Missing values - setup notebook

In [2]:
```
#import findspark
#findspark.init()
import pyspark
import random
```

In [3]:
```
from pyspark import SparkContext
sc = SparkContext()
from pyspark.sql import SQLContext
sqlc = SQLContext(sc)
```

In [39]:
```
df = sqlc.read.format('com.databricks.spark.csv').options(header = 'true',in
ferschema = 'true').load('iris_missing.csv')
```

In [40]:
```
df.show(5)
```

```
+-----------+----------+-----------+----------+-------+
|Sepallength|Sepalwidth|Petallength|Petalwidth|Species|
+-----------+----------+-----------+----------+-------+
|        5.1|       3.5|        1.4|       0.2| setosa|
|        4.9|       3.0|        1.4|       0.2| setosa|
|        4.7|       3.2|        1.3|       0.2| setosa|
|        4.6|       3.1|        1.5|       0.2| setosa|
|        5.0|       3.6|        1.4|       0.2| setosa|
+-----------+----------+-----------+----------+-------+
only showing top 5 rows
```

Exercise 38: Counting Missing Values in a DataFrame

In this exercise, we will learn how to count the missing values from the PySpark DataFrame column.

1. Load libraries:

In [87]:
```
from pyspark.sql.functions import isnan, when, count, col
```

2. Now, we will count the missing values in the Sepallength column of the Iris dataset loaded in the PySpark DataFrame df object.

In [96]:
```
df.filter(col('Sepallength').isNull()).count()
```

Out[96]:
```
5
```

In [97]:
```
df.filter(isnan('Sepallength')).count()
```

Out[97]:
```
0
```

when(), otherwise(), alias() functions

In [161]:
```
c='Sepalwidth'
#df.show()
#when(col(c)>3.0,'BIG') #returns column
#df.select(when(col(c)>3.0,'BIG')).show() #returns and shows dataframe
#df.select(when(col(c)>3.0,'BIG').otherwise('SMALL')).show()
#df.select(when((col(c)<4.0)&(col(c)>3.0),'BIG').when(col(c)>=4.0,'HUGE').ot
herwise('SMALL')).show()
#df.select(when((col(c)<4.0)&(col(c)>3.0),'BIG').when(col(c)>=4.0,'HUGE').ot
herwise('SMALL').alias(c)).show()
#df.select(count(when(col(c)>3.0,'BIG'))).show()
#df.select(count(when(col(c)>3.0,'BIG').alias(c))).show()

#for selecting you can ue Spark columns list:
#col1 = when(col(c)>3.0,'BIG').otherwise(0)
#col2 = when(col(c)<3.0,'SMALL').otherwise(0)
#df.select([col1,col2]).show()

#TRY to do Ex39 yourself as Activity
```

Exercise 39: Counting Missing Values in All DataFrame Columns

In this exercise, we will count the missing values present in all the columns of a PySpark DataFrame.

1. First import modules and show the data.

In [162]:
```
from pyspark.sql.functions import isnan, when, count, col
df.select([count(when(isnan(c) | col(c).isNull(),c)).alias(c) for c in df.co
lumns]).show()
```

```
+-----------+----------+-----------+----------+-------+
|Sepallength|Sepalwidth|Petallength|Petalwidth|Species|
+-----------+----------+-----------+----------+-------+
|          5|         4|          3|         3|      0|
+-----------+----------+-----------+----------+-------+
```

The output hows missing entries for each column.

2. A simple way is to just use the describe() function, which gives the count of non-missing values for each column, along with a bunch of other summary statistics. Let's execute the following command in the notebook.

In [44]:
```
#df.describe().show()
df.describe().show(1)
```

```
+-------+-----------+----------+-----------+----------+-------+
|summary|Sepallength|Sepalwidth|Petallength|Petalwidth|Species|
+-------+-----------+----------+-----------+----------+-------+
|  count|        145|       146|        147|       147|    150|
+-------+-----------+----------+-----------+----------+-------+
only showing top 1 row
```

Fetching Missing Value Records from the DataFrame

We can also filter out the records containing the missing value entries from the PySpark DataFrame using the following code:

In [45]:
```
df.where(col('Sepallength').isNull()).show()
```

```
+-----------+----------+-----------+----------+----------+
|Sepallength|Sepalwidth|Petallength|Petalwidth|   Species|
+-----------+----------+-----------+----------+----------+
|       null|       2.4|        3.7|       1.0|versicolor|
|       null|       3.1|        4.7|       1.5|versicolor|
|       null|      null|       null|       1.2|versicolor|
|       null|       3.8|        6.7|      null| virginica|
|       null|      null|       null|      null| virginica|
+-----------+----------+-----------+----------+----------+
```

Handling Missing Values in Spark DataFrames

In [ ]:

Exercise 40: Removing Records with Missing Values from a DataFrame

In this exercise, we will remove the records containing missing value entries for the PySpark DataFrame. Let's perform the following steps.

<To remove the missing values from a particular column, use the following command:/p>

1. To remove the missing values from a particular column, use the following command:

In [166]:
```
df.select('Sepallength').dropna().count()
```

Out[166]:
```
145
```

2. To remove all the records containing any missing value entry for any column from the PySpark DataFrame, use the following command:

In [167]:
```
df.dropna().count()
```

Out[167]:
```
143
```

Exercise 41: Filling Missing Values with a Constant in a DataFrame Column

In this exercise, we will replace the missing value entries of the PySpark DataFrame column with a constant numeric value.

1. Let's replace the missing value entries in two columns with a constant numeric value of 1.

In [173]:
```
colNames = ['Sepallength','Sepalwidth']
y = df.select(colNames).fillna(1)
```

2. Now, let's count the missing values in the new DataFrame, y, that we just created. The new DataFrame should have no missing values:

In [174]:
```
y.select([count(when(isnan(i) | col(i).isNull(), i)).alias(i) for i in y.col
umns]).show()
```

```
+-----------+----------+
|Sepallength|Sepalwidth|
+-----------+----------+
|          0|         0|
+-----------+----------+
```

3. Use the following command to replace all the missing values in the PySpark DataFrame with a constant numeric value of 1:

In [175]:
```
z = df.fillna(1)
```

4. Check your job.

In [177]:
```
z.select([count(when(isnan(k) | col(k).isNull(), k)).alias(k) for k in z.col
umns]).show()
```

```
+-----------+----------+-----------+----------+-------+
|Sepallength|Sepalwidth|Petallength|Petalwidth|Species|
+-----------+----------+-----------+----------+-------+
|          0|         0|          0|         0|      0|
+-----------+----------+-----------+----------+-------+
```

Correlation

Correlation is a statistical measure of the level of association between two numerical variables. It gives us an idea of how closely two variables are related with each other. For example, age and income are quite closely related variables. It has been observed that the average income grows with age within a threshold. Thus, we can assume that age and income are positively correlated with each other.

The most common metric used to compute this association is the Pearson Product- Moment Correlation, commonly known as Pearson correlation coefficient or simply as the correlation coefficient. It is named after its inventor, Karl Pearson.

The Pearson correlation coefficient is computed by dividing the covariance of the two variables by the product of their standard deviations. The correlation value lies between -1 and +1 with values close to 1 or -1 signifying strong association and values close to 0, signifying weak association. The sign (+, -) of the coefficient tells us whether the association is positive (both variables increase/decrease together) or negative (vice- versa).

Correlation is of great importance in statistical analysis, as it helps explain the data and sometimes highlights predictive relationships between variables. We will learn how to compute correlation between variables and compute a correlation matrix in PySpark.

Exercise 42: Computing Correlation

In this exercise, we will compute the value of the Pearson correlation coefficient between two numerical variables and a correlation matrix for all the numerical columns of our PySpark DataFrame. The correlation matrix helps us visualize the correlation of all the numerical columns with each other:

1. Perform the following steps to calculate the correlation between two variables:

```
In [203]: df.corr('Sepallength', 'Sepalwidth')
Out[203]: 0.150594543846508
```

2. Import the relevant modules, as illustrated here:

```
In [182]: from pyspark.mllib.stat import Statistics
          import pandas as pd
```

3. Remove any missing values from the data with the following command:

```
In [184]: z=df.fillna(1)
```

4. To remove any non-numerical columns before computing the correlation matrix, use the following command:

In [194]:
```
a = z.drop('Species')
a.show()
```

```
+-----------+----------+-----------+----------+
|Sepallength|Sepalwidth|Petallength|Petalwidth|
+-----------+----------+-----------+----------+
|        5.1|       3.5|        1.4|       0.2|
|        4.9|       3.0|        1.4|       0.2|
|        4.7|       3.2|        1.3|       0.2|
|        4.6|       3.1|        1.5|       0.2|
|        5.0|       3.6|        1.4|       0.2|
|        5.4|       3.9|        1.7|       0.4|
|        4.6|       3.4|        1.4|       0.3|
|        5.0|       3.4|        1.5|       0.2|
|        4.4|       2.9|        1.4|       0.2|
|        4.9|       3.1|        1.5|       0.1|
|        5.4|       3.7|        1.5|       0.2|
|        4.8|       3.4|        1.6|       0.2|
|        4.8|       3.0|        1.4|       0.1|
|        4.3|       3.0|        1.1|       0.1|
|        5.8|       4.0|        1.2|       0.2|
|        5.7|       4.4|        1.5|       0.4|
|        5.4|       3.9|        1.3|       0.4|
|        5.1|       3.5|        1.4|       0.3|
|        5.7|       3.8|        1.7|       0.3|
|        5.1|       3.8|        1.5|       0.3|
+-----------+----------+-----------+----------+
only showing top 20 rows
```

Notice, that we will get different correlation coefficient, when missing values are removed:

In [204]:
```
a.corr('Sepallength', 'Sepalwidth')
```

Out[204]:
```
0.06486651076535906
```

5. Now, let's compute the correlation matrix with the help of the following command:

In [193]:
```
features=a.rdd.map(lambda row: row[0:])
#sqlc.createDataFrame(features).show()
```

In [197]:
```
correlation_matrix = Statistics.corr(features, method="pearson")
```

6.To convert the matrix into a pandas DataFrame for easy visualization, execute the following command:

In [198]:
```
correlation_df = pd.DataFrame(correlation_matrix)
```

7. Rename the indexes of the pandas DataFrame with the name of the columns from the original PySpark DataFrame:

In [201]:
```
correlation_df.index, correlation_df.columns = a.columns, a.columns
```

```
In [202]: correlation_df
```

Out[202]:

	Sepallength	Sepalwidth	Petallength	Petalwidth
Sepallength	1.000000	0.064867	0.592412	0.549628
Sepalwidth	0.064867	1.000000	-0.264379	-0.338757
Petallength	0.592412	-0.264379	1.000000	0.909156
Petalwidth	0.549628	-0.338757	0.909156	1.000000

Activity 12

1. Import pyspark library, set up SparkContext and SQLContext.

2. Read the CSV data into a Spark object

3. Fill in the missing values in the Sepallength column with the column's mean.

3.1. First, calculate the mean of the Sepallength column.

3.2. Now, impute the missing values in the Sepallength column with the column's mean.

4. Compute the correlation matrix for the dataset. Make sure to import the required modules, as shown here:

```
In [ ]: from pyspark.mllib.stat import Statistics
        import pandas as p
```

5. Now, fill the missing values in the DataFrame before computing the correlation:

6. Next, remove the String columns ('Species') from the PySpark DataFrame.

7. Compute the correlation matrix in Spark

8. Convert the correlation matrix into a pandas DataFrame and display it.

9. Plot the variable pairs showing strong positive correlation (more than 0.7) and fit a linear line on them.

9.1. First, load the data from the Spark DataFrame into a pandas DataFrame.

9.2. Load the required modules and plotting data. Use Seaborn sns.lmplot() method (importing matplotlib will be also required).