df2.distinct().count()))
```

```
+--------+---+------+----+---+----------+--------+--------+-------+-----------
+----+
|      id|age|   BMI| PSA|
TG|Cholesterol|LDLChole|HDLChole|Glucose|Testosterone|BP_1|
+--------+---+------+----+---+----------+--------+--------+-------+-----------
+----+
|19782173| 59|28.378|0.34|204|       196|     132|      49|     92|
7.7|   1|
|32613511| 59|24.968|   1|147|       181|     129|      34|     96|
4.09|   1|
|32723850| 48|31.307|0.62|155|       185|     127|      41|    139|
4.5|   1|
|22913531| 47|27.837|0.38|488|       254|     158|      55|    250|
5.3|   2|
|32628551| 55|22.662|0.49| 87|       175|     120|      44|     99|
6.9|   1|
+--------+---+------+----+---+----------+--------+--------+-------+-----------
+----+
only showing top 5 rows

Count of rows: 13934
Count of distinct rows: 6968
```

[25]:
```
# Grouping the stringType columns
df2.groupBy("PSA","Cholesterol","Glucose").count().show(5)
```

```
+----+-----------+-------+-----+
| PSA|Cholesterol|Glucose|count|
+----+-----------+-------+-----+
| 2.1|        180|    112|    1|
| 1.1|        164|     96|    1|
|0.34|        171|     87|    1|
| 1.2|        220|    103|    1|
|0.35|        171|     98|    1|
+----+-----------+-------+-----+
only showing top 5 rows
```

[26]:
```
# Checking statistical summary
df2.select('PSA', 'Cholesterol','Glucose').summary('mean', '50%', 'max').show()
```

```
+-------+------------------+------------------+------------------+
|summary|               PSA|       Cholesterol|           Glucose|
+-------+------------------+------------------+------------------+
|   mean|1.0225676653640172|194.19178868791272|106.40123456790124|
|    50%|              0.76|             193.0|             101.0|
|    max|               9.3|                98|                99|
+-------+------------------+------------------+------------------+
```

[27]:
```
# counting the number of null Values.
for col in df2.columns:
    print(col.ljust(20), df2.filter(df2[col].isNull()).count())
```

```
id                   6967
age                  6967
BMI                  6967
PSA                  6967
TG                   6967
Cholesterol          6967
LDLChole             6967
HDLChole             6967
Glucose              6967
Testosterone         6967
BP_1                 6967
```

[28]:
```
df2.select('BP_1').summary('mean', '50%','max','stddev').show()
```

```
+-------+------------------+
|summary|              BP_1|
+-------+------------------+
|   mean| 1.4402181713793598|
```

```
|    50%|                  1|
|    max|                  2|
| stddev|0.49644889820019666|
+-------+-------------------+
```

[29]:
```
df2 = df2.na.fill(1.5) #filling the missing Value
```

[30]:
```
# dropping the columns with the highest number of mising values
df2 = df2.drop('PSA').drop('Cholesterol').drop('Testosterone').drop('Glucose')
```

[31]:
```
for col in df2.columns:
    print(col.ljust(20), df2.filter(df2[col].isNull()).count()) #checking for␣
 ↪the missing value
```

```
id                   0
age                  0
BMI                  0
TG                   0
LDLChole             0
HDLChole             0
BP_1                 0
```

Building Classification model

[32]:
```
# String Indexer
# We will convert STRING columnS to numeric index.
# This creates a new column for numeric leaving the original intact.
# So we will remove them afterward.
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression,\
                    RandomForestClassifier, GBTClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

[33]:
```
df2
```

[33]:
```
DataFrame[id: int, age: int, BMI: double, TG: int, LDLChole: int, HDLChole: int,
BP_1: int]
```

[34]:
```
# Vector Assembly
# Using VectorAssembler In Spark DataFrame
from pyspark.ml.linalg import Vector
from pyspark.ml.feature import VectorAssembler
```

[35]:
```
# Merging predictors  columns that are doing to be merged to make a vector in␣
 ↪each row and outputCol is the name of the merged column
vec_asmbl = VectorAssembler(inputCols=df2.columns[:7],
```

```
                            outputCol='features')

df2 = vec_asmbl.transform(df2).select('features', 'BP_1')
df2.show(4, truncate=False)
```

```
+---------------------------------------------+----+
|features                                     |BP_1|
+---------------------------------------------+----+
|[1.0,1.0,1.5,1.0,1.0,1.0,1.0]                |1   |
|[1.9782173E7,59.0,28.378,204.0,132.0,49.0,1.0]|1   |
|[1.0,1.0,1.5,1.0,1.0,1.0,1.0]                |1   |
|[3.2613511E7,59.0,24.968,147.0,129.0,34.0,1.0]|1   |
+---------------------------------------------+----+
only showing top 4 rows
```

[36]:
```
#Now we split the training data into the train and test part(0.8, 0.2
 ↪respectively)
train_data, test_data= df2.randomSplit([0.7, 0.3])
```

[37]:
```
train_data.show(4, truncate=False)
```

```
+-----------------------------+----+
|features                     |BP_1|
+-----------------------------+----+
|[1.0,1.0,1.5,1.0,1.0,1.0,1.0]|1   |
|[1.0,1.0,1.5,1.0,1.0,1.0,1.0]|1   |
|[1.0,1.0,1.5,1.0,1.0,1.0,1.0]|1   |
|[1.0,1.0,1.5,1.0,1.0,1.0,1.0]|1   |
+-----------------------------+----+
only showing top 4 rows
```

[38]:
```
performance = MulticlassClassificationEvaluator(labelCol='BP_1',
                                                 metricName='accuracy')
```

Classification model_1: LinearRegression

[39]:
```
from pyspark.ml.regression import LinearRegression
reg=LinearRegression(featuresCol='features', labelCol='BP_1')
reg=reg.fit(train_data)
pred = reg.transform(test_data)
performance.evaluate(pred) # performance of model linear regression
```

[39]:
```
0.029356505401596993
```

Classication Model_2:RandomForestClassifier

[40]:
```
random = RandomForestClassifier(labelCol='BP_1',
                            numTrees=50, maxDepth=3)

model = random.fit(train_data)
pred = model.transform(test_data)
performance.evaluate(pred) # performance of model 2 RandomForest
```

[40]: 1.0

Performance Evaluation

[41]:
```
pred=reg.evaluate(test_data)
```

[42]:
```
pred.predictions.show(5)
```

```
+--------------------+----+------------------+
|            features|BP_1|        prediction|
+--------------------+----+------------------+
|[1.0,1.0,1.5,1.0,…|   1|0.9999999999999999|
|[1.0,1.0,1.5,1.0,…|   1|0.9999999999999999|
|[1.0,1.0,1.5,1.0,…|   1|0.9999999999999999|
|[1.0,1.0,1.5,1.0,…|   1|0.9999999999999999|
|[1.0,1.0,1.5,1.0,…|   1|0.9999999999999999|
+--------------------+----+------------------+
only showing top 5 rows
```

[43]:
```
pred.meanAbsoluteError, pred.meanSquaredError
```

[43]: (1.9844780273419602e-16, 5.494572291128474e-32)

[44]:
```
#Getting the set of coefficients and intercepsts.
reg.coefficients
```

[44]: DenseVector([-0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 1.0])

[45]:
```
reg.intercept
```

[45]: 4.85786891647014e-16

[ ]:

[ ]: