databricks5.2 Manipulating Data

Manipulating Data

In this notebook, you will be working with the online retail sales data that you worked with in the Module 3 Lab. This time, you will work with the columns of data that contain `NULL` values and a non-standard date format.

Run the following queries to learn about how to work with and manage null values and timestamps in Spark SQL. In this notebook, you will:

- Sample a table
- Access individual values from an array
- Reformat values using a padding function
- Concatenate values to match a standard format
- Access parts of a `DateType` value like the month, day, or year

Getting Started

Run the cell below to set up your classroom environment.

```
%run ../Includes/Classroom-Setup
```

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false
```

```
res2: Boolean = false
```

Create table

Our data is stored as a csv. The optional arguments show the path to the data and store the first row as a header.

```
DROP TABLE IF EXISTS outdoorProductsRaw;
CREATE TABLE outdoorProductsRaw USING csv OPTIONS (
  path "/mnt/training/online_retail/data-001/data.csv",
  header "true"
)
```

OK

Describe

Recall that the `DESCRIBE` command tells us about the schema of the table. Notice that all of our columns are string values.

```
DESCRIBE outdoorProductsRaw
```

	col_name	data_type	comment
1	InvoiceNo	string	null
2	StockCode	string	null
3	Description	string	null
4	Quantity	string	null
5	InvoiceDate	string	null
6	UnitPrice	string	null
7	CustomerID	string	null

Showing all 8 rows.

Sample the table

In the previous reading, you accessed a random sample of rows from a table using the `RAND()` function and `LIMIT` keyword. While this is a common way to retrieve a sample with other SQL dialects, Spark SQL includes a built-in function that you may want to use instead.

The function, `TABLESAMPLE`, allows you to return a number of rows or a certain percentage of the data. In the cell directly below this one, we show that `TABLESAMPLE` can be used to access a specific number of rows. In the following cell, we show that it can be used to access a given percentage of the data. Please note, however, any table display is limited to 1,000 rows. If the percentage of data you request returns more thna 1,000 rows, only the first 1000 will show.


```
SELECT * FROM outdoorProductsRaw TABLESAMPLE (5 ROWS)
```

	InvoiceNo	StockCode	Description	Quanti
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6
2	536365	71053	WHITE METAL LANTERN	6
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6

Showing all 5 rows.

```
SELECT * FROM outdoorProductsRaw TABLESAMPLE (2 PERCENT) ORDER BY InvoiceDate
```

	InvoiceNo	StockCode	Description
1	540566	84459A	PINK METAL CHICKEN HEART
2	540568	21933	PINK VINTAGE PAISLEY PICNIC BAG
3	540639	21880	RED RETROSPOT TAPE
4	540642	16238	PARTY TIME PENCIL ERASERS
5	540644	21715	GIRLS VINTAGE TIN SEASIDE BUCKET
6	540646	90114	SUMMER DAISIES BAG CHARM
7	540646	70007	HI TEC ALPINE HAND WARMER

Truncated results, showing first 1000 rows.

Check for null values

Run this cell to see the number of `NULL` values in the `Description` column of our table.

```
SELECT count(*) FROM outdoorProductsRaw WHERE Description IS NULL;
```

	count(1)
1	166

Showing all 1 rows.

Create a temporary view

The next cell creates the temporary view `outdoorProducts`. By now, you should be familiar with how to create (or replace) a temporary view. There are a few new commands to notice in this particular command.

This is where we will start to work with the problematic date formatting mentioned previously. Did you notice the inconsistency in your displays?

Our dates do not have a standard number of digits for months and years. For example, `12/1/11` has a two-digit month and one-digit day, while `1/10/11` has an one-digit month and two-digit day. It's easy enough to specify a format to convert a string to a date, but the format must be consistent throughout the table. We will begin to attempt a fix for this problem by simply separating all of the components of the date and dropping the time value entirely.

Code breakdown

`COALESCE` - This command is popular among many different SQL dialects. We can use it to replace `NULL` values. For all `NULL` values in the `Description` column, `COALESCE()` will replace the null with a value you include in the function. In this case,

the value is `"Misc"`. For more information about `COALESCE`, check the documentation (https://spark.apache.org/docs/latest/api/sql/index.html#coalesce).

`SPLIT` - This command splits a string value around a specified character and returns an **array**. An array is a list of values that you can access by position. In this case, the forward slash ("/") is the character we use to split the data. The first value in the array is the month. This list is **zero-indexed** for the index of the first position is **0**. Since we want to pull out the first value as the month, we indicate the value like this: `SPLIT(InvoiceDate, "/")[0]` and rename the column **`month`**. The day is the second value and its index is 1.

The third `SPLIT` is different. Remember that our `InvoiceDate` column is a string that includes a date and time. Each part of the date is seperated by a forward slash, but between the date and the time, there is only a space. **`Line 10`** contains a **nested** `SPLIT` function that splits the string on a space delimiter.

`SPLIT(InvoiceDate, " ")[0]` --> Drops the time from the string and leaves the date intact. Then, we split that value on the forward slash delimiter. We access the year at index 2. Learn more about the `SPLIT` function by accessing the documentation (https://spark.apache.org/docs/latest/api/sql/#split).

```
DROP TABLE IF EXISTS outdoorProducts;
CREATE
OR REPLACE TEMPORARY VIEW outdoorProducts AS
SELECT
  InvoiceNo,
  StockCode,
  COALESCE(Description, "Misc") AS Description,
  Quantity,
  SPLIT(InvoiceDate, "/")[0] month,
  SPLIT(InvoiceDate, "/")[1] day,
  SPLIT(SPLIT(InvoiceDate, " ")[0], "/")[2] year,
  UnitPrice,
  Country
FROM
  outdoorProductsRaw
```

OK

```
---MY QUERY
SELECT * FROM outdoorProducts
ORDER BY RAND() LIMIT 5;
```

	InvoiceNo	StockCode	Description	Quantity
1	541104	21677	HEARTS STICKERS	1
2	536639	22632	HAND WARMER RED RETROSPOT	12
3	536530	22411	JUMBO SHOPPER VINTAGE RED PAISLEY	6
4	539595	22653	BUTTON BOX	1
5	539038	84826	ASSTD DESIGN 3D PAPER STICKERS	60

Showing all 5 rows.

Check "Misc" values

We perform a quick sanity check here to demonstrate that all of the `NULL` values in Description have been replaced with the string `"Misc"`.

```
SELECT count(*) FROM outdoorProducts WHERE Description = "Misc"
```

	count(1)
1	166

Showing all 1 rows.

Create a new table

Now, we can write a new table with a consistently formatted date string. Notice that this table creation statement has a CTE inside of it. Recall that the CTE starts with a `WITH` clause.

Notice the `LPAD()` functions on lines 11 and 12. This function (https://spark.apache.org/docs/latest/api/sql/#lpad) inserts characters to the left of a string until the string reachers a certain length. In this example, we use `LPAD` to insert

a zero to the left of any value in the month or day column that **is not** two digits. For values that are two digits, `LPAD` does nothing.

We use the `padStrings` CTE to standardize the length of the individual date components. When we query the CTE, we use the `CONCAT_WS()` function to put the date string back together. This function (https://spark.apache.org/docs/latest/api/sql/#concat_ws) returns a concatenated string with a specified separator. In this example, we concatenate values from the

```
DROP TABLE IF EXISTS standardDate;
CREATE TABLE standardDate

WITH padStrings AS
(
SELECT
  InvoiceNo,
  StockCode,
  Description,
  Quantity,
  LPAD(month, 2, 0) AS month,
  LPAD(day, 2, 0) AS day,
  year,
  UnitPrice,
  Country
FROM outdoorProducts
)
SELECT
 InvoiceNo,
  StockCode,
  Description,
  Quantity,
  concat_ws("/", month, day, year) sDate,
  UnitPrice,
  Country
FROM padStrings;
```

Query returned no results

Table check

When we view our new table, we can see that the date field shows two digits each for the month, day, and year.

```
SELECT * FROM standardDate LIMIT 5;
```

	InvoiceNo	StockCode	Description	Quanti
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6
2	536365	71053	WHITE METAL LANTERN	6
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6

Showing all 5 rows.

Check schema

Oops! All of our values are still strings. The date field would be much more useful as a `DateType`.

```
DESCRIBE standardDate;
```

	col_name	data_type	comment
1	InvoiceNo	string	
2	StockCode	string	
3	Description	string	
4	Quantity	string	
5	sDate	string	
6	UnitPrice	string	
7	Country	string	

Showing all 10 rows.

Change to DateType

In the next cell, we create a new temporary view that converts the value to a date. The optional argument `MM/dd/yy` indicates the meaning of each part of the date. You can find a complete guide to Spark SQL's Datetime Patterns here (https://spark.apache.org/docs/latest/sql-ref-datetime-pattern.html).

Also, we cast the `UnitPrice` as a `DOUBLE` so that we can treat it as a number.

```
CREATE
OR REPLACE TEMPORARY VIEW salesDateFormatted AS
SELECT
  InvoiceNo,
  StockCode,
  to_date(sDate, "MM/dd/yy") date,
  Quantity,
  CAST(UnitPrice AS DOUBLE)
FROM
  standardDate
```

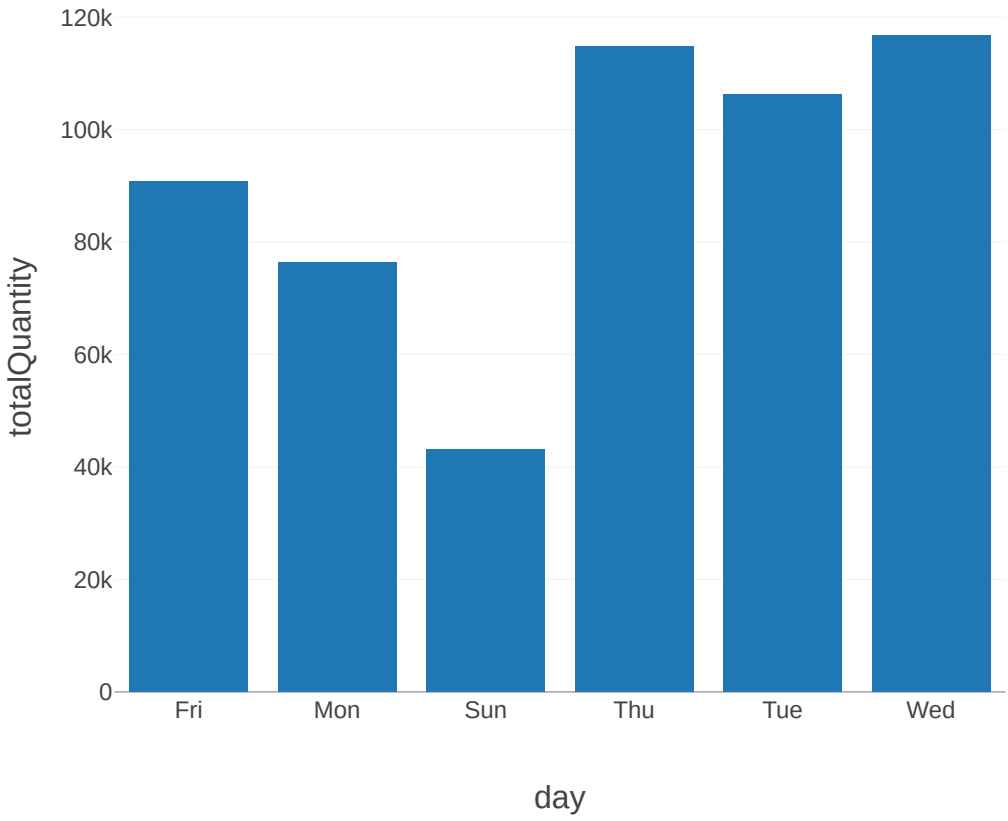
OK

Visualize Data

We can extract the day of the week and figure out the total quantitybar of items sold on each day. You can create a quick visual by clicking on the chart icon and creating a bar chart where the key is the `day` and the values are the `quantity`.

We use the `date_format()` function to map the day to a day of the week. This function (https://spark.apache.org/docs/latest/api/sql/#date_format) converts a `timestamp` to a `string` in the format specified. For this command, the `"E"` specifies that we want the output to be the day of the week.

```
SELECT
  date_format(date, "E") day,
  SUM(quantity) totalQuantity
FROM
  salesDateFormatted
GROUP BY (day)
ORDER BY day
```




```
%run ../Includes/Classroom-Cleanup
```