05_MissingData

March 29, 2020

0.1 Load data

[1]:
```
from pyspark.sql import SparkSession
```

[2]:
```
spark = SparkSession.builder.appName("MissingData").getOrCreate()
```

[3]:
```
path = "Python-and-Spark-for-Big-Data-master/Spark_DataFrames/ContainsNull.csv"
```

[4]:
```
df = spark.read.csv(path, inferSchema=True, header=True)
```

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

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John| null|
|emp2| null| null|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

0.2 Dop na

[6]:
```
df.na.drop().show()       #drop all rows, which have null
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp4|Cindy|456.0|
+----+-----+-----+
```

[7]:
```
# This is similar to set how="any" (by default)
df.na.drop(how="any").show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
```

```
+----+-----+-----+
|emp4|Cindy|456.0|
+----+-----+-----+
```

[8]:
```
# rows with >= 2 non-null values are keep. The 2nd row has only one non-null␣
 ↪value -> drop
df.na.drop(thresh=2).show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John| null|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

[9]:
```
# only drop rows if all values are null
df.na.drop(how="all").show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John| null|
|emp2| null| null|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

[10]:
```
# combine with subset
df.na.drop(subset=["Sales"]).show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

0.3 Filling missing values

Spark is smart enough to fill values based on data type of each columns

[11]:
```
df.printSchema()
```

```
root
 |-- Id: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sales: double (nullable = true)
```

[12]:
```
# if we pass in a string argument
df.na.fill("My string").show()
```

```
+----+---------+-----+
|  Id|     Name|Sales|
+----+---------+-----+
|emp1|     John| null|
|emp2|My string| null|
|emp3|My string|345.0|
|emp4|    Cindy|456.0|
+----+---------+-----+
```

[13]:
```
# if we pass in a number
df.na.fill(999.).show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John|999.0|
|emp2| null|999.0|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

[14]:
```
# it is a good practice to specify the subset, so that everyone reads the code later can understand your intention
df.na.fill("No name", subset=["Name"]).show()
```

```
+----+-------+-----+
|  Id|   Name|Sales|
+----+-------+-----+
|emp1|   John| null|
|emp2|No name| null|
|emp3|No name|345.0|
|emp4|  Cindy|456.0|
+----+-------+-----+
```

0.4 Using mean to fill na

[15]:
```
from pyspark.sql.functions import mean
```

[20]:
```
# because we use spark function, we need to use select() method
sales_mean = df.select(mean(df["Sales"])).collect()
```

[22]:
```
sales_mean
```

[22]:
```
[Row(avg(Sales)=400.5)]
```

[24]:
```
mean_val = sales_mean[0][0]
mean_val
```

[24]:
```
400.5
```

[25]:
```
df.na.fill(mean_val, subset=["Sales"]).show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John|400.5|
|emp2| null|400.5|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

[26]:
```
# all in one line
df.na.fill(df.select(mean(df["Sales"])).collect()[0][0]), subset=["Sales"]).
 ↪show()
```

```
          File "<ipython-input-26-11783628562d>", line 2
        df.na.fill(df.select(mean(df["Sales"])).collect()[0][0]),␣
↪subset=["Sales"]).show()
                                                                 ␣
↪ ^
    SyntaxError: invalid syntax
```

[ ]:
```

```