In [3]:
```
val df = spark.read
    .format("csv")
    .option("inferSchema", "true")
    .option("header", "true")
    .load("../../../src/201508_station_data.csv")

df.show(5)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|
+----------+--------------------+---------+-----------+---------+--------+------------+
|         2|San Jose Diridon ...|37.329732|-121.901782|       27|San Jose|    8/6/2013|
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013|
|         4|Santa Clara at Al...|37.333988|-121.894902|       11|San Jose|    8/6/2013|
|         5|    Adobe on Almaden|37.331415|  -121.8932|       19|San Jose|    8/5/2013|
|         6|    San Pedro Square|37.336721|-121.894074|       15|San Jose|    8/7/2013|
+----------+--------------------+---------+-----------+---------+--------+------------+
only showing top 5 rows
```

Out[3]:
```
df: org.apache.spark.sql.DataFrame = [station_id: int, name: string ... 5 more fields]
```

In [4]:
```
df.printSchema()
```

```
root
 |-- station_id: integer (nullable = true)
 |-- name: string (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- dockcount: integer (nullable = true)
 |-- landmark: string (nullable = true)
 |-- installation: string (nullable = true)
```

## Converting to Spark Types

In [5]:
```
df.select(lit(1), lit("one"), lit(1.0)).show(2)
```

```
+---+---+---+
|  1|one|1.0|
+---+---+---+
|  1|one|1.0|
|  1|one|1.0|
+---+---+---+
only showing top 2 rows
```

In [6]:
```
df.select(lit(1), lit("one"), lit(1.0)).printSchema()
```

```
root
 |-- 1: integer (nullable = false)
 |-- one: string (nullable = false)
 |-- 1.0: double (nullable = false)
```

In [7]:
```
df.withColumn("one", lit(1)).show(2)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+---+
|station_id|                name|      lat|       long|dockcount|landmark|installation|one|
+----------+--------------------+---------+-----------+---------+--------+------------+---+
|         2|San Jose Diridon ...|37.329732|-121.901782|       27|San Jose|    8/6/2013|  1|
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013|  1|
+----------+--------------------+---------+-----------+---------+--------+------------+---+
only showing top 2 rows
```

## Working with Booleans

In [11]:
```
df.createOrReplaceTempView("df")
spark.sql("""
SELECT * FROM df
WHERE dockcount = 15
""").show(4)

df.where(df("dockcount") === 15).selectExpr("*").show(4)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|
+----------+--------------------+---------+-----------+---------+--------+------------+
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013|
|         6|    San Pedro Square|37.336721|-121.894074|       15|San Jose|    8/7/2013|
|         7|Paseo de San Antonio|37.333798|-121.886943|       15|San Jose|    8/7/2013|
|         8| San Salvador at 1st|37.330165|-121.885831|       15|San Jose|    8/5/2013|
+----------+--------------------+---------+-----------+---------+--------+------------+
only showing top 4 rows
```

In [8]:
```
spark.sql("""
SELECT * FROM df
WHERE dockcount <> 27
""").show(4)

df.where(col("dockcount") =!= 27).selectExpr("*").show(4)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|
+----------+--------------------+---------+-----------+---------+--------+------------+
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013|
|         4|Santa Clara at Al...|37.333988|-121.894902|       11|San Jose|    8/6/2013|
|         5|    Adobe on Almaden|37.331415|  -121.8932|       19|San Jose|    8/5/2013|
|         6|    San Pedro Square|37.336721|-121.894074|       15|San Jose|    8/7/2013|
+----------+--------------------+---------+-----------+---------+--------+------------+
only showing top 4 rows
```

In [9]:
```
spark.sql("""
SELECT * FROM df
WHERE landmark <> 'San Jose'
""").show(4)

df.where("landmark <> 'San Jose'").selectExpr("*").show(4)
```

```
+----------+--------------------+---------+-----------+---------+------------+------------+
|station_id|                name|      lat|       long|dockcount|    landmark|installation|
+----------+--------------------+---------+-----------+---------+------------+------------+
|        21|   Franklin at Maple|37.481758|-122.226904|       15|Redwood City|   8/12/2013|
|        22|Redwood City Calt...|37.486078|-122.232089|       25|Redwood City|   8/15/2013|
|        23|San Mateo County ...|37.487616|-122.229951|       15|Redwood City|   8/15/2013|
|        24|Redwood City Publ...|37.484219|-122.227424|       15|Redwood City|   8/12/2013|
+----------+--------------------+---------+-----------+---------+------------+------------+
only showing top 4 rows
```

In [20]:
```
spark.sql("""
SELECT * FROM df
WHERE landmark = 'San Francisco'
""").show(4)

df.where("landmark = 'San Francisco'").selectExpr("*").show(4)
```

```
+----------+--------------------+---------+-----------+---------+-------------+------------+
|station_id|                name|      lat|       long|dockcount|     landmark|installation|
+----------+--------------------+---------+-----------+---------+-------------+------------+
|        41|     Clay at Battery|37.795001| -122.39997|       15|San Francisco|   8/19/2013|
|        42|    Davis at Jackson| 37.79728|-122.398436|       15|San Francisco|   8/19/2013|
|        45|Commercial at Mon...|37.794231|-122.402923|       15|San Francisco|   8/19/2013|
|        46|Washington at Kea...|37.795425|-122.404767|       15|San Francisco|   8/19/2013|
+----------+--------------------+---------+-----------+---------+-------------+------------+
only showing top 4 rows
```

In [12]:
```
val stationFilter = col("station_id") > 6
val landmarkFilter = col("landmark").contains("San")

df.where(stationFilter.or(landmarkFilter)).show(15)
```

```
+----------+--------------------+---------+-----------+---------+------------+------------+
|station_id|                name|      lat|       long|dockcount|    landmark|installation|
+----------+--------------------+---------+-----------+---------+------------+------------+
|         2|San Jose Diridon ...|37.329732|-121.901782|       27|    San Jose|    8/6/2013|
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|    San Jose|    8/5/2013|
|         4|Santa Clara at Al...|37.333988|-121.894902|       11|    San Jose|    8/6/2013|
|         5|    Adobe on Almaden|37.331415|  -121.8932|       19|    San Jose|    8/5/2013|
|         6|    San Pedro Square|37.336721|-121.894074|       15|    San Jose|    8/7/2013|
|         7|Paseo de San Antonio|37.333798|-121.886943|       15|    San Jose|    8/7/2013|
|         8| San Salvador at 1st|37.330165|-121.885831|       15|    San Jose|    8/5/2013|
|         9|           Japantown|37.348742|-121.894715|       15|    San Jose|    8/5/2013|
|        10|  San Jose City Hall|37.337391|-121.886995|       15|    San Jose|    8/6/2013|
|        11|         MLK Library|37.335885| -121.88566|       19|    San Jose|    8/6/2013|
|        12|SJSU 4th at San C...|37.332808|-121.883891|       19|    San Jose|    8/7/2013|
|        13|       St James Park|37.339301|-121.889937|       15|    San Jose|    8/6/2013|
|        14|Arena Green / SAP...|37.332692|-121.900084|       19|    San Jose|    8/5/2013|
|        16|SJSU - San Salvad...|37.333955|-121.877349|       15|    San Jose|    8/7/2013|
|        21|   Franklin at Maple|37.481758|-122.226904|       15|Redwood City|   8/12/2013|
+----------+--------------------+---------+-----------+---------+------------+------------+
only showing top 15 rows
```

Out[12]:
```
stationFilter: org.apache.spark.sql.Column = (station_id > 6)
landmarkFilter: org.apache.spark.sql.Column = contains(landmark, San)
```

In [14]:
```
df.where(stationFilter).where(landmarkFilter).show(10)

// same thing with
df.where(stationFilter.and(landmarkFilter)).show(10)
```

```
+----------+--------------------+---------+-----------+---------+-------------+------------+
|station_id|                name|      lat|       long|dockcount|     landmark|installation|
+----------+--------------------+---------+-----------+---------+-------------+------------+
|         7|Paseo de San Antonio|37.333798|-121.886943|       15|     San Jose|    8/7/2013|
|         8| San Salvador at 1st|37.330165|-121.885831|       15|     San Jose|    8/5/2013|
|         9|           Japantown|37.348742|-121.894715|       15|     San Jose|    8/5/2013|
|        10|  San Jose City Hall|37.337391|-121.886995|       15|     San Jose|    8/6/2013|
|        11|         MLK Library|37.335885| -121.88566|       19|     San Jose|    8/6/2013|
|        12|SJSU 4th at San C...|37.332808|-121.883891|       19|     San Jose|    8/7/2013|
|        13|       St James Park|37.339301|-121.889937|       15|     San Jose|    8/6/2013|
|        14|Arena Green / SAP...|37.332692|-121.900084|       19|     San Jose|    8/5/2013|
|        16|SJSU - San Salvad...|37.333955|-121.877349|       15|     San Jose|    8/7/2013|
|        41|     Clay at Battery|37.795001| -122.39997|       15|San Francisco|   8/19/2013|
+----------+--------------------+---------+-----------+---------+-------------+------------+
only showing top 10 rows
```

## Working with Numbers

In [21]:
```
val test = (pow(col("station_id"), 3) + 3) * col("dockcount")
df.select(expr("*"), test.alias("test")).show(4)

df.selectExpr("*", "(POWER(station_id, 3) + 3) * dockcount as test").show(4)

spark.sql("""
SELECT *, ((POWER(station_id, 3) + 3) * dockcount) AS test
FROM df """).show(4)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|  test|
+----------+--------------------+---------+-----------+---------+--------+------------+------+
|         2|San Jose Diridon ...|37.329732|-121.901782|       27|San Jose|    8/6/2013| 297.0|
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013| 450.0|
|         4|Santa Clara at Al...|37.333988|-121.894902|       11|San Jose|    8/6/2013| 737.0|
|         5|    Adobe on Almaden|37.331415|  -121.8932|       19|San Jose|    8/5/2013|2432.0|
+----------+--------------------+---------+-----------+---------+--------+------------+------+
only showing top 4 rows
```

In [13]:
```
df.withColumn("manyDock", not(df("dockcount").leq(25)))
    .filter("manyDock")
    .select("station_id", "name", "dockcount").show(3)

df.withColumn("notManyDock", expr("NOT dockcount >= 12"))
    .filter("notManyDock")
    .select("station_id", "name", "dockcount").show(3)
```

```
+----------+--------------------+---------+
|station_id|                name|dockcount|
+----------+--------------------+---------+
|         2|San Jose Diridon ...|       27|
|        61|     2nd at Townsend|       27|
|        67|      Market at 10th|       27|
+----------+--------------------+---------+
only showing top 3 rows

+----------+--------------------+---------+
|station_id|                name|dockcount|
+----------+--------------------+---------+
|         4|Santa Clara at Al...|       11|
|        32|Castro Street and...|       11|
|        35|University and Em...|       11|
+----------+--------------------+---------+
only showing top 3 rows
```

In [16]:
```
spark.sql("""
SELECT round(lat) AS lat_rounded
FROM df """).show(3)

df.select(round(col("lat"), 2).as("lat_rounded")).show(3)
```

```
+-----------+
|lat_rounded|
+-----------+
|       37.0|
|       37.0|
|       37.0|
+-----------+
only showing top 3 rows
```

In [14]:
```
df.withColumn("brounded_lat", bround(df("lat"))).show(3)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+------------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|brounded_lat|
+----------+--------------------+---------+-----------+---------+--------+------------+------------+
|         2|San Jose Diridon ...|37.329732|-121.901782|       27|San Jose|    8/6/2013|        37.0|
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013|        37.0|
|         4|Santa Clara at Al...|37.333988|-121.894902|       11|San Jose|    8/6/2013|        37.0|
+----------+--------------------+---------+-----------+---------+--------+------------+------------+
only showing top 3 rows
```

In [17]:
```
df.withColumn("rounded_lat", round(col("lat"), 2)).show(3)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+-----------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|rounded_lat|
+----------+--------------------+---------+-----------+---------+--------+------------+-----------+
|         2|San Jose Diridon ...|37.329732|-121.901782|       27|San Jose|    8/6/2013|      37.33|
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013|      37.33|
|         4|Santa Clara at Al...|37.333988|-121.894902|       11|San Jose|    8/6/2013|      37.33|
+----------+--------------------+---------+-----------+---------+--------+------------+-----------+
only showing top 3 rows
```

#### General stats

In [34]:
```
df.select("station_id", "name", "lat", "dockcount").describe().show()
```

```
+-------+------------------+--------------------+-------------------+-----------------+
|summary|        station_id|                name|                lat|        dockcount|
+-------+------------------+--------------------+-------------------+-----------------+
|  count|                70|                  70|                 70|               70|
|   mean|              43.0|                null|  37.59024338428572|17.65714285714286|
| stddev|24.166091947189145|                null|0.20347253639672416|4.010441857493954|
|    min|                 2|       2nd at Folsom|          37.329732|               11|
|    max|                84|Yerba Buena Cente...|           37.80477|               27|
+-------+------------------+--------------------+-------------------+-----------------+
```

**Correlation**

In [37]:
```
df.stat.corr("station_id", "dockcount")
```

Out[37]:
```
res33: Double = 0.24015841145323474
```

In [38]:
```
df.select(corr("station_id", "dockcount")).show()

spark.sql("SELECT corr(station_id, dockcount) FROM df").show()
```

```
+---------------------------+
|corr(station_id, dockcount)|
+---------------------------+
|        0.24015841145323474|
+---------------------------+
```

**Crosstab**

According to [the official documentation (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=crosstab#pyspark.sql.DataFrame.crosstab)](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=crosstab#pyspark.sql.DataFrame.crosstab) the crosstab method computes a pair-wise frequency table of the given columns. Also known as a contingency table. The number of distinct values for each column should be less than 1e4. At most 1e6 non-zero pair frequencies will be returned. The first column of each row will be the distinct values of col1 and the column names will be the distinct values of col2.

In [15]:
```
//df.stat.crosstab("lat", "long").show()
```

Whereas [freqItems (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=freqitems#pyspark.sql.DataFrame.freqItems)](http://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=freqitems#pyspark.sql.DataFrame.freqItems) let's you find frequent items for columns, possibly with false positives.

In [41]:
```
df.stat.freqItems(Seq("lat", "long")).show()
```

```
+--------------------+--------------------+
|       lat_freqItems|      long_freqItems|
+--------------------+--------------------+
|[37.80477, 37.798...|[-122.229951, -12...|
+--------------------+--------------------+
```

Adding a unique ID to each row (note that the number doesn't necesseraly follow each other)

In [45]:
```
df.select(monotonically_increasing_id(), col("station_id")).show(5)
```

```
+-----------------------------+----------+
|monotonically_increasing_id()|station_id|
+-----------------------------+----------+
|                            0|         2|
|                            1|         3|
|                            2|         4|
|                            3|         5|
|                            4|         6|
+-----------------------------+----------+
only showing top 5 rows
```

## Working with Strings

In [46]:
```
df.show(2)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|
+----------+--------------------+---------+-----------+---------+--------+------------+
|         2|San Jose Diridon ...|37.329732|-121.901782|       27|San Jose|    8/6/2013|
|         3|San Jose Civic Ce...|37.330698|-121.888979|       15|San Jose|    8/5/2013|
+----------+--------------------+---------+-----------+---------+--------+------------+
only showing top 2 rows
```

In [51]:
```
df.select(lower(col("Name")), upper(col("Name")), initcap(lower(col("Name")))).show(2, false)

spark.sql("SELECT lower(Name), upper(Name), initcap(lower(Name)) FROM df").show(2)
```

```
+---------------------------------+---------------------------------+---------------------------------+
|lower(Name)                      |upper(Name)                      |initcap(lower(Name))             |
+---------------------------------+---------------------------------+---------------------------------+
|san jose diridon caltrain station|SAN JOSE DIRIDON CALTRAIN STATION|San Jose Diridon Caltrain Station|
|san jose civic center            |SAN JOSE CIVIC CENTER            |San Jose Civic Center            |
+---------------------------------+---------------------------------+---------------------------------+
only showing top 2 rows
```

Note that if lpad or rpad takes a number less than the length of the string, it will always remove values from the right side of the string.

In [55]:
```
df.select(
    ltrim(lit("     HELLO     ")).as("ltrim"),
    rtrim(lit("     HELLO     ")).as("rtrim"),
    trim(lit("     HELLO     ")).as("trim"),
    lpad(lit("HELLO"), 3, " ").as("lp"),
    rpad(lit("HELLO"), 10, " ").as("rp")).show(2)

spark.sql("""SELECT
    ltrim('   HELLLO   '),
    rtrim('   HELLLO   '),
    trim('   HELLLO   '),
    lpad('HELLO', 3, ' '),
    rpad('HELLO', 10, ' ')
FROM df""").show(2)
```

```
+-------------------+-------------------+-------------------+-----------------+------------------+
|ltrim(   HELLLO   )|rtrim(   HELLLO   )|trim(   HELLLO   )|lpad(HELLO, 3,  )|rpad(HELLO, 10,  )|
+-------------------+-------------------+-------------------+-----------------+------------------+
|          HELLLO   |             HELLLO|             HELLLO|              HEL|        HELLO     |
|          HELLLO   |             HELLLO|             HELLLO|              HEL|        HELLO     |
+-------------------+-------------------+-------------------+-----------------+------------------+
only showing top 2 rows
```

**Regular Expressions**

There are two key functions in Spark that you'll need in order to perform regular expression tasks: regexp_extract and regexp_replace. These functions extract values and replace values, respectively.

Replace substitute specific names in our name column with TEST:

In [36]:
```
val dfBis = df.withColumn("name", upper(col("name")))

import org.apache.spark.sql.functions.regexp_replace

val simpleWords = Seq("san", "santa", "adobe")
val regexString = simpleWords.map(_.toUpperCase).mkString("|")
// the | signifies `OR` in regular expression syntax

dfBis.select(
    regexp_replace(col("name"), regexString, "TEST").alias("name_clean"),
    col("name")).show(5)
```

```
+--------------------+--------------------+
|          name_clean|                name|
+--------------------+--------------------+
|TEST JOSE DIRIDON...|SAN JOSE DIRIDON ...|
|TEST JOSE CIVIC C...|SAN JOSE CIVIC CE...|
|TESTTA CLARA AT A...|SANTA CLARA AT AL...|
|     TEST ON ALMADEN|    ADOBE ON ALMADEN|
|   TEST PEDRO SQUARE|    SAN PEDRO SQUARE|
+--------------------+--------------------+
only showing top 5 rows
```

Out[36]:
```
dfBis: org.apache.spark.sql.DataFrame = [station_id: int, name: string ... 5 more fields]
import org.apache.spark.sql.functions.regexp_replace
simpleWords: Seq[String] = List(san, santa, adobe)
regexString: String = SAN|SANTA|ADOBE
```

Replace given characters with other characters.

In [67]:
```
df.select(translate(col("landmark"), "San", "S4N"), col("landmark")).show(2)
```

```
+----------------------------+--------+
|translate(landmark, San, S4N)|landmark|
+----------------------------+--------+
|                    S4N Jose|San Jose|
|                    S4N Jose|San Jose|
+----------------------------+--------+
only showing top 2 rows
```

Pulling out the first mentioned word:

In [39]:
```
import org.apache.spark.sql.functions.regexp_extract
val regexString = simpleWords.map(_.toUpperCase).mkString("(", "|", ")")
// the | signifies OR in regular expression syntax
dfBis.select(
    regexp_extract(col("name"), regexString, 1).alias("name_clean"),
    col("name")).show(2)
```

```
+----------+--------------------+
|name_clean|                name|
+----------+--------------------+
|       SAN|SAN JOSE DIRIDON ...|
|       SAN|SAN JOSE CIVIC CE...|
+----------+--------------------+
only showing top 2 rows
```

Out[39]:
```
import org.apache.spark.sql.functions.regexp_extract
regexString: String = (SAN|SANTA|ADOBE)
```

Check for their existence

In [40]:
```
dfBis.show(2)
```

```
+----------+--------------------+---------+-----------+---------+--------+------------+
|station_id|                name|      lat|       long|dockcount|landmark|installation|
+----------+--------------------+---------+-----------+---------+--------+------------+
|         2|SAN JOSE DIRIDON ...|37.329732|-121.901782|       27|San Jose|    8/6/2013|
|         3|SAN JOSE CIVIC CE...|37.330698|-121.888979|       15|San Jose|    8/5/2013|
+----------+--------------------+---------+-----------+---------+--------+------------+
only showing top 2 rows
```

In [43]:
```
val containsSan = col("name").contains("SAN")
val containsJose = col("landmark").contains("Jose")

dfBis.withColumn("test", containsSan.or(containsJose))
    .where("test")
    .select("name", "landmark").show(3, false)
```

```
+--------------------------------+--------+
|name                            |landmark|
+--------------------------------+--------+
|SAN JOSE DIRIDON CALTRAIN STATION|San Jose|
|SAN JOSE CIVIC CENTER           |San Jose|
|SANTA CLARA AT ALMADEN          |San Jose|
+--------------------------------+--------+
only showing top 3 rows
```

Out[43]:
```
containsSan: org.apache.spark.sql.Column = contains(name, SAN)
containsJose: org.apache.spark.sql.Column = contains(landmark, Jose)
```

## Working with Dates and Timestamps

In [23]:
```
val dateDF = spark.range(3)
    .withColumn("current_date", current_date())
    .withColumn("current_timestamp", current_timestamp())

dateDF.createOrReplaceTempView("dateTable")
dateDF.show()
dateDF.printSchema()
```

```
+---+------------+--------------------+
| id|current_date|   current_timestamp|
+---+------------+--------------------+
|  0|  2020-08-29|2020-08-29 15:58:...|
|  1|  2020-08-29|2020-08-29 15:58:...|
|  2|  2020-08-29|2020-08-29 15:58:...|
+---+------------+--------------------+

root
 |-- id: long (nullable = false)
 |-- current_date: date (nullable = false)
 |-- current_timestamp: timestamp (nullable = false)
```

Out[23]:
```
dateDF: org.apache.spark.sql.DataFrame = [id: bigint, current_date: date ... 1 more field]
```

In [25]:
```
dateDF.select(date_sub(col("current_date"), 5), date_add(col("current_date"), 5)).show(1)
spark.sql("SELECT date_sub(current_date, 5), date_add(current_date, 5) FROM dateTable").show(1)
```

```
+------------------------+------------------------+
|date_sub(current_date, 5)|date_add(current_date, 5)|
+------------------------+------------------------+
|               2020-08-24|               2020-09-03|
+------------------------+------------------------+
only showing top 1 row
```

In [29]:
```
dateDF.withColumn("week_ago", date_sub(col("current_date"), 7))
    .select(datediff(col("week_ago"), col("current_date"))).show(1)

dateDF.select(
    to_date(lit("2016-01-01")).alias("start"),
    to_date(lit("2017-05-22")).alias("end"))
    .select(months_between(col("start"), col("end"))).show(1)
```

```
+-------------------------------+
|datediff(week_ago, current_date)|
+-------------------------------+
|                              -7|
+-------------------------------+
only showing top 1 row

+------------------------------+
|months_between(start, end, true)|
+------------------------------+
|                    -16.67741935|
+------------------------------+
only showing top 1 row
```

Date doesn't require a format

In [30]:
```
spark.range(5).withColumn("date", lit("2017-01-01"))
    .select(to_date(col("date"))).show(1)
```

```
+---------------+
|to_date(`date`)|
+---------------+
|     2017-01-01|
+---------------+
only showing top 1 row
```

The to_date function allows you to convert a string to a date, optionally with a specified format. Spark will not throw an error if it cannot parse the date; rather, it will just return null.

In [31]:
```
dateDF.select(to_date(lit("2016-20-12")),to_date(lit("2017-12-11"))).show(1)
```

```
+---------------------+---------------------+
|to_date('2016-20-12')|to_date('2017-12-11')|
+---------------------+---------------------+
|                 null|           2017-12-11|
+---------------------+---------------------+
only showing top 1 row
```

In [110]:
```
val dateFormat = "yyyy-dd-MM"
val cleanDateDF = spark.range(1).select(
    to_date(lit("2017-12-11"), dateFormat).alias("date"),
    to_date(lit("2017-20-12"), dateFormat).alias("date2"))

cleanDateDF.createOrReplaceTempView("dateTable2")
cleanDateDF.show()
```

```
+----------+----------+
|      date|     date2|
+----------+----------+
|2017-11-12|2017-12-20|
+----------+----------+
```

Out[110]:
```
dateFormat: String = yyyy-dd-MM
cleanDateDF: org.apache.spark.sql.DataFrame = [date: date, date2: date]
```

In [47]:
```
val cleanDF = cleanDateDF
    .withColumn("date", to_timestamp(col("date"), dateFormat))
    .withColumn("date2", to_timestamp(col("date2"), dateFormat))

cleanDF.show()
cleanDF.select(to_timestamp(col("date"), dateFormat)).printSchema()

spark.sql("""SELECT to_timestamp(date, 'yyyy-dd-MM'), to_timestamp(date2, 'yyyy-dd-MM')
FROM dateTable2""").show()
```

```
+-------------------+-------------------+
|               date|              date2|
+-------------------+-------------------+
|2017-11-12 00:00:00|2017-12-20 00:00:00|
+-------------------+-------------------+

root
 |-- to_timestamp(`date`, 'yyyy-dd-MM'): timestamp (nullable = true)

+----------------------------------------------+-----------------------------------------------+
|to_timestamp(datetable2.`date`, 'yyyy-dd-MM')|to_timestamp(datetable2.`date2`, 'yyyy-dd-MM')|
+----------------------------------------------+-----------------------------------------------+
|                           2017-11-12 00:00:00|                            2017-12-20 00:00:00|
+----------------------------------------------+-----------------------------------------------+
```

Out[47]:
```
cleanDF: org.apache.spark.sql.DataFrame = [date: timestamp, date2: timestamp]
```

In [70]:
```
val dateDF = spark.range(1).withColumn("current_date", current_date())
dateDF.show()

dateDF.filter(col("current_date") <= lit("2017-12-12")).show()
dateDF.filter(col("current_date") >= lit("2017-12-12")).show()
```

```
+---+------------+
| id|current_date|
+---+------------+
|  0|  2020-08-29|
+---+------------+

+---+------------+
| id|current_date|
+---+------------+
+---+------------+

+---+------------+
| id|current_date|
+---+------------+
|  0|  2020-08-29|
+---+------------+
```

Out[70]:
```
dateDF: org.apache.spark.sql.DataFrame = [id: bigint, current_date: date]
```

In [129]:
```
val dateDF = spark.range(1)
    .withColumn("unix_ts", unix_timestamp(current_timestamp, "MM:dd:yyyy hh:mm:ss"))
    .withColumn("time_stamp", col("unix_ts").cast("timestamp")) // or to_timestamp
    .withColumn("date", to_date(col("time_stamp")))

dateDF.show()
```

```
+---+----------+-------------------+----------+
| id|   unix_ts|         time_stamp|      date|
+---+----------+-------------------+----------+
|  0|1598722033|2020-08-29 17:27:13|2020-08-29|
+---+----------+-------------------+----------+
```

Out[129]:
```
dateDF: org.apache.spark.sql.DataFrame = [id: bigint, unix_ts: bigint ... 2 more fields]
```

## Working with Nulls in Data

**Coalesce** The coalesce function allows you to select the first non-null value from a set of columns If there are no null values, so it simply returns the first column.

In [5]:
```
val df = spark.read.format("csv")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("../../../src/2010-12-01.csv")

df.show(3)
```

```
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
|InvoiceNo|StockCode|         Description|Quantity|        InvoiceDate|UnitPrice|CustomerID|       Country|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|2010-12-01 08:26:00|     2.55|   17850.0|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|2010-12-01 08:26:00|     3.39|   17850.0|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|2010-12-01 08:26:00|     2.75|   17850.0|United Kingdom|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
only showing top 3 rows
```

Out[5]:
```
df: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 6 more fields]
```

In [ ]:
```
df("columnName")            // On a specific DataFrame.
col("columnName")           // A generic column no yet associated with a DataFrame.
col("columnName.field")     // Extracting a struct field
col("`a.column.with.dots`") // Escape `.` in column names.
$"columnName"               // Scala short hand for a named column.
expr("a + 1")               // A column that is constructed from a parsed SQL Expression.
lit("abc")                  // A column that produces a literal (constant) value.
```

In [6]:
```
df.select(coalesce(col("StockCode"), df("CustomerID"))).show(3)
```

```
+------------------------------+
|coalesce(StockCode, CustomerID)|
+------------------------------+
|                        85123A|
|                         71053|
|                        84406B|
+------------------------------+
only showing top 3 rows
```

In [31]:
```
//not working examples
//df.select([count(when(isnan(c), c)).alias(c) for c in df.columns]).show()
//df.select(for (c <- df.columns) yield count(when(isnan(c), c)).alias(c)).show()
```

Nb of NaNs

In [7]:
```
println(df.filter(df("Description").isNull).count)
println(df.filter(df("Description") === "").count)
```

```
10
0
```

In [8]:
```
df.filter(df("Description").isNull || df("Description") === "").count
```

Out[8]:
```
res5: Long = 10
```

In [9]:
```
for (c <- df.columns) println(c, df.filter(df("Description").isNull || df("Description") === "").count)
```

```
(InvoiceNo,10)
(StockCode,10)
(Description,10)
(Quantity,10)
(InvoiceDate,10)
(UnitPrice,10)
(CustomerID,10)
(Country,10)
```

How to find duplicated lines

In [14]:
```
df.groupBy("InvoiceNo", "StockCode", "Description", "Quantity", "InvoiceDate", "UnitPrice", "CustomerID", "Co
    .count().show(3)
```

```
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
-----+
|InvoiceNo|StockCode|         Description|Quantity|        InvoiceDate|UnitPrice|CustomerID|       Country|
count|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
-----+
|   536407|    22632|HAND WARMER RED P...|       6|2010-12-01 11:34:00|     1.85|   17850.0|United Kingdom|
1|
|   536408|   84029E|RED WOOLLY HOTTIE...|       4|2010-12-01 11:41:00|     3.75|   14307.0|United Kingdom|
1|
|   536409|    22531|MAGIC DRAWING SLA...|       1|2010-12-01 11:45:00|     0.42|   17908.0|United Kingdom|
1|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+
-----+
only showing top 3 rows
```

In [ ]:
```
import pyspark.sql.functions as f
df.groupBy(df.columns)\
    .count()\
    .where(f.col('count') > 1)\
    .select(f.sum('count'))\
    .show()
```

In [11]:
```
println(df.distinct().count(), df.count())
```

```
(3064,3108)
```

In [13]:
```
var dfB = df.dropDuplicates()
dfB.count()
```

Out[13]:
```
dfB: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [InvoiceNo: string, StockCode: string ... 6 m
ore fields]
res10: Long = 3064
```

Remove rows that contain nulls : the default is to drop any row in which any value is null

In [17]:
```
dfB.na.drop("any").count() // or drop()
```

Out[17]:
```
res14: Long = 1924
```

In [18]:
```
dfB.na.drop("all").count()
```

Out[18]:
```
res15: Long = 3064
```

Fill all null values in columns of type String

In [19]:
```
df.na.fill("All Null values become this string")
```

Out[19]:
```
res16: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 6 more fields]
```

For integers

In [20]:
```
df.na.fill(5, Seq("StockCode", "InvoiceNo"))
```

Out[20]:
```
res17: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 6 more fields]
```

In [21]:
```
val fillColValues = Map("StockCode" -> 5, "Description" -> "No Value")
df.na.fill(fillColValues)
```

Out[21]:
```
fillColValues: scala.collection.immutable.Map[String,Any] = Map(StockCode -> 5, Description -> No Value)
res18: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 6 more fields]
```

A more flexible option that you can use with more than just null values. (only requirement is that this value be the same type as the original value)

In [22]:
```
df.na.replace("Description", Map("" -> "UNKNOWN"))
```

Out[22]:
```
res19: org.apache.spark.sql.DataFrame = [InvoiceNo: string, StockCode: string ... 6 more fields]
```