3_Missing_Data

February 22, 2022

1 Missing Data

Often data sources are incomplete, which means you will have missing data.

you have 3 basic options for filling in missing data (you will personally have to make the decision for what is the right approach:

- Just keep the missing data points.
- Drop them missing data points (including the entire row).
- Fill them in with some other value.

Let's cover examples of each of these methods!

1.1 Keeping the missing data

A few machine learning algorithms can easily deal with missing data, let's see what it looks like:

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

spark = SparkSession.builder.appName("missingdata").getOrCreate()
```

```
22/02/22 10:50:03 WARN Utils: Your hostname, ThinkCentre resolves to a loopback
address: 127.0.1.1; using 10.180.5.223 instead (on interface eno1)
22/02/22 10:50:03 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
22/02/22 10:50:04 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
22/02/22 10:50:05 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
22/02/22 10:50:05 WARN Utils: Service 'SparkUI' could not bind on port 4041.
Attempting port 4042.
```

[2]:
```python
df = spark.read.csv("ContainsNull.csv", header=True, inferSchema=True)
```

[3]:
```python
df.show()
```

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

[4]:
```
df.describe().show() # For statistics, check why mean and stdev are null
```

```
+-------+----+-----+-----------------+
|summary|  Id| Name|            Sales|
+-------+----+-----+-----------------+
|  count|   4|    2|                2|
|   mean|null| null|            400.5|
| stddev|null| null|78.48885271170677|
|    min|emp1|Cindy|            345.0|
|    max|emp4| John|            456.0|
+-------+----+-----+-----------------+
```

1.2 Drop the missing data

You can use the .na functions for missing data. The drop command has the following parameters:

```
df.na.drop(how='any', thresh=None, subset=None)

* param how: 'any' or 'all'.

    If 'any', drop a row if it contains any nulls.
    If 'all', drop a row only if all its values are null.

* param thresh: int, default None

    If specified, drop rows that have less than `thresh` non-null values.
    This overwrites the `how` parameter.

* param subset:
    optional list of column names to consider.
```

[5]:
```
# Drop any row that contains missing data
# Or, df.na.drop() you drop the rows containing any null or NaN values.
# Only emp4 has complete data, others have null values
df.na.drop().show()
```

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

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

[6]:

```
# Has to have at least 2 NON-null values, then only drop
df.na.drop(thresh=2).show()
```

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

[7]:

```
# Drop only rows in wich Sales has null or NA
df.na.drop(subset=["Sales"]).show()
```

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

[8]:

```
# Drop rows, if any column has NA or null
df.na.drop(how='any').show()
```

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

[9]:

```
# drop a row only if all its values are null.
df.na.drop(how='all').show()
# df.na.drop
```

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

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

1.3 Fill the missing values

- We can also fill the missing values with new values.
- If you have multiple nulls across multiple data types, **Spark is actually smart enough to match up the data types.**

For example:

[10]:
```python
# If you are filling with a string then it fills
# only the missing values which are strings

df.na.fill('NEW VALUE').show() # Fill with some value
```

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

[11]:
```python
# If you are filling with a numeric value then it fills
# only the missing values which are numeric

df.na.fill(0).show() # Fill with zeros
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John|  0.0|
|emp2| null|  0.0|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

Usually you should specify what columns you want to fill with the subset parameter

[12]:
```python
df.na.fill('No Name',subset=['Name']).show() # Fill only a specific column
```

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

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

[13]:
```python
# List of columns
df.na.fill(0,subset=["Sales"]).show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John|  0.0|
|emp2| null|  0.0|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

[14]:
```python
# Single Column
df.na.fill(0,subset="Sales").show()
```

```
+----+-----+-----+
|  Id| Name|Sales|
+----+-----+-----+
|emp1| John|  0.0|
|emp2| null|  0.0|
|emp3| null|345.0|
|emp4|Cindy|456.0|
+----+-----+-----+
```

[15]:
```python
# Fill multiple columns with specified values
df.na.fill({'Sales':0,'Name':'No Name'}).show()
```

```
+----+-------+-----+
|  Id|   Name|Sales|
+----+-------+-----+
|emp1|   John|  0.0|
|emp2|No Name|  0.0|
|emp3|No Name|345.0|
|emp4|  Cindy|456.0|
+----+-------+-----+
```

A very common practice is to fill values with the mean value for the column, for example:

[16]:
```
from pyspark.sql.functions import mean
```

[17]:
```
mean_val = df.select(mean(df['Sales'])).collect()
```

[18]:
```
mean_val
```

[18]:
```
[Row(avg(Sales)=400.5)]
```

[19]:
```
# Weird nested formatting of Row object!
mean_val[0][0]
#mean_val
```

[19]:
```
400.5
```

[20]:
```
mean_sales = mean_val[0][0]
```

[22]:
```
df.na.fill(mean_val[0][0],["Sales"]).show()
```

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