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!

Keeping the missing data

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

In [1]:

```
from pyspark.sql import SparkSession
# May take a little while on a local computer
spark = SparkSession.builder.appName("missingdata").getOrCreate()
```

In [2]:

```
df = spark.read.csv("ContainsNull.csv",header=True,inferSchema=True)
```

In [3]:

```
df.show()
```

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

Notice how the data remains as a null.

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.
```

In [6]:

```
# Drop any row that contains missing data
df.na.drop().show()
```

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

In [8]:

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

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

In [9]:

```
df.na.drop(subset=["Sales"]).show()
```

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

In [10]:

```
df.na.drop(how='any').show()
```

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

In [11]:

```
df.na.drop(how='all').show()
```

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

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:

In [15]:

```
df.na.fill('NEW VALUE').show()
```

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

In [16]:

```
df.na.fill(0).show()
```

```
+----+-----+-----+
|  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

In [17]:

```
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|
+----+-------+-----+
```

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

In [23]:

```
from pyspark.sql.functions import mean
mean_val = df.select(mean(df['Sales'])).collect()

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

Out[23]:

```
400.5
```

In [24]:

```
mean_sales = mean_val[0][0]
```

In [26]:

```
df.na.fill(mean_sales,["Sales"]).show()
```

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

In [28]:

```
# One (very ugly) one-liner
df.na.fill(df.select(mean(df['Sales'])).collect()[0][0],['Sales']).show()
```

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

That is all we need to know for now!