**What is Spark, anyway?**

Spark is a platform for cluster computing. Spark lets you spread data and computations over *clusters* with multiple *nodes* (think of each node as a separate computer). Splitting up your data makes it easier to work with very large datasets because each node only works with a small amount of data.

As each node works on its own subset of the total data, it also carries out a part of the total calculations required, so that both data processing and computation are performed *in parallel* over the nodes in the cluster. It is a fact that parallel computation can make certain types of programming tasks much faster.

However, with greater computing power comes greater complexity.

Deciding whether or not Spark is the best solution for your problem takes some experience, but you can consider questions like:

* Is my data too big to work with on a single machine?
* Can my calculations be easily parallelized?

Are you excited to learn more about Spark?

**Answer the question**

**50 XP**

**Possible Answers**

* 

No way!

press1

* 

Yes way!

press2

Submit Answer

Take Hint (-15xp)

**Incorrect Submission**

That's not a very good attitude.

**Using Spark in Python**

The first step in using Spark is connecting to a cluster.

In practice, the cluster will be hosted on a remote machine that's connected to all other nodes. There will be one computer, called the *master* that manages splitting up the data and the computations. The master is connected to the rest of the computers in the cluster, which are called *worker*. The master sends the workers data and calculations to run, and they send their results back to the master.

When you're just getting started with Spark it's simpler to just run a cluster locally. Thus, for this course, instead of connecting to another computer, all computations will be run on DataCamp's servers in a simulated cluster.

Creating the connection is as simple as creating an instance of the SparkContext class. The class constructor takes a few optional arguments that allow you to specify the attributes of the cluster you're connecting to.

An object holding all these attributes can be created with the SparkConf() constructor. Take a look at the [**documentation**](http://spark.apache.org/docs/2.1.0/api/python/pyspark.html) for all the details!

For the rest of this course you'll have a SparkContext called sc already available in your workspace.

How do you connect to a Spark cluster from PySpark?

**Answer the question**

**50 XP**

**Possible Answers**

* 

Create an instance of the SparkContext class.

press1

* 

Import the pyspark library.

press2

* 

Plug your computer into the cluster.

press3

**Exercise**

**Exercise**

**Examining The SparkContext**

In this exercise you'll get familiar with the SparkContext.

You'll probably notice that code takes longer to run than you might expect. This is because Spark is some serious software. It takes more time to start up than you might be used to. You may also find that running simpler computations might take longer than expected. That's because all the optimizations that Spark has under its hood are designed for complicated operations with big data sets. That means that for simple or small problems Spark may actually perform worse than some other solutions!

**Instructions**

**100 XP**

Get to know the SparkContext.

* Call print() on sc to verify there's a SparkContext in your environment.
* print() sc.version to see what version of Spark is running on your cluster.

[**Take Hint (-30 XP)**](javascript:void(0))

# Verify SparkContext

print(sc)

# Print Spark version

print(sc.version)

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

<SparkContext master=local[\*] appName=pyspark-shell>

2.3.1

In [1]:

**Using DataFrames**

Spark's core data structure is the Resilient Distributed Dataset (RDD). This is a low level object that lets Spark work its magic by splitting data across multiple nodes in the cluster. However, RDDs are hard to work with directly, so in this course you'll be using the Spark DataFrame abstraction built on top of RDDs.

The Spark DataFrame was designed to behave a lot like a SQL table (a table with variables in the columns and observations in the rows). Not only are they easier to understand, DataFrames are also more optimized for complicated operations than RDDs.

When you start modifying and combining columns and rows of data, there are many ways to arrive at the same result, but some often take much longer than others. When using RDDs, it's up to the data scientist to figure out the right way to optimize the query, but the DataFrame implementation has much of this optimization built in!

To start working with Spark DataFrames, you first have to create a SparkSession object from your SparkContext. You can think of the SparkContext as your connection to the cluster and the SparkSession as your interface with that connection.

Remember, for the rest of this course you'll have a SparkSession called spark available in your workspace!

Which of the following is an advantage of Spark DataFrames over RDDs?

**Answer the question**

**50 XP**

**Possible Answers**

* 

Operations using DataFrames are automatically optimized.

press1

* 

They are smaller.

press2

* 

They can perform more kinds of operations.

press3

* 

They can hold more kinds of data.

press4

Submit Answer

Take Hint (-15xp)

+50 XP

Exactly! This is another way DataFrames are like SQL tables.

**Exercise**

**Exercise**

**Creating a SparkSession**

We've already created a SparkSession for you called spark, but what if you're not sure there already is one? Creating multiple SparkSessions and SparkContexts can cause issues, so it's best practice to use the SparkSession.builder.getOrCreate() method. This returns an existing SparkSession if there's already one in the environment, or creates a new one if necessary!

**Instructions**

**100 XP**

* Import SparkSession from pyspark.sql.
* Make a new SparkSession called my\_spark using SparkSession.builder.getOrCreate().
* Print my\_spark to the console to verify it's a SparkSession.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import SparkSession from pyspark.sql

from pyspark.sql import SparkSession

# Create my\_spark

my\_spark = SparkSession.builder.getOrCreate()

# Print my\_spark

print(my\_spark)

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

<pyspark.sql.session.SparkSession object at 0x7fb38e6d3eb8>

In [1]:

**Exercise**

**Exercise**

**Viewing tables**

Once you've created a SparkSession, you can start poking around to see what data is in your cluster!

Your SparkSession has an attribute called catalog which lists all the data inside the cluster. This attribute has a few methods for extracting different pieces of information.

One of the most useful is the .listTables() method, which returns the names of all the tables in your cluster as a list.

**Instructions**

**100 XP**

* See what tables are in your cluster by calling spark.catalog.listTables() and printing the result!

[**Take Hint (-30 XP)**](javascript:void(0))

# Print the tables in the catalog

print(spark.catalog.listTables())

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

[Table(name='flights', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]

In [1]:

**Exercise**

**Exercise**

**Are you query-ious?**

One of the advantages of the DataFrame interface is that you can run SQL queries on the tables in your Spark cluster. If you don't have any experience with SQL, don't worry, we'll provide you with queries! (To learn more SQL, start with our [**Introduction to SQL**](https://www.datacamp.com/courses/intro-to-sql-for-data-science) course.)

As you saw in the last exercise, one of the tables in your cluster is the flights table. This table contains a row for every flight that left Portland International Airport (PDX) or Seattle-Tacoma International Airport (SEA) in 2014 and 2015.

Running a query on this table is as easy as using the .sql() method on your SparkSession. This method takes a string containing the query and returns a DataFrame with the results!

If you look closely, you'll notice that the table flights is only mentioned in the query, not as an argument to any of the methods. This is because there isn't a local object in your environment that holds that data, so it wouldn't make sense to pass the table as an argument.

Remember, we've already created a SparkSession called spark in your workspace. (It's no longer called my\_spark because we created it for you!)

**Instructions**

**100 XP**

* Use the .sql() method to get the first 10 rows of the flights table and save the result to flights10. The variable query contains the appropriate SQL query.
* Use the DataFrame method .show() to print flights10.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call spark.sql()?

# Don't change this query

query = "FROM flights SELECT \* LIMIT 10"

# Get the first 10 rows of flights

flights10 = spark.sql(query)

# Show the results

flights10.show()

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights10 = query.sql()

AttributeError: 'str' object has no attribute 'sql'

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

In [1]:

**Exercise**

**Exercise**

**Pandafy a Spark DataFrame**

Suppose you've run a query on your huge dataset and aggregated it down to something a little more manageable.

Sometimes it makes sense to then take that table and work with it locally using a tool like pandas. Spark DataFrames make that easy with the .toPandas() method. Calling this method on a Spark DataFrame returns the corresponding pandas DataFrame. It's as simple as that!

This time the query counts the number of flights to each airport from SEA and PDX.

Remember, there's already a SparkSession called spark in your workspace!

**Instructions**

**100 XP**

* Run the query using the .sql() method. Save the result in flight\_counts.
* Use the .toPandas() method on flight\_counts to create a pandas DataFrame called pd\_counts.
* Print the .head() of pd\_counts to the console.

[**Take Hint (-30 XP)**](javascript:void(0))

# Don't change this query

query = "SELECT origin, dest, COUNT(\*) as N FROM flights GROUP BY origin, dest"

# Run the query

flight\_counts = spark.sql(query)

# Convert the results to a pandas DataFrame

pd\_counts = flight\_counts.toPandas()

# Print the head of pd\_counts

print(pd\_counts.head())

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: spark

Out[1]: <pyspark.sql.session.SparkSession at 0x7fe798200240>

In [2]: spark.sql(query)

Out[2]: DataFrame[year: string, month: string, day: string, dep\_time: string, dep\_delay: string, arr\_time: string, arr\_delay: string, carrier: string, tailnum: string, flight: string, origin: string, dest: string, air\_time: string, distance: string, hour: string, minute: string]

In [3]: type(spark.sql(query))

Out[3]: pyspark.sql.dataframe.DataFrame

In [4]: query

Out[4]: 'FROM flights SELECT \* LIMIT 10'

In [5]: spark.toPandas(spark.sql(query))

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

spark.toPandas(spark.sql(query))

AttributeError: 'SparkSession' object has no attribute 'toPandas'

In [6]: (spark.sql(query)).toPandas

Out[6]: <bound method DataFrame.toPandas of DataFrame[year: string, month: string, day: string, dep\_time: string, dep\_delay: string, arr\_time: string, arr\_delay: string, carrier: string, tailnum: string, flight: string, origin: string, dest: string, air\_time: string, distance: string, hour: string, minute: string]>

In [7]: (spark.sql(query)).toPandas()

Out[7]:

year month day dep\_time dep\_delay ... dest air\_time distance hour minute

0 2014 12 8 658 -7 ... LAX 132 954 6 58

1 2014 1 22 1040 5 ... HNL 360 2677 10 40

2 2014 3 9 1443 -2 ... SFO 111 679 14 43

3 2014 4 9 1705 45 ... SJC 83 569 17 5

4 2014 3 9 754 -1 ... BUR 127 937 7 54

5 2014 1 15 1037 7 ... DEN 121 991 10 37

6 2014 7 2 847 42 ... OAK 90 543 8 47

7 2014 5 12 1655 -5 ... SFO 98 679 16 55

8 2014 4 19 1236 -4 ... SAN 135 1050 12 36

9 2014 11 19 1812 -3 ... ORD 198 1721 18 12

[10 rows x 16 columns]

In [8]: (spark.sql(query))

Out[8]: DataFrame[year: string, month: string, day: string, dep\_time: string, dep\_delay: string, arr\_time: string, arr\_delay: string, carrier: string, tailnum: string, flight: string, origin: string, dest: string, air\_time: string, distance: string, hour: string, minute: string]

In [9]: type(spark.sql(query).show())

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

Out[9]: NoneType

<script.py> output:

origin dest N

0 SEA RNO 8

1 SEA DTW 98

2 SEA CLE 2

3 SEA LAX 450

4 PDX SEA 144

In [10]:

**Exercise**

**Exercise**

**Put some Spark in your data**

In the last exercise, you saw how to move data from Spark to pandas. However, maybe you want to go the other direction, and put a pandas DataFrame into a Spark cluster! The SparkSession class has a method for this as well.

The .createDataFrame() method takes a pandas DataFrame and returns a Spark DataFrame.

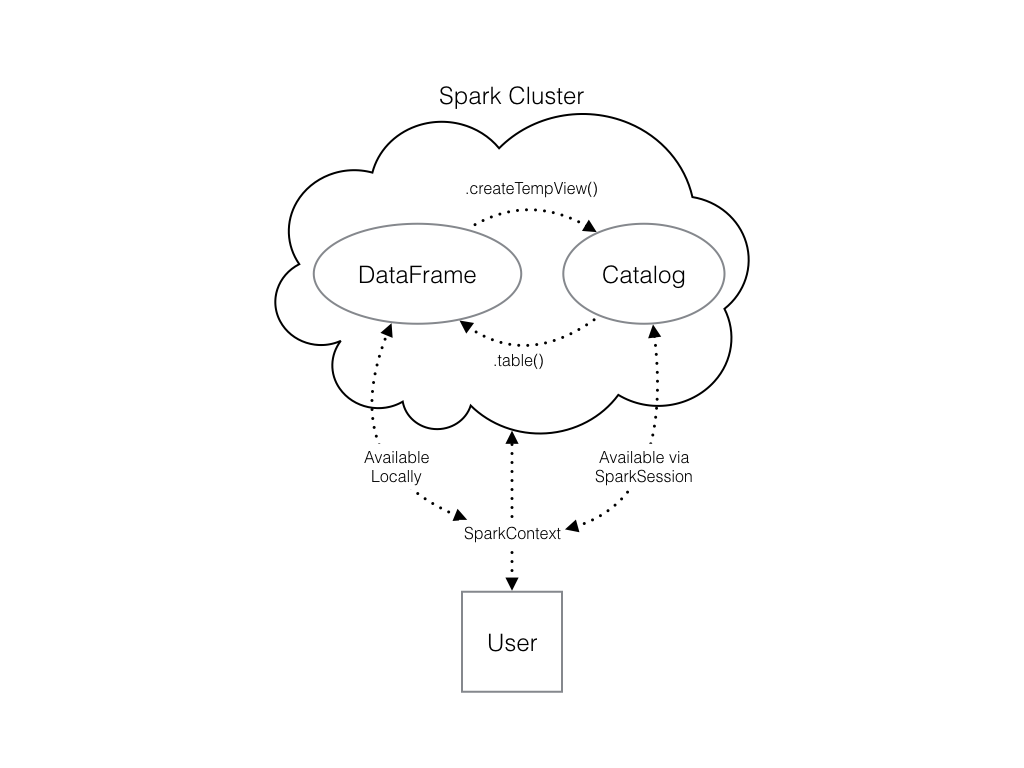
The output of this method is stored locally, not in the SparkSession catalog. This means that you can use all the Spark DataFrame methods on it, but you can't access the data in other contexts.

For example, a SQL query (using the .sql() method) that references your DataFrame will throw an error. To access the data in this way, you have to save it as a *temporary table*.

You can do this using the .createTempView() Spark DataFrame method, which takes as its only argument the name of the temporary table you'd like to register. This method registers the DataFrame as a table in the catalog, but as this table is temporary, it can only be accessed from the specific SparkSession used to create the Spark DataFrame.

There is also the method .createOrReplaceTempView(). This safely creates a new temporary table if nothing was there before, or updates an existing table if one was already defined. You'll use this method to avoid running into problems with duplicate tables.

Check out the diagram to see all the different ways your Spark data structures interact with each other.



There's already a SparkSession called spark in your workspace, numpy has been imported as np, and pandas as pd.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* The code to create a pandas DataFrame of random numbers has already been provided and saved under pd\_temp.
* Create a Spark DataFrame called spark\_temp by calling the .createDataFrame() method with pd\_temp as the argument.
* Examine the list of tables in your Spark cluster and verify that the new DataFrame is *not* present. Remember you can use spark.catalog.listTables() to do so.
* Register spark\_temp as a temporary table named "temp" using the .createOrReplaceTempView() method. Remember that the table name is set including it as the only argument!
* Examine the list of tables again!

[**Take Hint (-30 XP)**](javascript:void(0))

**Hint**

Remember, you can use .createDataFrame() to create a Spark DataFrame.

# Create pd\_temp

pd\_temp = pd.DataFrame(np.random.random(10))

# Create spark\_temp from pd\_temp

spark\_temp = spark.createDataFrame(pd\_temp)

# Examine the tables in the catalog

print(spark.catalog.listTables())

# Add spark\_temp to the catalog

spark\_temp.createOrReplaceTempView("temp")

# Examine the tables in the catalog again

print(spark.catalog.listTables())

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: SparkSession

Out[1]: pyspark.sql.session.SparkSession

In [2]: spark

Out[2]: <pyspark.sql.session.SparkSession at 0x7f70f8da1f98>

In [3]:

<script.py> output:

[]

[Table(name='temp', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]

In [3]:

+0 XP

Awesome! Now you can get your data in and out of Spark.

**Exercise**

**Exercise**

**Dropping the middle man**

Now you know how to put data into Spark via pandas, but you're probably wondering why deal with pandas at all? Wouldn't it be easier to just read a text file straight into Spark? Of course it would!

Luckily, your SparkSession has a .read attribute which has several methods for reading different data sources into Spark DataFrames. Using these you can create a DataFrame from a .csv file just like with regular pandas DataFrames!

The variable file\_path is a string with the path to the file airports.csv. This file contains information about different airports all over the world.

A SparkSession named spark is available in your workspace.

**Instructions**

**100 XP**

* Use the .read.csv() method to create a Spark DataFrame called airports
  + The first argument is file\_path
  + Pass the argument header=True so that Spark knows to take the column names from the first line of the file.
* Print out this DataFrame by calling .show().

[**Take Hint (-30 XP)**](javascript:void(0))

# Don't change this file path

file\_path = "/usr/local/share/datasets/airports.csv"

# Read in the airports data

airports = spark.read.csv(file\_path, header=True)

# Show the data

airports.show()

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

+---+--------------------+----------------+-----------------+----+---+---+

|faa| name| lat| lon| alt| tz|dst|

+---+--------------------+----------------+-----------------+----+---+---+

|04G| Lansdowne Airport| 41.1304722| -80.6195833|1044| -5| A|

|06A|Moton Field Munic...| 32.4605722| -85.6800278| 264| -5| A|

|06C| Schaumburg Regional| 41.9893408| -88.1012428| 801| -6| A|

|06N| Randall Airport| 41.431912| -74.3915611| 523| -5| A|

|09J|Jekyll Island Air...| 31.0744722| -81.4277778| 11| -4| A|

|0A9|Elizabethton Muni...| 36.3712222| -82.1734167|1593| -4| A|

|0G6|Williams County A...| 41.4673056| -84.5067778| 730| -5| A|

|0G7|Finger Lakes Regi...| 42.8835647| -76.7812318| 492| -5| A|

|0P2|Shoestring Aviati...| 39.7948244| -76.6471914|1000| -5| U|

|0S9|Jefferson County ...| 48.0538086| -122.8106436| 108| -8| A|

|0W3|Harford County Ai...| 39.5668378| -76.2024028| 409| -5| A|

|10C| Galt Field Airport| 42.4028889| -88.3751111| 875| -6| U|

|17G|Port Bucyrus-Craw...| 40.7815556| -82.9748056|1003| -5| A|

|19A|Jackson County Ai...| 34.1758638| -83.5615972| 951| -4| U|

|1A3|Martin Campbell F...| 35.0158056| -84.3468333|1789| -4| A|

|1B9| Mansfield Municipal| 42.0001331| -71.1967714| 122| -5| A|

|1C9|Frazier Lake Airpark|54.0133333333333|-124.768333333333| 152| -8| A|

|1CS|Clow Internationa...| 41.6959744| -88.1292306| 670| -6| U|

|1G3| Kent State Airport| 41.1513889| -81.4151111|1134| -4| A|

|1OH| Fortman Airport| 40.5553253| -84.3866186| 885| -5| U|

+---+--------------------+----------------+-----------------+----+---+---+

only showing top 20 rows

In [1]:

+100 XP

Awesome job! You've got the basics of Spark under your belt!

**Exercise**

**Exercise**

**Creating columns**

In this chapter, you'll learn how to use the methods defined by Spark's DataFrame class to perform common data operations.

Let's look at performing column-wise operations. In Spark you can do this using the .withColumn() method, which takes two arguments. First, a string with the name of your new column, and second the new column itself.

The new column must be an object of class Column. Creating one of these is as easy as extracting a column from your DataFrame using df.colName.

Updating a Spark DataFrame is somewhat different than working in pandas because the Spark DataFrame is *immutable*. This means that it can't be changed, and so columns can't be updated in place.

Thus, all these methods return a new DataFrame. To overwrite the original DataFrame you must reassign the returned DataFrame using the method like so:

df = df.withColumn("newCol", df.oldCol + 1)

The above code creates a DataFrame with the same columns as df plus a new column, newCol, where every entry is equal to the corresponding entry from oldCol, plus one.

To overwrite an existing column, just pass the name of the column as the first argument!

Remember, a SparkSession called spark is already in your workspace.

**Instructions**

**100 XP**

* Use the spark.table() method with the argument "flights" to create a DataFrame containing the values of the flights table in the .catalog. Save it as flights.
* Show the head of flights using flights.show(). The column air\_time contains the duration of the flight in minutes.
* Update flights to include a new column called duration\_hrs, that contains the duration of each flight in hours.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you show the head of flights correctly?

Did you add duration\_hrs to flights correctly?

**Hint**

To get the duration of each flight in hours, you can do flights.air\_time/60

# Create the DataFrame flights

flights = spark.table("flights")

# Show the head

flights.show()

# Add duration\_hrs

flights = flights.withColumn('duration\_hrs', flights.air\_time/60)

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.catalog()

File "script.py", line 1182, in \_\_getattr\_\_

"'%s' object has no attribute '%s'" % (self.\_\_class\_\_.\_\_name\_\_, name))

AttributeError: 'DataFrame' object has no attribute 'catalog'

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.catalog(air\_time)

File "script.py", line 1182, in \_\_getattr\_\_

"'%s' object has no attribute '%s'" % (self.\_\_class\_\_.\_\_name\_\_, name))

AttributeError: 'DataFrame' object has no attribute 'catalog'

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights(air\_time).catalog

NameError: name 'air\_time' is not defined

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights('air\_time').catalog

TypeError: 'DataFrame' object is not callable

Traceback (most recent call last):

File "script.py", line 8, in <module>

flights = flights.withColumn('duration\_hrs')

TypeError: withColumn() missing 1 required positional argument: 'col'

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.catalog('air\_time')

File "script.py", line 1182, in \_\_getattr\_\_

"'%s' object has no attribute '%s'" % (self.\_\_class\_\_.\_\_name\_\_, name))

AttributeError: 'DataFrame' object has no attribute 'catalog'

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 11| 8| 1653| -2| 1924| -1| AS| N323AS| 448| SEA| LAX| 130| 954| 16| 53|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 10| 30| 811| 21| 1038| 29| AS| N433AS| 608| SEA| LAS| 127| 867| 8| 11|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 10| 31| 1314| 89| 1544| 111| AS| N713AS| 306| SEA| SFO| 129| 679| 13| 14|

|2014| 1| 29| 2009| 3| 2159| 9| UA| N27205| 1458| PDX| SFO| 90| 550| 20| 9|

|2014| 12| 17| 2015| 50| 2150| 41| AS| N626AS| 368| SEA| SMF| 76| 605| 20| 15|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 6| 5| 1733| -12| 1945| -10| OO| N215AG| 3488| PDX| BUR| 111| 817| 17| 33|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

Traceback (most recent call last):

File "script.py", line 8, in <module>

flights = flights.withColumn('duration\_hrs')

TypeError: withColumn() missing 1 required positional argument: 'col'

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 11| 8| 1653| -2| 1924| -1| AS| N323AS| 448| SEA| LAX| 130| 954| 16| 53|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 10| 30| 811| 21| 1038| 29| AS| N433AS| 608| SEA| LAS| 127| 867| 8| 11|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 10| 31| 1314| 89| 1544| 111| AS| N713AS| 306| SEA| SFO| 129| 679| 13| 14|

|2014| 1| 29| 2009| 3| 2159| 9| UA| N27205| 1458| PDX| SFO| 90| 550| 20| 9|

|2014| 12| 17| 2015| 50| 2150| 41| AS| N626AS| 368| SEA| SMF| 76| 605| 20| 15|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 6| 5| 1733| -12| 1945| -10| OO| N215AG| 3488| PDX| BUR| 111| 817| 17| 33|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

Traceback (most recent call last):

File "script.py", line 8, in <module>

flights = (flights.air\_time/60).withColumn('duration\_hrs')

TypeError: 'Column' object is not callable

Traceback (most recent call last):

File "script.py", line 5, in <module>

(flights.air\_time/60).show()

TypeError: 'Column' object is not callable

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 11| 8| 1653| -2| 1924| -1| AS| N323AS| 448| SEA| LAX| 130| 954| 16| 53|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 10| 30| 811| 21| 1038| 29| AS| N433AS| 608| SEA| LAS| 127| 867| 8| 11|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 10| 31| 1314| 89| 1544| 111| AS| N713AS| 306| SEA| SFO| 129| 679| 13| 14|

|2014| 1| 29| 2009| 3| 2159| 9| UA| N27205| 1458| PDX| SFO| 90| 550| 20| 9|

|2014| 12| 17| 2015| 50| 2150| 41| AS| N626AS| 368| SEA| SMF| 76| 605| 20| 15|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 6| 5| 1733| -12| 1945| -10| OO| N215AG| 3488| PDX| BUR| 111| 817| 17| 33|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

Traceback (most recent call last):

File "script.py", line 8, in <module>

flights = flights.withColumn('duration\_hrs')

TypeError: withColumn() missing 1 required positional argument: 'col'

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 11| 8| 1653| -2| 1924| -1| AS| N323AS| 448| SEA| LAX| 130| 954| 16| 53|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 10| 30| 811| 21| 1038| 29| AS| N433AS| 608| SEA| LAS| 127| 867| 8| 11|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 10| 31| 1314| 89| 1544| 111| AS| N713AS| 306| SEA| SFO| 129| 679| 13| 14|

|2014| 1| 29| 2009| 3| 2159| 9| UA| N27205| 1458| PDX| SFO| 90| 550| 20| 9|

|2014| 12| 17| 2015| 50| 2150| 41| AS| N626AS| 368| SEA| SMF| 76| 605| 20| 15|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 6| 5| 1733| -12| 1945| -10| OO| N215AG| 3488| PDX| BUR| 111| 817| 17| 33|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

Traceback (most recent call last):

File "script.py", line 8, in <module>

flights = flights.withColumn('duration\_hrs')

TypeError: withColumn() missing 1 required positional argument: 'col'

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 11| 8| 1653| -2| 1924| -1| AS| N323AS| 448| SEA| LAX| 130| 954| 16| 53|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 10| 30| 811| 21| 1038| 29| AS| N433AS| 608| SEA| LAS| 127| 867| 8| 11|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 10| 31| 1314| 89| 1544| 111| AS| N713AS| 306| SEA| SFO| 129| 679| 13| 14|

|2014| 1| 29| 2009| 3| 2159| 9| UA| N27205| 1458| PDX| SFO| 90| 550| 20| 9|

|2014| 12| 17| 2015| 50| 2150| 41| AS| N626AS| 368| SEA| SMF| 76| 605| 20| 15|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 6| 5| 1733| -12| 1945| -10| OO| N215AG| 3488| PDX| BUR| 111| 817| 17| 33|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

Traceback (most recent call last):

File "script.py", line 8, in <module>

flights = flights.withColumn(flights.air\_time/60, 'duration\_hrs')

File "script.py", line 1848, in withColumn

assert isinstance(col, Column), "col should be Column"

AssertionError: col should be Column

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| LAX| 132| 954| 6| 58|

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| SFO| 111| 679| 14| 43|

|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| SJC| 83| 569| 17| 5|

|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| BUR| 127| 937| 7| 54|

|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| DEN| 121| 991| 10| 37|

|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| OAK| 90| 543| 8| 47|

|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| SFO| 98| 679| 16| 55|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 11| 8| 1653| -2| 1924| -1| AS| N323AS| 448| SEA| LAX| 130| 954| 16| 53|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 10| 30| 811| 21| 1038| 29| AS| N433AS| 608| SEA| LAS| 127| 867| 8| 11|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 10| 31| 1314| 89| 1544| 111| AS| N713AS| 306| SEA| SFO| 129| 679| 13| 14|

|2014| 1| 29| 2009| 3| 2159| 9| UA| N27205| 1458| PDX| SFO| 90| 550| 20| 9|

|2014| 12| 17| 2015| 50| 2150| 41| AS| N626AS| 368| SEA| SMF| 76| 605| 20| 15|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 6| 5| 1733| -12| 1945| -10| OO| N215AG| 3488| PDX| BUR| 111| 817| 17| 33|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

In [1]:

+70 XP

Good job! Now you can make new columns derived from the old ones!

**SQL in a nutshell**

As you move forward, it will help to have a basic understanding of SQL. A more in depth look can be found [**here**](https://www.datacamp.com/courses/intro-to-sql-for-data-science).

A SQL query returns a table derived from one or more tables contained in a database.

Every SQL query is made up of commands that tell the database what you want to do with the data. The two commands that every query has to contain are SELECT and FROM.

The SELECT command is followed by the *columns* you want in the resulting table.

The FROM command is followed by the name of the table that contains those columns. The minimal SQL query is:

SELECT \* FROM my\_table;

The \* selects all columns, so this returns the entire table named my\_table.

Similar to .withColumn(), you can do column-wise computations within a SELECT statement. For example,

SELECT origin, dest, air\_time / 60 FROM flights;

returns a table with the origin, destination, and duration in hours for each flight.

Another commonly used command is WHERE. This command filters the rows of the table based on some logical condition you specify. The resulting table contains the rows where your condition is true. For example, if you had a table of students and grades you could do:

SELECT \* FROM students

WHERE grade = 'A';

to select all the columns and the rows containing information about students who got As.

Which of the following queries returns a table of tail numbers and destinations for flights that lasted more than 10 hours?

**Answer the question**

**50 XP**

**Possible Answers**

SELECT dest, tail\_num FROM flights WHERE air\_time > 10;

press

1

SELECT dest, tail\_num FROM flights WHERE air\_time > 600;

press

2

SELECT \* FROM flights WHERE air\_time > 600;

press

3

**SQL in a nutshell**

As you move forward, it will help to have a basic understanding of SQL. A more in depth look can be found [**here**](https://www.datacamp.com/courses/intro-to-sql-for-data-science).

A SQL query returns a table derived from one or more tables contained in a database.

Every SQL query is made up of commands that tell the database what you want to do with the data. The two commands that every query has to contain are SELECT and FROM.

The SELECT command is followed by the *columns* you want in the resulting table.

The FROM command is followed by the name of the table that contains those columns. The minimal SQL query is:

SELECT \* FROM my\_table;

The \* selects all columns, so this returns the entire table named my\_table.

Similar to .withColumn(), you can do column-wise computations within a SELECT statement. For example,

SELECT origin, dest, air\_time / 60 FROM flights;

returns a table with the origin, destination, and duration in hours for each flight.

Another commonly used command is WHERE. This command filters the rows of the table based on some logical condition you specify. The resulting table contains the rows where your condition is true. For example, if you had a table of students and grades you could do:

SELECT \* FROM students

WHERE grade = 'A';

to select all the columns and the rows containing information about students who got As.

Which of the following queries returns a table of tail numbers and destinations for flights that lasted more than 10 hours?

**Answer the question**

**50 XP**

**Possible Answers**

SELECT dest, tail\_num FROM flights WHERE air\_time > 10;

press1

SELECT dest, tail\_num FROM flights WHERE air\_time > 600;

press2

SELECT \* FROM flights WHERE air\_time > 600;

press3

Submit Answer

Take Hint (-15xp)

**Incorrect Submission**

That's almost right, but remember air\_time is measured in minutes.

Did you find this hint helpful?

YesNo

Awesome, thanks for your feedback!

**SQL in a nutshell (2)**

Another common database task is aggregation. That is, reducing your data by breaking it into chunks and summarizing each chunk.

This is done in SQL using the GROUP BY command. This command breaks your data into groups and applies a function from your SELECT statement to each group.

For example, if you wanted to count the number of flights from each of two origin destinations, you could use the query

SELECT COUNT(\*) FROM flights

GROUP BY origin;

GROUP BY origin tells SQL that you want the output to have a row for each unique value of the origin column. The SELECT statement selects the values you want to populate each of the columns. Here, we want to COUNT() every row in each of the groups.

It's possible to GROUP BY more than one column. When you do this, the resulting table has a row for every combination of the unique values in each column. The following query counts the number of flights from SEA and PDX to every destination airport:

SELECT origin, dest, COUNT(\*) FROM flights

GROUP BY origin, dest;

The output will have a row for every combination of the values in origin and dest (i.e. a row listing each origin and destination that a flight flew to). There will also be a column with the COUNT() of all the rows in each group.

Remember, a more in depth look at SQL can be found [**here**](https://www.datacamp.com/courses/intro-to-sql-for-data-science).

What information would this query get? Remember the flights table holds information about flights that departed PDX and SEA in 2014 and 2015. Note that AVG() function gets the average value of a column!

SELECT AVG(air\_time) / 60 FROM flights

GROUP BY origin, carrier;

**Answer the question**

**50 XP**

**Possible Answers**

The average length of each airline's flights from SEA and from PDX in hours.

press

1

The average length of each flight.

press

2

The average length of each airline's flights.

press

3

**Incorrect Submission**

That's almost right, but take a second look at the GROUP BY statement.

Hm... what's the average length of a single flight?

**Exercise**

**Exercise**

**Filtering Data**

Now that you have a bit of SQL know-how under your belt, it's easier to talk about the analogous operations using Spark DataFrames.

Let's take a look at the .filter() method. As you might suspect, this is the Spark counterpart of SQL's WHERE clause. The .filter() method takes either an expression that would follow the WHERE clause of a SQL expression as a string, or a Spark Column of boolean (True/False) values.

For example, the following two expressions will produce the same output:

flights.filter("air\_time > 120").show()

flights.filter(flights.air\_time > 120).show()

Notice that in the first case, we pass a *string* to .filter(). In SQL, we would write this filtering task as SELECT \* FROM flights WHERE air\_time > 120. Spark's .filter() can accept any expression that could go in the WHEREclause of a SQL query (in this case, "air\_time > 120"), as long as it is passed as a string. Notice that in this case, we do not reference the name of the table in the string -- as we wouldn't in the SQL request.

In the second case, we actually pass a *column of boolean values* to .filter(). Remember that flights.air\_time > 120 returns a column of boolean values that has True in place of those records in flights.air\_time that are over 120, and False otherwise.

Remember, a SparkSession called spark is already in your workspace, along with the Spark DataFrame flights.

**Instructions**

**100 XP**

* Use the .filter() method to find all the flights that flew over 1000 miles two ways:
  + First, pass a SQL **string** to .filter() that checks whether the distance is greater than 1000. Save this as long\_flights1.
  + Then pass a column of boolean values to .filter() that checks the same thing. Save this as long\_flights2.
* Use .show() to print heads of both DataFrames and make sure they're actually equal!

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you use .show() to examine long\_flights1?

Did you define long\_flights1 correctly? Did you pass a **string** as the argument to the first call of `.filter()?

Remember, you can generate a column of boolean values with flights.distance > 1000.

# Filter flights by passing a string

long\_flights1 = flights.filter("distance > 1000")

# Filter flights by passing a column of boolean values

long\_flights2 = flights.filter(flights.distance > 1000)

# Print the data to check they're equal

long\_flights1.show()

long\_flights2.show()

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

DataFrame[year: string, month: string, day: string, dep\_time: string, dep\_delay: string, arr\_time: string, arr\_delay: string, carrier: string, tailnum: string, flight: string, origin: string, dest: string, air\_time: string, distance: string, hour: string, minute: string]

DataFrame[year: string, month: string, day: string, dep\_time: string, dep\_delay: string, arr\_time: string, arr\_delay: string, carrier: string, tailnum: string, flight: string, origin: string, dest: string, air\_time: string, distance: string, hour: string, minute: string]

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

None

None

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

None

None

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

None

None

<script.py> output:

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|dest|air\_time|distance|hour|minute|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| HNL| 360| 2677| 10| 40|

|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| SAN| 135| 1050| 12| 36|

|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| ORD| 198| 1721| 18| 12|

|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| PHX| 154| 1107| 11| 20|

|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| ANC| 183| 1448| 23| 46|

|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| MDW| 216| 1733| 10| 17|

|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| BOS| 290| 2496| 21| 56|

|2014| 9| 26| 610| -5| 1523| 65| US| N127UW| 616| SEA| PHL| 293| 2378| 6| 10|

|2014| 12| 4| 954| -6| 1348| -17| HA| N395HA| 29| SEA| OGG| 333| 2640| 9| 54|

|2014| 6| 4| 1115| 0| 1346| -3| AS| N461AS| 488| SEA| SAN| 133| 1050| 11| 15|

|2014| 6| 26| 2054| -1| 2318| -6| B6| N590JB| 907| SEA| ANC| 179| 1448| 20| 54|

|2014| 6| 7| 1823| -7| 2112| -28| AS| N512AS| 815| SEA| LIH| 335| 2701| 18| 23|

|2014| 4| 30| 801| 1| 1757| 90| AS| N407AS| 18| SEA| MCO| 342| 2554| 8| 1|

|2014| 11| 29| 905| 155| 1655| 170| DL| N824DN| 1598| SEA| ATL| 229| 2182| 9| 5|

|2014| 6| 2| 2222| 7| 55| 15| AS| N402AS| 99| SEA| ANC| 190| 1448| 22| 22|

|2014| 11| 15| 1034| -6| 1414| -26| AS| N589AS| 794| SEA| ABQ| 139| 1180| 10| 34|

|2014| 10| 20| 1328| -1| 1949| 4| UA| N68805| 1212| SEA| IAH| 228| 1874| 13| 28|

|2014| 12| 16| 1500| 0| 1906| 19| US| N662AW| 500| SEA| PHX| 151| 1107| 15| 0|

|2014| 11| 19| 1319| -6| 1821| -14| DL| N309US| 2164| PDX| MSP| 169| 1426| 13| 19|

|2014| 5| 21| 515| 0| 757| 0| US| N172US| 593| SEA| PHX| 143| 1107| 5| 15|

+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+----+--------+--------+----+------+

only showing top 20 rows

In [1]:

+0 XP

Awesome! PySpark often provides a few different ways to get the same results.

**Exercise**

**Exercise**

**Selecting**

The Spark variant of SQL's SELECT is the .select() method. This method takes multiple arguments - one for each column you want to select. These arguments can either be the column name as a string (one for each column) or a column object (using the df.colName syntax). When you pass a column object, you can perform operations like addition or subtraction on the column to change the data contained in it, much like inside .withColumn().

The difference between .select() and .withColumn() methods is that .select() returns only the columns you specify, while .withColumn() returns all the columns of the DataFrame in addition to the one you defined. It's often a good idea to drop columns you don't need at the beginning of an operation so that you're not dragging around extra data as you're wrangling. In this case, you would use .select() and not .withColumn().

Remember, a SparkSession called spark is already in your workspace, along with the Spark DataFrame flights.

**Instructions**

**100 XP**

* Select the columns tailnum, origin, and dest from flights by passing the column names as strings. Save this as selected1.
* Select the columns origin, dest, and carrier using the df.colName syntax and then filter the result using both of the filters already defined for you (filterA and filterB) to only keep flights from SEA to PDX. Save this as selected2.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you select the correct set of columns for selected1?   
Did you select the correct set of columns for temp?

# Select the first set of columns

selected1 = flights.select('tailnum', 'origin', 'dest')

# Select the second set of columns

temp = flights.select(df.origin, df.dest, df.carrier)

# Define first filter

filterA = flights.origin == "SEA"

# Define second filter

filterB = flights.dest == "PDX"

# Filter the data, first by filterA then by filterB

selected2 = temp.filter(filterA).filter(filterB)

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

Traceback (most recent call last):

File "script.py", line 2, in <module>

selected1 = flights.select('tailnum, origin, dest')

File "script.py", line 1202, in select

jdf = self.\_jdf.select(self.\_jcols(\*cols))

File "script.py", line 1257, in \_\_call\_\_

answer, self.gateway\_client, self.target\_id, self.name)

File "script.py", line 69, in deco

raise AnalysisException(s.split(': ', 1)[1], stackTrace)

pyspark.sql.utils.AnalysisException: "cannot resolve '`tailnum, origin, dest`' given input columns: [arr\_delay, minute, month, origin, arr\_time, air\_time, day, dep\_delay, carrier, hour, year, dest, flight, dep\_time, tailnum, distance];;\n'Project ['tailnum, origin, dest]\n+- AnalysisBarrier\n +- Relation[year#724,month#725,day#726,dep\_time#727,dep\_delay#728,arr\_time#729,arr\_delay#730,carrier#731,tailnum#732,flight#733,origin#734,dest#735,air\_time#736,distance#737,hour#738,minute#739] csv\n"

Traceback (most recent call last):

File "script.py", line 5, in <module>

temp = flights.select(df.origin, df.dest, df.carrier)

NameError: name 'df' is not defined

In [1]:

+100 XP

Great work! You're speeding right through this course!

**Selecting II**

Similar to SQL, you can also use the .select() method to perform column-wise operations. When you're selecting a column using the df.colName notation, you can perform any column operation and the .select() method will return the transformed column. For example,

flights.select(flights.air\_time/60)

returns a column of flight durations in hours instead of minutes. You can also use the .alias() method to rename a column you're selecting. So if you wanted to .select() the column duration\_hrs (which isn't in your DataFrame) you could do

flights.select((flights.air\_time/60).alias("duration\_hrs"))

The equivalent Spark DataFrame method .selectExpr() takes SQL expressions as a string:

flights.selectExpr("air\_time/60 as duration\_hrs")

with the SQL as keyword being equivalent to the .alias() method. To select multiple columns, you can pass multiple strings.

Remember, a SparkSession called spark is already in your workspace, along with the Spark DataFrame flights.

**Instructions**

**100 XP**

Create a table of the average speed of each flight both ways.

* Calculate average speed by dividing the distance by the air\_time (converted to hours). Use the .alias() method name this column "avg\_speed". Save the output as the variable avg\_speed.
* Select the columns "origin", "dest", "tailnum", and avg\_speed (without quotes!). Save this as speed1.
* Create the same table using .selectExpr() and a string containing a SQL expression. Save this as speed2.

[**Take Hint (-30 XP)**](javascript:void(0))

# Define avg\_speed

avg\_speed = (flights.distance/(flights.air\_time/60)).alias("avg\_speed")

# Select the correct columns

speed1 = flights.select("origin", "dest", "tailnum", avg\_speed)

# Create the same table using a SQL expression

speed2 = flights.selectExpr("origin", "dest", "tailnum", "distance/(air\_time/60) as avg\_speed")

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]:

**Exercise**

**Exercise**

**Aggregating**

All of the common aggregation methods, like .min(), .max(), and .count() are GroupedData methods. These are created by calling the .groupBy() DataFrame method. You'll learn exactly what that means in a few exercises. For now, all you have to do to use these functions is call that method on your DataFrame. For example, to find the minimum value of a column, col, in a DataFrame, df, you could do

df.groupBy().min("col").show()

This creates a GroupedData object (so you can use the .min() method), then finds the minimum value in col, and returns it as a DataFrame.

Now you're ready to do some aggregating of your own!

A SparkSession called spark is already in your workspace, along with the Spark DataFrame flights.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Find the length of the shortest (in terms of distance) flight that left PDX by first .filter()ing and using the .min() method. Perform the filtering by referencing the column directly, not passing a SQL string.
* Find the length of the longest (in terms of time) flight that left SEA by filter()ing and using the .max() method. Perform the filtering by referencing the column directly, not passing a SQL string.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you filter flights to find the shortest flight from PDX by *referencing the column direcly*? Did you find the minimal distance?

**Hint**

* Make sure that you pass a *logical column* as an argument to filter (e.g. filter(df.col < 2)), not a string.
* The .min() and .max() methods, however, take the name of a column as a *string*, e.g. max("col").

script.py

# Find the shortest flight from PDX in terms of distance

flights.filter(flights.origin == 'PDX').groupBy().\_\_\_\_.show()

# Find the longest flight from SEA in terms of air time

flights.filter(flights.origin == 'SEA').groupBy().\_\_\_\_.show()

solution.py

# Find the shortest flight from PDX in terms of distance

flights.filter(flights.origin == "PDX").groupBy().min("distance").show()

# Find the longest flight from SEA in terms of air time

flights.filter(flights.origin == "SEA").groupBy().max("air\_time").show()

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: df

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

df

NameError: name 'df' is not defined

In [2]:

In [2]: flights

Out[2]: DataFrame[year: string, month: string, day: string, dep\_time: string, dep\_delay: string, arr\_time: string, arr\_delay: string, carrier: string, tailnum: string, flight: string, origin: string, dest: string, air\_time: int, distance: int, hour: string, minute: string]

Traceback (most recent call last):

File "script.py", line 2, in <module>

flights.filter('PDX').groupBy().min.show()

File "script.py", line 1240, in filter

jdf = self.\_jdf.filter(condition)

File "script.py", line 1257, in \_\_call\_\_

answer, self.gateway\_client, self.target\_id, self.name)

File "script.py", line 69, in deco

raise AnalysisException(s.split(': ', 1)[1], stackTrace)

pyspark.sql.utils.AnalysisException: "cannot resolve '`PDX`' given input columns: [arr\_delay, dep\_delay, air\_time, minute, distance, day, carrier, month, flight, dest, hour, tailnum, origin, dep\_time, year, arr\_time]; line 1 pos 0;\n'Filter 'PDX\n+- AnalysisBarrier\n +- Project [year#10, month#11, day#12, dep\_time#13, dep\_delay#14, arr\_time#15, arr\_delay#16, carrier#17, tailnum#18, flight#19, origin#20, dest#21, cast(air\_time#22 as int) AS air\_time#59, distance#42, hour#24, minute#25]\n +- Project [year#10, month#11, day#12, dep\_time#13, dep\_delay#14, arr\_time#15, arr\_delay#16, carrier#17, tailnum#18, flight#19, origin#20, dest#21, air\_time#22, cast(distance#23 as int) AS distance#42, hour#24, minute#25]\n +- Relation[year#10,month#11,day#12,dep\_time#13,dep\_delay#14,arr\_time#15,arr\_delay#16,carrier#17,tailnum#18,flight#19,origin#20,dest#21,air\_time#22,distance#23,hour#24,minute#25] csv\n"

In [3]: df.origin

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

df.origin

NameError: name 'df' is not defined

In [4]: flights.origin

Out[4]: Column<b'origin'>

In [5]: list(flights.origin)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

list(flights.origin)

File "<stdin>", line 345, in \_\_iter\_\_

raise TypeError("Column is not iterable")

TypeError: Column is not iterable

Traceback (most recent call last):

File "script.py", line 2, in <module>

flights.filter(flights.origin == 'PDX').groupBy().\_\_\_\_.show()

AttributeError: 'GroupedData' object has no attribute '\_\_\_\_'

Traceback (most recent call last):

File "script.py", line 2, in <module>

flights.filter(flights.origin == 'PDX').groupBy().min.show()

AttributeError: 'function' object has no attribute 'show'

Traceback (most recent call last):

File "script.py", line 2, in <module>

flights.filter(flights.origin == 'PDX').groupBy().distance.show()

AttributeError: 'GroupedData' object has no attribute 'distance'

In [6]:

<script.py> output:

+-------------+

|min(distance)|

+-------------+

| 106|

+-------------+

+-------------+

|max(air\_time)|

+-------------+

| 409|

+-------------+

In [6]:

+0 XP

Fantastic work! How do these methods help you learn about your data?

**Exercise**

**Exercise**

**Aggregating II**

To get you familiar with more of the built in aggregation methods, here's a few more exercises involving the flights table!

Remember, a SparkSession called spark is already in your workspace, along with the Spark DataFrame flights.

**Instructions**

**100 XP**

* Use the .avg() method to get the average air time of Delta Airlines flights (where the carrier column has the value "DL") that left SEA. The place of departure is stored in the column origin. show() the result.
* Use the .sum() method to get the total number of hours all planes in this dataset spent in the air by creating a column called duration\_hrs from the column air\_time. show() the result.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you calculate the average air time of Delta flights correctly?

Did you sum() the total air time correctly?

# Average duration of Delta flights

flights.filter(flights.carrier == "DL").filter(flights.origin == 'SEA').groupBy().avg(air\_time).show()

# Total hours in the air

flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy().sum('air\_time').show()

# Average duration of Delta flights

flights.filter(flights.carrier == "DL").filter(flights.origin == 'SEA').groupBy().avg('air\_time').show()

# Total hours in the air

flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy().sum("duration\_hrs").show()

+0 XP

Stellar job! Now you can answer some interesting questions about the data.

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

Traceback (most recent call last):

File "script.py", line 2, in <module>

flights.filter(flights.carrier == "DL").filter(flights.origin == 'SEA').groupBy().avg(air\_time).show()

NameError: name 'air\_time' is not defined

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

+-------------+

|sum(air\_time)|

+-------------+

| 1517376|

+-------------+

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy().sum(flights.air\_time).show()

File "script.py", line 41, in \_api

jdf = getattr(self.\_jgd, name)(\_to\_seq(self.sql\_ctx.\_sc, cols))

File "script.py", line 67, in \_to\_seq

return sc.\_jvm.PythonUtils.toSeq(cols)

File "script.py", line 1248, in \_\_call\_\_

args\_command, temp\_args = self.\_build\_args(\*args)

File "script.py", line 1212, in \_build\_args

(new\_args, temp\_args) = self.\_get\_args(args)

File "script.py", line 1199, in \_get\_args

temp\_arg = converter.convert(arg, self.gateway\_client)

File "script.py", line 501, in convert

java\_list.add(element)

File "script.py", line 1248, in \_\_call\_\_

args\_command, temp\_args = self.\_build\_args(\*args)

File "script.py", line 1212, in \_build\_args

(new\_args, temp\_args) = self.\_get\_args(args)

File "script.py", line 1199, in \_get\_args

temp\_arg = converter.convert(arg, self.gateway\_client)

File "script.py", line 500, in convert

for element in object:

File "script.py", line 345, in \_\_iter\_\_

raise TypeError("Column is not iterable")

TypeError: Column is not iterable

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy().sum('hour').show()

File "script.py", line 41, in \_api

jdf = getattr(self.\_jgd, name)(\_to\_seq(self.sql\_ctx.\_sc, cols))

File "script.py", line 1257, in \_\_call\_\_

answer, self.gateway\_client, self.target\_id, self.name)

File "script.py", line 69, in deco

raise AnalysisException(s.split(': ', 1)[1], stackTrace)

pyspark.sql.utils.AnalysisException: '"hour" is not a numeric column. Aggregation function can only be applied on a numeric column.;'

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy().sum(flights.hour).show()

File "script.py", line 41, in \_api

jdf = getattr(self.\_jgd, name)(\_to\_seq(self.sql\_ctx.\_sc, cols))

File "script.py", line 67, in \_to\_seq

return sc.\_jvm.PythonUtils.toSeq(cols)

File "script.py", line 1248, in \_\_call\_\_

args\_command, temp\_args = self.\_build\_args(\*args)

File "script.py", line 1212, in \_build\_args

(new\_args, temp\_args) = self.\_get\_args(args)

File "script.py", line 1199, in \_get\_args

temp\_arg = converter.convert(arg, self.gateway\_client)

File "script.py", line 501, in convert

java\_list.add(element)

File "script.py", line 1248, in \_\_call\_\_

args\_command, temp\_args = self.\_build\_args(\*args)

File "script.py", line 1212, in \_build\_args

(new\_args, temp\_args) = self.\_get\_args(args)

File "script.py", line 1199, in \_get\_args

temp\_arg = converter.convert(arg, self.gateway\_client)

File "script.py", line 500, in convert

for element in object:

File "script.py", line 345, in \_\_iter\_\_

raise TypeError("Column is not iterable")

TypeError: Column is not iterable

In [1]: flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy()

Out[1]: <pyspark.sql.group.GroupedData at 0x7f5793863a20>

In [2]: help(flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy().sum)

Help on method sum in module pyspark.sql.group:

sum(\*cols) method of pyspark.sql.group.GroupedData instance

Compute the sum for each numeric columns for each group.

:param cols: list of column names (string). Non-numeric columns are ignored.

>>> df.groupBy().sum('age').collect()

[Row(sum(age)=7)]

>>> df3.groupBy().sum('age', 'height').collect()

[Row(sum(age)=7, sum(height)=165)]

.. versionadded:: 1.3

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.withColumn("duration\_hrs", flights.air\_time/60).groupBy().sum('hour').show()

File "script.py", line 41, in \_api

jdf = getattr(self.\_jgd, name)(\_to\_seq(self.sql\_ctx.\_sc, cols))

File "script.py", line 1257, in \_\_call\_\_

answer, self.gateway\_client, self.target\_id, self.name)

File "script.py", line 69, in deco

raise AnalysisException(s.split(': ', 1)[1], stackTrace)

pyspark.sql.utils.AnalysisException: '"hour" is not a numeric column. Aggregation function can only be applied on a numeric column.;'

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

+-------------+-------------+------------------+

|sum(air\_time)|sum(distance)| sum(duration\_hrs)|

+-------------+-------------+------------------+

| 1517376| 12081516|25289.600000000126|

+-------------+-------------+------------------+

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

Traceback (most recent call last):

File "script.py", line 5, in <module>

flights.withColumn("hour", flights.air\_time/60).groupBy().sum('duration\_hrs').show()

File "script.py", line 41, in \_api

jdf = getattr(self.\_jgd, name)(\_to\_seq(self.sql\_ctx.\_sc, cols))

File "script.py", line 1257, in \_\_call\_\_

answer, self.gateway\_client, self.target\_id, self.name)

File "script.py", line 69, in deco

raise AnalysisException(s.split(': ', 1)[1], stackTrace)

pyspark.sql.utils.AnalysisException: 'Cannot resolve column name "duration\_hrs" among (year, month, day, dep\_time, dep\_delay, arr\_time, arr\_delay, carrier, tailnum, flight, origin, dest, air\_time, distance, hour, minute);'

<script.py> output:

+------------------+

| avg(air\_time)|

+------------------+

|188.20689655172413|

+------------------+

+------------------+

| sum(duration\_hrs)|

+------------------+

|25289.600000000126|

+------------------+

In [3]:

**Exercise**

**Exercise**

**Grouping and Aggregating I**

Part of what makes aggregating so powerful is the addition of groups. PySpark has a whole class devoted to grouped data frames: pyspark.sql.GroupedData, which you saw in the last two exercises.

You've learned how to create a grouped DataFrame by calling the .groupBy() method on a DataFrame with no arguments.

Now you'll see that when you pass the name of one or more columns in your DataFrame to the .groupBy() method, the aggregation methods behave like when you use a GROUP BY statement in a SQL query!

Remember, a SparkSession called spark is already in your workspace, along with the Spark DataFrame flights.

**Instructions**

**100 XP**

* Create a DataFrame called by\_plane that is grouped by the column tailnum.
* Use the .count() method with no arguments to count the number of flights each plane made.
* Create a DataFrame called by\_origin that is grouped by the column origin.
* Find the .avg() of the air\_time column to find average duration of flights from PDX and SEA.

[**Take Hint (-30 XP)**](javascript:void(0))

# Group by tailnum

by\_plane = flights.groupBy("tailnum")

# Number of flights each plane made

by\_plane.count().show()

# Group by origin

by\_origin = flights.groupBy("origin")

# Average duration of flights from PDX and SEA

by\_origin.avg("air\_time").show()

+100 XP

Great work! You're passing with flying colors!

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

+-------+-----+

|tailnum|count|

+-------+-----+

| N442AS| 38|

| N102UW| 2|

| N36472| 4|

| N38451| 4|

| N73283| 4|

| N513UA| 2|

| N954WN| 5|

| N388DA| 3|

| N567AA| 1|

| N516UA| 2|

| N927DN| 1|

| N8322X| 1|

| N466SW| 1|

| N6700| 1|

| N607AS| 45|

| N622SW| 4|

| N584AS| 31|

| N914WN| 4|

| N654AW| 2|

| N336NW| 1|

+-------+-----+

only showing top 20 rows

+------+------------------+

|origin| avg(air\_time)|

+------+------------------+

| SEA| 160.4361496051259|

| PDX|137.11543248288737|

+------+------------------+

In [1]:

**Exercise**

**Exercise**

**Grouping and Aggregating II**

In addition to the GroupedData methods you've already seen, there is also the .agg() method. This method lets you pass an aggregate column expression that uses any of the aggregate functions from the pyspark.sql.functions submodule.

This submodule contains many useful functions for computing things like standard deviations. All the aggregation functions in this submodule take the name of a column in a GroupedData table.

Remember, a SparkSession called spark is already in your workspace, along with the Spark DataFrame flights. The grouped DataFrames you created in the last exercise are also in your workspace.

**Instructions**

**100 XP**

* Import the submodule pyspark.sql.functions as F.
* Create a GroupedData table called by\_month\_dest that's grouped by both the month and dest columns. Refer to the two columns by passing both strings as separate arguments.
* Use the .avg() method on the by\_month\_dest DataFrame to get the average dep\_delay in each month for each destination.
* Find the standard deviation of dep\_delay by using the .agg() method with the function F.stddev().

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you define by\_month\_dest correctly?

Did you call .avg() correctly? Did you average over the correct column?

Did you call stddev() correctly?

# Import pyspark.sql.functions as F

import pyspark.sql.functions as F

# Group by month and dest

by\_month\_dest = flights.groupBy('month', 'dest')

# Average departure delay by month and destination

by\_month\_dest.avg('dep\_delay').show()

# Standard deviation of departure delay

by\_month\_dest.agg(F.stddev('dep\_delay')).show()

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: # Import pyspark.sql.functions as F

import pyspark.sql.functions as F

# Group by month and dest

by\_month\_dest = flights.groupBy(['month', 'dest'])

# Average departure delay by month and destination

by\_month\_dest.\_\_\_\_.show()

# Standard deviation of departure delay

by\_month\_dest.agg(F.\_\_\_\_(\_\_\_\_\_)).show()

Traceback (most recent call last):

File "<stdin>", line 8, in <module>

by\_month\_dest.\_\_\_\_.show()

AttributeError: 'GroupedData' object has no attribute '\_\_\_\_'

In [2]: by\_month\_dest

Out[2]: <pyspark.sql.group.GroupedData at 0x7f506df5e668>

Traceback (most recent call last):

File "script.py", line 8, in <module>

by\_month\_dest.avg(dep\_delay).show()

NameError: name 'dep\_delay' is not defined

Traceback (most recent call last):

File "script.py", line 8, in <module>

by\_month\_dest.avg(dep\_delay).show()

NameError: name 'dep\_delay' is not defined

<script.py> output:

+-----+----+--------------------+

|month|dest| avg(dep\_delay)|

+-----+----+--------------------+

| 11| TUS| -2.3333333333333335|

| 11| ANC| 7.529411764705882|

| 1| BUR| -1.45|

| 1| PDX| -5.6923076923076925|

| 6| SBA| -2.5|

| 5| LAX|-0.15789473684210525|

| 10| DTW| 2.6|

| 6| SIT| -1.0|

| 10| DFW| 18.176470588235293|

| 3| FAI| -2.2|

| 10| SEA| -0.8|

| 2| TUS| -0.6666666666666666|

| 12| OGG| 25.181818181818183|

| 9| DFW| 4.066666666666666|

| 5| EWR| 14.25|

| 3| RDM| -6.2|

| 8| DCA| 2.6|

| 7| ATL| 4.675675675675675|

| 4| JFK| 0.07142857142857142|

| 10| SNA| -1.1333333333333333|

+-----+----+--------------------+

only showing top 20 rows

Traceback (most recent call last):

File "script.py", line 11, in <module>

by\_month\_dest.agg(F.stddev(dep\_delay)).show()

NameError: name 'dep\_delay' is not defined

<script.py> output:

+-----+----+--------------------+

|month|dest| avg(dep\_delay)|

+-----+----+--------------------+

| 11| TUS| -2.3333333333333335|

| 11| ANC| 7.529411764705882|

| 1| BUR| -1.45|

| 1| PDX| -5.6923076923076925|

| 6| SBA| -2.5|

| 5| LAX|-0.15789473684210525|

| 10| DTW| 2.6|

| 6| SIT| -1.0|

| 10| DFW| 18.176470588235293|

| 3| FAI| -2.2|

| 10| SEA| -0.8|

| 2| TUS| -0.6666666666666666|

| 12| OGG| 25.181818181818183|

| 9| DFW| 4.066666666666666|

| 5| EWR| 14.25|

| 3| RDM| -6.2|

| 8| DCA| 2.6|

| 7| ATL| 4.675675675675675|

| 4| JFK| 0.07142857142857142|

| 10| SNA| -1.1333333333333333|

+-----+----+--------------------+

only showing top 20 rows

Traceback (most recent call last):

File "script.py", line 11, in <module>

by\_month\_dest.agg(F.stddev(dep\_delay)).show()

NameError: name 'dep\_delay' is not defined

<script.py> output:

+-----+----+--------------------+

|month|dest| avg(dep\_delay)|

+-----+----+--------------------+

| 11| TUS| -2.3333333333333335|

| 11| ANC| 7.529411764705882|

| 1| BUR| -1.45|

| 1| PDX| -5.6923076923076925|

| 6| SBA| -2.5|

| 5| LAX|-0.15789473684210525|

| 10| DTW| 2.6|

| 6| SIT| -1.0|

| 10| DFW| 18.176470588235293|

| 3| FAI| -2.2|

| 10| SEA| -0.8|

| 2| TUS| -0.6666666666666666|

| 12| OGG| 25.181818181818183|

| 9| DFW| 4.066666666666666|

| 5| EWR| 14.25|

| 3| RDM| -6.2|

| 8| DCA| 2.6|

| 7| ATL| 4.675675675675675|

| 4| JFK| 0.07142857142857142|

| 10| SNA| -1.1333333333333333|

+-----+----+--------------------+

only showing top 20 rows

+-----+----+----------------------+

|month|dest|stddev\_samp(dep\_delay)|

+-----+----+----------------------+

| 11| TUS| 3.0550504633038935|

| 11| ANC| 18.604716401245316|

| 1| BUR| 15.22627576540667|

| 1| PDX| 5.677214918493858|

| 6| SBA| 2.380476142847617|

| 5| LAX| 13.36268698685904|

| 10| DTW| 5.639148871948674|

| 6| SIT| NaN|

| 10| DFW| 45.53019017606675|

| 3| FAI| 3.1144823004794873|

| 10| SEA| 18.70523227029577|

| 2| TUS| 14.468356276140469|

| 12| OGG| 82.64480404939947|

| 9| DFW| 21.728629347782924|

| 5| EWR| 42.41595968929191|

| 3| RDM| 2.16794833886788|

| 8| DCA| 9.946523680831074|

| 7| ATL| 22.767001039582183|

| 4| JFK| 8.156774303176903|

| 10| SNA| 13.726234873756304|

+-----+----+----------------------+

only showing top 20 rows

In [3]:

+100 XP

Amazing! You're learning so much from just a few simple methods!

**Joining**

Another very common data operation is the *join*. Joins are a whole topic unto themselves, so in this course we'll just look at simple joins. If you'd like to learn more about joins, you can take a look [**here**](https://www.datacamp.com/courses/merging-dataframes-with-pandas).

A join will combine two different tables along a column that they share. This column is called the *key*. Examples of keys here include the tailnum and carrier columns from the flights table.

For example, suppose that you want to know more information about the plane that flew a flight than just the tail number. This information isn't in the flights table because the same plane flies many different flights over the course of two years, so including this information in every row would result in a lot of duplication. To avoid this, you'd have a second table that has only one row for each plane and whose columns list all the information about the plane, including its tail number. You could call this table planes

When you join the flights table to this table of airplane information, you're adding all the columns from the planes table to the flights table. To fill these columns with information, you'll look at the tail number from the flights table and find the matching one in the planes table, and then use that row to fill out all the new columns.

Now you'll have a much bigger table than before, but now every row has all information about the plane that flew that flight!

Which of the following is **not** true?

Answer the question

50 XP

Possible Answers

Joins combine tables.

Joins add information to a table.

Storing information in separate tables can reduce repetition.

There is only one kind of join.

**Incorrect Submission**

Joins combine two tables, so the output should have more information in it than the input.

A join adds information from one table to another table.

Putting all your data in a single table can mean having to repeat a lot of values. Think about how planes make many flights.

+50 XP

Great job! If there were only one kind of join, it would be tough to create some more complicated kinds of tables.

**Exercise**

**Exercise**

**Joining II**

In PySpark, joins are performed using the DataFrame method .join(). This method takes three arguments. The first is the second DataFrame that you want to join with the first one. The second argument, on, is the name of the key column(s) as a string. The names of the key column(s) must be the same in each table. The third argument, how, specifies the kind of join to perform. In this course we'll always use the value how="leftouter".

The flights dataset and a new dataset called airports are already in your workspace.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Examine the airports DataFrame by calling .show(). Note which key column will let you join airports to the flights table.
* Rename the faa column in airports to dest by re-assigning the result of airports.withColumnRenamed("faa", "dest") to airports.
* Join the flights with the airports DataFrame on the dest column by calling the .join() method on flights. Save the result as flights\_with\_airports.
  + The first argument should be the other DataFrame, airports.
  + The argument on should be the key column.
  + The argument how should be "leftouter".
* Call .show() on flights\_with\_airports to examine the data again. Note the new information that has been added.

[**Take Hint (-30 XP)**](javascript:void(0))

# Examine the data

print(airports.show())

# Rename the faa column

airports = airports.withColumnRenamed("faa", "dest")

# Join the DataFrames

flights\_with\_airports = flights.join(airports, on='key', how='leftouter')

# Examine the new DataFrame

print(flights\_with\_airports.show())

# Examine the data

print(airports.show())

# Rename the faa column

airports = airports.withColumnRenamed("faa", "dest")

# Join the DataFrames

flights\_with\_airports = flights.join(airports, on='dest', how='leftouter')

# Examine the new DataFrame

print(flights\_with\_airports.show())

**Incorrect Submission**

Check your call of flights.join(). Did you correctly specify the argument on? Expected "dest", but got 'key'.

**Hint**

Make sure the key columns have the same name and that you've specified all three arguments to .join().

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

+---+--------------------+----------------+-----------------+----+---+---+

|faa| name| lat| lon| alt| tz|dst|

+---+--------------------+----------------+-----------------+----+---+---+

|04G| Lansdowne Airport| 41.1304722| -80.6195833|1044| -5| A|

|06A|Moton Field Munic...| 32.4605722| -85.6800278| 264| -5| A|

|06C| Schaumburg Regional| 41.9893408| -88.1012428| 801| -6| A|

|06N| Randall Airport| 41.431912| -74.3915611| 523| -5| A|

|09J|Jekyll Island Air...| 31.0744722| -81.4277778| 11| -4| A|

|0A9|Elizabethton Muni...| 36.3712222| -82.1734167|1593| -4| A|

|0G6|Williams County A...| 41.4673056| -84.5067778| 730| -5| A|

|0G7|Finger Lakes Regi...| 42.8835647| -76.7812318| 492| -5| A|

|0P2|Shoestring Aviati...| 39.7948244| -76.6471914|1000| -5| U|

|0S9|Jefferson County ...| 48.0538086| -122.8106436| 108| -8| A|

|0W3|Harford County Ai...| 39.5668378| -76.2024028| 409| -5| A|

|10C| Galt Field Airport| 42.4028889| -88.3751111| 875| -6| U|

|17G|Port Bucyrus-Craw...| 40.7815556| -82.9748056|1003| -5| A|

|19A|Jackson County Ai...| 34.1758638| -83.5615972| 951| -4| U|

|1A3|Martin Campbell F...| 35.0158056| -84.3468333|1789| -4| A|

|1B9| Mansfield Municipal| 42.0001331| -71.1967714| 122| -5| A|

|1C9|Frazier Lake Airpark|54.0133333333333|-124.768333333333| 152| -8| A|

|1CS|Clow Internationa...| 41.6959744| -88.1292306| 670| -6| U|

|1G3| Kent State Airport| 41.1513889| -81.4151111|1134| -4| A|

|1OH| Fortman Airport| 40.5553253| -84.3866186| 885| -5| U|

+---+--------------------+----------------+-----------------+----+---+---+

only showing top 20 rows

None

Traceback (most recent call last):

File "script.py", line 8, in <module>

flights\_with\_airports = flights.join(airports, on='key', how='leftouter')

File "script.py", line 931, in join

jdf = self.\_jdf.join(other.\_jdf, on, how)

File "script.py", line 1257, in \_\_call\_\_

answer, self.gateway\_client, self.target\_id, self.name)

File "script.py", line 69, in deco

raise AnalysisException(s.split(': ', 1)[1], stackTrace)

pyspark.sql.utils.AnalysisException: 'USING column `key` cannot be resolved on the left side of the join. The left-side columns: [year, month, day, dep\_time, dep\_delay, arr\_time, arr\_delay, carrier, tailnum, flight, origin, dest, air\_time, distance, hour, minute];'

<script.py> output:

+----+--------------------+----------------+-----------------+----+---+---+

|dest| name| lat| lon| alt| tz|dst|

+----+--------------------+----------------+-----------------+----+---+---+

| 04G| Lansdowne Airport| 41.1304722| -80.6195833|1044| -5| A|

| 06A|Moton Field Munic...| 32.4605722| -85.6800278| 264| -5| A|

| 06C| Schaumburg Regional| 41.9893408| -88.1012428| 801| -6| A|

| 06N| Randall Airport| 41.431912| -74.3915611| 523| -5| A|

| 09J|Jekyll Island Air...| 31.0744722| -81.4277778| 11| -4| A|

| 0A9|Elizabethton Muni...| 36.3712222| -82.1734167|1593| -4| A|

| 0G6|Williams County A...| 41.4673056| -84.5067778| 730| -5| A|

| 0G7|Finger Lakes Regi...| 42.8835647| -76.7812318| 492| -5| A|

| 0P2|Shoestring Aviati...| 39.7948244| -76.6471914|1000| -5| U|

| 0S9|Jefferson County ...| 48.0538086| -122.8106436| 108| -8| A|

| 0W3|Harford County Ai...| 39.5668378| -76.2024028| 409| -5| A|

| 10C| Galt Field Airport| 42.4028889| -88.3751111| 875| -6| U|

| 17G|Port Bucyrus-Craw...| 40.7815556| -82.9748056|1003| -5| A|

| 19A|Jackson County Ai...| 34.1758638| -83.5615972| 951| -4| U|

| 1A3|Martin Campbell F...| 35.0158056| -84.3468333|1789| -4| A|

| 1B9| Mansfield Municipal| 42.0001331| -71.1967714| 122| -5| A|

| 1C9|Frazier Lake Airpark|54.0133333333333|-124.768333333333| 152| -8| A|

| 1CS|Clow Internationa...| 41.6959744| -88.1292306| 670| -6| U|

| 1G3| Kent State Airport| 41.1513889| -81.4151111|1134| -4| A|

| 1OH| Fortman Airport| 40.5553253| -84.3866186| 885| -5| U|

+----+--------------------+----------------+-----------------+----+---+---+

only showing top 20 rows

None

+----+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+--------+--------+----+------+--------------------+---------+-----------+----+---+---+

|dest|year|month|day|dep\_time|dep\_delay|arr\_time|arr\_delay|carrier|tailnum|flight|origin|air\_time|distance|hour|minute| name| lat| lon| alt| tz|dst|

+----+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+--------+--------+----+------+--------------------+---------+-----------+----+---+---+

| LAX|2014| 12| 8| 658| -7| 935| -5| VX| N846VA| 1780| SEA| 132| 954| 6| 58| Los Angeles Intl|33.942536|-118.408075| 126| -8| A|

| HNL|2014| 1| 22| 1040| 5| 1505| 5| AS| N559AS| 851| SEA| 360| 2677| 10| 40| Honolulu Intl|21.318681|-157.922428| 13|-10| N|

| SFO|2014| 3| 9| 1443| -2| 1652| 2| VX| N847VA| 755| SEA| 111| 679| 14| 43| San Francisco Intl|37.618972|-122.374889| 13| -8| A|

| SJC|2014| 4| 9| 1705| 45| 1839| 34| WN| N360SW| 344| PDX| 83| 569| 17| 5|Norman Y Mineta S...| 37.3626|-121.929022| 62| -8| A|

| BUR|2014| 3| 9| 754| -1| 1015| 1| AS| N612AS| 522| SEA| 127| 937| 7| 54| Bob Hope|34.200667|-118.358667| 778| -8| A|

| DEN|2014| 1| 15| 1037| 7| 1352| 2| WN| N646SW| 48| PDX| 121| 991| 10| 37| Denver Intl|39.861656|-104.673178|5431| -7| A|

| OAK|2014| 7| 2| 847| 42| 1041| 51| WN| N422WN| 1520| PDX| 90| 543| 8| 47|Metropolitan Oakl...|37.721278|-122.220722| 9| -8| A|

| SFO|2014| 5| 12| 1655| -5| 1842| -18| VX| N361VA| 755| SEA| 98| 679| 16| 55| San Francisco Intl|37.618972|-122.374889| 13| -8| A|

| SAN|2014| 4| 19| 1236| -4| 1508| -7| AS| N309AS| 490| SEA| 135| 1050| 12| 36| San Diego Intl|32.733556|-117.189667| 17| -8| A|

| ORD|2014| 11| 19| 1812| -3| 2352| -4| AS| N564AS| 26| SEA| 198| 1721| 18| 12| Chicago Ohare Intl|41.978603| -87.904842| 668| -6| A|

| LAX|2014| 11| 8| 1653| -2| 1924| -1| AS| N323AS| 448| SEA| 130| 954| 16| 53| Los Angeles Intl|33.942536|-118.408075| 126| -8| A|

| PHX|2014| 8| 3| 1120| 0| 1415| 2| AS| N305AS| 656| SEA| 154| 1107| 11| 20|Phoenix Sky Harbo...|33.434278|-112.011583|1135| -7| N|

| LAS|2014| 10| 30| 811| 21| 1038| 29| AS| N433AS| 608| SEA| 127| 867| 8| 11| Mc Carran Intl|36.080056| -115.15225|2141| -8| A|

| ANC|2014| 11| 12| 2346| -4| 217| -28| AS| N765AS| 121| SEA| 183| 1448| 23| 46|Ted Stevens Ancho...|61.174361|-149.996361| 152| -9| A|

| SFO|2014| 10| 31| 1314| 89| 1544| 111| AS| N713AS| 306| SEA| 129| 679| 13| 14| San Francisco Intl|37.618972|-122.374889| 13| -8| A|

| SFO|2014| 1| 29| 2009| 3| 2159| 9| UA| N27205| 1458| PDX| 90| 550| 20| 9| San Francisco Intl|37.618972|-122.374889| 13| -8| A|

| SMF|2014| 12| 17| 2015| 50| 2150| 41| AS| N626AS| 368| SEA| 76| 605| 20| 15| Sacramento Intl|38.695417|-121.590778| 27| -8| A|

| MDW|2014| 8| 11| 1017| -3| 1613| -7| WN| N8634A| 827| SEA| 216| 1733| 10| 17| Chicago Midway Intl|41.785972| -87.752417| 620| -6| A|

| BOS|2014| 1| 13| 2156| -9| 607| -15| AS| N597AS| 24| SEA| 290| 2496| 21| 56|General Edward La...|42.364347| -71.005181| 19| -5| A|

| BUR|2014| 6| 5| 1733| -12| 1945| -10| OO| N215AG| 3488| PDX| 111| 817| 17| 33| Bob Hope|34.200667|-118.358667| 778| -8| A|

+----+----+-----+---+--------+---------+--------+---------+-------+-------+------+------+--------+--------+----+------+--------------------+---------+-----------+----+---+---+

only showing top 20 rows

None

In [1]:

+70 XP

Fantastic work! You're a data manipulation pro!

**Machine Learning Pipelines**

In the next two chapters you'll step through every stage of the machine learning pipeline, from data intake to model evaluation. Let's get to it!

At the core of the pyspark.ml module are the Transformer and Estimator classes. Almost every other class in the module behaves similarly to these two basic classes.

Transformer classes have a .transform() method that takes a DataFrame and returns a new DataFrame; usually the original one with a new column appended. For example, you might use the class Bucketizer to create discrete bins from a continuous feature or the class PCA to reduce the dimensionality of your dataset using principal component analysis.

Estimator classes all implement a .fit() method. These methods also take a DataFrame, but instead of returning another DataFrame they return a model object. This can be something like a StringIndexerModel for including categorical data saved as strings in your models, or a RandomForestModel that uses the random forest algorithm for classification or regression.

Which of the following is **not** true about machine learning in Spark?

**Answer the question**

**50 XP**

Possible Answers

Spark's algorithms give better results than other algorithms.

press

1

Working in Spark allows you to create reproducible machine learning pipelines.

press

2

Machine learning pipelines in Spark are made up of Transformers and Estimators.

press

3

PySpark uses the pyspark.ml submodule to interface with Spark's machine learning routines.

press

4

+50 XP

That's right! Spark is just a platform that implements the same algorithms that can be found elsewhere.

**Exercise**

**Exercise**

**Join the DataFrames**

In the next two chapters you'll be working to build a model that predicts whether or not a flight will be delayed based on the flights data we've been working with. This model will also include information about the plane that flew the route, so the first step is to join the two tables: flights and planes!

**Instructions**

**100 XP**

* First, rename the year column of planes to plane\_year to avoid duplicate column names.
* Create a new DataFrame called model\_data by joining the flights table with planes using the tailnum column as the key.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Check your call of flights.join(). Did you correctly specify the first argument? Expected planes, but got \_\_\_\_.

Check your call of flights.join(). Did you correctly specify the first argument? Expected planes, but got 'planes'.

# Rename year column

planes = planes.withColumnRenamed('year', 'plane\_year')

# Join the DataFrames

model\_data = flights.join(planes, on='tailnum', how="leftouter")

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: help(planes.withColumnRenamed)

Help on method withColumnRenamed in module pyspark.sql.dataframe:

withColumnRenamed(existing, new) method of pyspark.sql.dataframe.DataFrame instance

Returns a new :class:`DataFrame` by renaming an existing column.

This is a no-op if schema doesn't contain the given column name.

:param existing: string, name of the existing column to rename.

:param col: string, new name of the column.

>>> df.withColumnRenamed('age', 'age2').collect()

[Row(age2=2, name='Alice'), Row(age2=5, name='Bob')]

.. versionadded:: 1.3

Traceback (most recent call last):

File "script.py", line 5, in <module>

model\_data = flights.join(\_\_\_\_, on=\_\_\_\_, how="leftouter")

NameError: name '\_\_\_\_' is not defined

Traceback (most recent call last):

File "script.py", line 5, in <module>

model\_data = flights.join('planes', on=tailnum, how="leftouter")

NameError: name 'tailnum' is not defined

Traceback (most recent call last):

File "script.py", line 5, in <module>

model\_data = flights.join('planes', on='tailnum', how="leftouter")

File "script.py", line 931, in join

jdf = self.\_jdf.join(other.\_jdf, on, how)

AttributeError: 'str' object has no attribute '\_jdf'

In [2]:

+100 XP

Awesome work! You're one step closer to a model!

**Data types**

Good work! Before you get started modeling, it's important to know that Spark only handles numeric data. That means all of the columns in your DataFrame must be either integers or decimals (called 'doubles' in Spark).

When we imported our data, we let Spark guess what kind of information each column held. Unfortunately, Spark doesn't always guess right and you can see that some of the columns in our DataFrame are strings containing numbers as opposed to actual numeric values.

To remedy this, you can use the .cast() method in combination with the .withColumn() method. It's important to note that .cast() works on columns, while .withColumn() works on DataFrames.

The only argument you need to pass to .cast() is the kind of value you want to create, in string form. For example, to create integers, you'll pass the argument "integer" and for decimal numbers you'll use "double".

You can put this call to .cast() inside a call to .withColumn() to overwrite the already existing column, just like you did in the previous chapter!

What kind of data does Spark need for modeling?

**Answer the question**

**50 XP**

**Possible Answers**

Doubles

press

1

Integers

press

2

Decimals

press

3

Numeric

press

4

Strings

press

5

+50 XP

Great job! Spark needs numeric values (doubles or integers) to do machine learning.

**Exercise**

**Exercise**

**String to integer**

Now you'll use the .cast() method you learned in the previous exercise to convert all the appropriate columns from your DataFrame model\_data to integers!

To convert the type of a column using the .cast() method, you can write code like this:

dataframe = dataframe.withColumn("col", dataframe.col.cast("new\_type"))

**Instructions**

**100 XP**

* Use the method .withColumn() to .cast() the following columns to type "integer". Access the columns using the df.col notation:
  + model\_data.arr\_delay
  + model\_data.air\_time
  + model\_data.month
  + model\_data.plane\_year

[**Take Hint (-30 XP)**](javascript:void(0))

# Cast the columns to integers

model\_data = model\_data.withColumn("arr\_delay", model\_data.arr\_delay.cast("integer"))

model\_data = model\_data.withColumn("air\_time", model\_data.air\_time.cast("integer"))

model\_data = model\_data.withColumn("month", model\_data.month.cast("integer"))

model\_data = model\_data.withColumn("plane\_year", model\_data.plane\_year.cast("integer"))

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]:

+100 XP

Awesome! You're a pro at converting columns!

**Exercise**

**Exercise**

**Create a new column**

In the last exercise, you converted the column plane\_year to an integer. This column holds the year each plane was manufactured. However, your model will use the planes' *age*, which is slightly different from the year it was made!

**Instructions**

**100 XP**

* Create the column plane\_age using the .withColumn() method and subtracting the year of manufacture (column plane\_year) from the year (column year) of the flight.

[**Take Hint (-30 XP)**](javascript:void(0))

# Create the column plane\_age

model\_data = model\_data.withColumn("plane\_age", model\_data.year-model\_data.plane\_year)

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: # Create the column plane\_age

model\_data = model\_data.withColumn("plane\_age", model\_data.year-model\_data.plane\_year)

In [2]:

+100 XP

Great work! Now you have one more variable to include in your model.

**Exercise**

**Exercise**

**Making a Boolean**

Consider that you're modeling a yes or no question: is the flight late? However, your data contains the arrival delay in minutes for each flight. Thus, you'll need to create a boolean column which indicates whether the flight was late or not!

**Instructions**

**100 XP**

* Use the .withColumn() method to create the column is\_late. This column is equal to model\_data.arr\_delay > 0.
* Convert this column to an integer column so that you can use it in your model and name it label (this is the default name for the response variable in Spark's machine learning routines).
* Filter out missing values (this has been done for you).

Ctrl+H

[**Take Hint (-30 XP)**](javascript:void(0))

# Create is\_late

model\_data = model\_data.withColumn("is\_late", model\_data.arr\_delay > 0)

# Convert to an integer

model\_data = model\_data.withColumn("label", model\_data.is\_late.cast("integer"))

# Remove missing values

model\_data = model\_data.filter("arr\_delay is not NULL and dep\_delay is not NULL and air\_time is not NULL and plane\_year is not NULL")

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]:

+100 XP

Awesome! Now you've defined the column that you're going to use as the outcome in your model.

**Strings and factors**

As you know, Spark requires numeric data for modeling. So far this hasn't been an issue; even boolean columns can easily be converted to integers without any trouble. But you'll also be using the airline and the plane's destination as features in your model. These are coded as strings and there isn't any obvious way to convert them to a numeric data type.

Fortunately, PySpark has functions for handling this built into the pyspark.ml.features submodule. You can create what are called 'one-hot vectors' to represent the carrier and the destination of each flight. A *one-hot vector* is a way of representing a categorical feature where every observation has a vector in which all elements are zero except for at most one element, which has a value of one (1).

Each element in the vector corresponds to a level of the feature, so it's possible to tell what the right level is by seeing which element of the vector is equal to one (1).

The first step to encoding your categorical feature is to create a StringIndexer. Members of this class are Estimators that take a DataFrame with a column of strings and map each unique string to a number. Then, the Estimator returns a Transformer that takes a DataFrame, attaches the mapping to it as metadata, and returns a new DataFrame with a numeric column corresponding to the string column.

The second step is to encode this numeric column as a one-hot vector using a OneHotEncoder. This works exactly the same way as the StringIndexer by creating an Estimator and then a Transformer. The end result is a column that encodes your categorical feature as a vector that's suitable for machine learning routines!

This may seem complicated, but don't worry! All you have to remember is that you need to create a StringIndexer and a OneHotEncoder, and the Pipeline will take care of the rest.

Why do you have to encode a categorical feature as a one-hot vector?

**Answer the question**

**50 XP**

**Possible Answers**

It makes fitting the model faster.

press

1

Spark can only model numeric features.

press

2

For compatibility with scikit-learn.

press

3

+50 XP

Awesome! You remembered that Spark can only model numeric features.

**Exercise**

**Exercise**

**Carrier**

In this exercise you'll create a StringIndexer and a OneHotEncoder to code the carrier column. To do this, you'll call the class constructors with the arguments inputCol and outputCol.

The inputCol is the name of the column you want to index or encode, and the outputCol is the name of the new column that the Transformer should create.

**Instructions**

**100 XP**

* Create a StringIndexer called carr\_indexer by calling StringIndexer() with inputCol="carrier" and outputCol="carrier\_index".
* Create a OneHotEncoder called carr\_encoder by calling OneHotEncoder() with inputCol="carrier\_index" and outputCol="carrier\_fact".

[**Take Hint (-30 XP)**](javascript:void(0))

# Create a StringIndexer

carr\_indexer = StringIndexer(inputCol="carrier", outputCol="carrier\_index")

# Create a OneHotEncoder

carr\_encoder = OneHotEncoder(inputCol="carrier\_index", outputCol="carrier\_fact")

+100 XP

Fantastic work! You're ready to include this column in your model now!

**Exercise**

**Exercise**

**Destination**

Now you'll encode the dest column just like you did in the previous exercise.

**Instructions**

**100 XP**

* Create a StringIndexer called dest\_indexer by calling StringIndexer() with inputCol="dest" and outputCol="dest\_index".
* Create a OneHotEncoder called dest\_encoder by calling OneHotEncoder() with inputCol="dest\_index" and outputCol="dest\_fact".

[**Take Hint (-30 XP)**](javascript:void(0))

# Create a StringIndexer

dest\_indexer = StringIndexer(inputCol="dest", outputCol="dest\_index")

# Create a OneHotEncoder

dest\_encoder = OneHotEncoder(inputCol="dest\_index", outputCol="dest\_fact")

+100 XP

Perfect! You're all done messing with factors.

**Exercise**

**Exercise**

**Assemble a vector**

The last step in the Pipeline is to combine all of the columns containing our features into a single column. This has to be done before modeling can take place because every Spark modeling routine expects the data to be in this form. You can do this by storing each of the values from a column as an entry in a vector. Then, from the model's point of view, every observation is a vector that contains all of the information about it and a label that tells the modeler what value that observation corresponds to.

Because of this, the pyspark.ml.feature submodule contains a class called VectorAssembler. This Transformer takes all of the columns you specify and combines them into a new vector column.

**Instructions**

**100 XP**

* Create a VectorAssembler by calling VectorAssembler() with the inputCols names as a list and the outputCol name "features".
  + The list of columns should be ["month", "air\_time", "carrier\_fact", "dest\_fact", "plane\_age"].

[**Take Hint (-30 XP)**](javascript:void(0))

# Make a VectorAssembler

vec\_assembler = VectorAssembler(inputCols=["month", "air\_time", "carrier\_fact", "dest\_fact", "plane\_age"], outputCol="features")

+100 XP

Good job! Your data is all assembled now.

**Exercise**

**Exercise**

**Create the pipeline**

You're finally ready to create a Pipeline!

Pipeline is a class in the pyspark.ml module that combines all the Estimators and Transformers that you've already created. This lets you reuse the same modeling process over and over again by wrapping it up in one simple object. Neat, right?

**Instructions**

**100 XP**

* Import Pipeline from pyspark.ml.
* Call the Pipeline() constructor with the keyword argument stages to create a Pipeline called flights\_pipe.
  + stages should be a list holding all the stages you want your data to go through in the pipeline. Here this is just: [dest\_indexer, dest\_encoder, carr\_indexer, carr\_encoder, vec\_assembler]

[**Take Hint (-30 XP)**](javascript:void(0))

# Import Pipeline

from pyspark.ml import Pipeline

# Make the pipeline

flights\_pipe = Pipeline(stages=[dest\_indexer, dest\_encoder, carr\_indexer, carr\_encoder, vec\_assembler])

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: dest\_indexer

Out[1]: StringIndexer\_4303bac7b9a76e45f273

In [2]:

+100 XP

Fantastic! You've made a fully reproducible machine learning pipeline!

**Test vs Train**

After you've cleaned your data and gotten it ready for modeling, one of the most important steps is to split the data into a *test set* and a *train set*. After that, don't touch your test data until you think you have a good model! As you're building models and forming hypotheses, you can test them on your training data to get an idea of their performance.

Once you've got your favorite model, you can see how well it predicts the new data in your test set. This never-before-seen data will give you a much more realistic idea of your model's performance in the real world when you're trying to predict or classify new data.

In Spark it's important to make sure you split the data **after** all the transformations. This is because operations like StringIndexer don't always produce the same index even when given the same list of strings.

Why is it important to use a test set in model evaluation?

**Answer the question**

**50 XP**

**Possible Answers**

Evaluating your model improves its accuracy.

press

1

By evaluating your model with a test set you can get a good idea of performance on new data.

press

2

Using a test set lets you check your code for errors.

press

3

+50 XP

Exactly! A test set approximates the 'real world error' of your model.

**Exercise**

**Exercise**

**Transform the data**

Hooray, now you're finally ready to pass your data through the Pipeline you created!

**Instructions**

**100 XP**

* Create the DataFrame piped\_data by calling the Pipeline methods .fit() and .transform() in a chain. Both of these methods take model\_data as their only argument.

[**Take Hint (-30 XP)**](javascript:void(0))

# Fit and transform the data

piped\_data = flights\_pipe.fit(model\_data).transform(model\_data)

+100 XP

Great work! Your pipeline chewed right through that data!

**Exercise**

**Exercise**

**Split the data**

Now that you've done all your manipulations, the last step before modeling is to split the data!

**Instructions**

**100 XP**

* Use the DataFrame method .randomSplit() to split piped\_data into two pieces, training with 60% of the data, and test with 40% of the data by passing the list [.6, .4] to the .randomSplit() method.

[**Take Hint (-30 XP)**](javascript:void(0))

# Split the data into training and test sets

training, test = piped\_data.randomSplit([.6, .4])

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: help(piped\_data.randomSplit)

Help on method randomSplit in module pyspark.sql.dataframe:

randomSplit(weights, seed=None) method of pyspark.sql.dataframe.DataFrame instance

Randomly splits this :class:`DataFrame` with the provided weights.

:param weights: list of doubles as weights with which to split the DataFrame. Weights will

be normalized if they don't sum up to 1.0.

:param seed: The seed for sampling.

>>> splits = df4.randomSplit([1.0, 2.0], 24)

>>> splits[0].count()

1

>>> splits[1].count()

3

.. versionadded:: 1.4

In [2]:

+100 XP

Awesome! Now you're ready to start fitting a model!

**What is logistic regression?**

The model you'll be fitting in this chapter is called a *logistic regression*. This model is very similar to a linear regression, but instead of predicting a numeric variable, it predicts the probability (between 0 and 1) of an event.

To use this as a classification algorithm, all you have to do is assign a cutoff point to these probabilities. If the predicted probability is above the cutoff point, you classify that observation as a 'yes' (in this case, the flight being late), if it's below, you classify it as a 'no'!

You'll tune this model by testing different values for several *hyperparameters*. A *hyperparameter* is just a value in the model that's not estimated from the data, but rather is supplied by the user to maximize performance. For this course it's not necessary to understand the mathematics behind all of these values - what's important is that you'll try out a few different choices and pick the best one.

Why do you supply hyperparameters?

**Answer the question**

**50 XP**

**Possible Answers**

They explain information about the data.

press1

They improve model performance.

press2

They improve model fitting speed.

press3

+50 XP

Great job! You supply hyperparameters to optimize your model.

**Exercise**

**Exercise**

**Create the modeler**

The Estimator you'll be using is a LogisticRegression from the pyspark.ml.classification submodule.

**Instructions**

**100 XP**

* Import the LogisticRegression class from pyspark.ml.classification.
* Create a LogisticRegression called lr by calling LogisticRegression() with no arguments.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import LogisticRegression

from pyspark.ml.classification import LogisticRegression

# Create a LogisticRegression Estimator

lr = LogisticRegression()

+100 XP

Great work! That's the first step to any modeling in PySpark.

**Cross validation**

In the next few exercises you'll be tuning your logistic regression model using a procedure called *k-fold cross validation*. This is a method of estimating the model's performance on unseen data (like your test DataFrame).

It works by splitting the training data into a few different partitions. The exact number is up to you, but in this course you'll be using PySpark's default value of three. Once the data is split up, one of the partitions is set aside, and the model is fit to the others. Then the error is measured against the held out partition. This is repeated for each of the partitions, so that every block of data is held out and used as a test set exactly once. Then the error on each of the partitions is averaged. This is called the *cross validation error* of the model, and is a good estimate of the actual error on the held out data.

You'll be using cross validation to choose the hyperparameters by creating a grid of the possible pairs of values for the two hyperparameters, elasticNetParam and regParam, and using the cross validation error to compare all the different models so you can choose the best one!

What does cross validation allow you to estimate?

**Answer the question**

**50 XP**

**Possible Answers**

The model's error on held out data.

press

1

The model's error on data used for fitting.

press

2

The time it will take to fit the model.

press

3

+50 XP

Exactly! The cross validation error is an estimate of the model's error on the test set.

**Exercise**

**Exercise**

**Create the evaluator**

The first thing you need when doing cross validation for model selection is a way to compare different models. Luckily, the pyspark.ml.evaluation submodule has classes for evaluating different kinds of models. Your model is a binary classification model, so you'll be using the BinaryClassificationEvaluator from the pyspark.ml.evaluation module.

This evaluator calculates the area under the ROC. This is a metric that combines the two kinds of errors a binary classifier can make (false positives and false negatives) into a simple number. You'll learn more about this towards the end of the chapter!

**Instructions**

**100 XP**

* Import the submodule pyspark.ml.evaluation as evals.
* Create evaluator by calling evals.BinaryClassificationEvaluator() with the argument metricName="areaUnderROC".

[**Take Hint (-30 XP)**](javascript:void(0))

# Import the evaluation submodule

import pyspark.ml.evaluation as evals

# Create a BinaryClassificationEvaluator

evaluator = evals.BinaryClassificationEvaluator(metricName="areaUnderROC")

+100 XP

Perfect! Now you can compare models using the metric output by your evaluator!

**Exercise**

**Exercise**

**Make a grid**

Next, you need to create a grid of values to search over when looking for the optimal hyperparameters. The submodule pyspark.ml.tuning includes a class called ParamGridBuilder that does just that (maybe you're starting to notice a pattern here; PySpark has a submodule for just about everything!).

You'll need to use the .addGrid() and .build() methods to create a grid that you can use for cross validation. The .addGrid() method takes a model parameter (an attribute of the model Estimator, lr, that you created a few exercises ago) and a list of values that you want to try. The .build() method takes no arguments, it just returns the grid that you'll use later.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Import the submodule pyspark.ml.tuning under the alias tune.
* Call the class constructor ParamGridBuilder() with no arguments. Save this as grid.
* Call the .addGrid() method on grid with lr.regParam as the first argument and np.arange(0, .1, .01) as the second argument. This second call is a function from the numpy module (imported as np) that creates a list of numbers from 0 to .1, incrementing by .01. Overwrite grid with the result.
* Update grid again by calling the .addGrid() method a second time create a grid for lr.elasticNetParam that includes only the values [0, 1].
* Call the .build() method on grid and overwrite it with the output.

[**Take Hint (-30 XP)**](javascript:void(0))

# Import the tuning submodule

import pyspark.ml.tuning as tune

# Create the parameter grid

grid = tune.ParamGridBuilder()

# Add the hyperparameter

grid = grid.addGrid(lr.regParam, np.arange(0, .1, .01))

grid = grid.addGrid(lr.elasticNetParam, [0, 1])

# Build the grid

grid = grid.build()

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: lr.regParam

Out[1]: Param(parent='LogisticRegression\_459dbeba348c9af3978b', name='regParam', doc='regularization parameter (>= 0).')

In [2]: np

Out[2]: <module 'numpy' from '/usr/local/lib/python3.5/dist-packages/numpy/\_\_init\_\_.py'>

In [3]:

+100 XP

Awesome! That's the last ingredient in your cross validation recipe!

**Exercise**

**Exercise**

**Make the validator**

The submodule pyspark.ml.tuning also has a class called CrossValidator for performing cross validation. This Estimator takes the modeler you want to fit, the grid of hyperparameters you created, and the evaluator you want to use to compare your models.

The submodule pyspark.ml.tune has already been imported as tune. You'll create the CrossValidator by passing it the logistic regression Estimator lr, the parameter grid, and the evaluator you created in the previous exercises.

**Instructions**

**100 XP**

* Create a CrossValidator by calling tune.CrossValidator() with the arguments:
  + estimator=lr
  + estimatorParamMaps=grid
  + evaluator=evaluator
* Name this object cv.

[**Take Hint (-30 XP)**](javascript:void(0))

# Create the CrossValidator

cv = tune.CrossValidator(

estimator=lr,

estimatorParamMaps=grid,

evaluator=evaluator

)

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

In [1]: lr, grid, evaluator

Out[1]:

(LogisticRegression\_4658a639324f403b8c97,

[{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.0,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.0,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.01,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.01,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.02,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.02,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.03,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.03,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.04,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.04,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.05,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.05,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.06,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.06,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.07,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.07,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.08,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.08,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.09,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 0},

{Param(parent='LogisticRegression\_4658a639324f403b8c97', name='regParam', doc='regularization parameter (>= 0).'): 0.09,

Param(parent='LogisticRegression\_4658a639324f403b8c97', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'): 1}],

BinaryClassificationEvaluator\_419ba2d94025968f415b)

In [2]:

+100 XP

Great job! You're almost a machine learning pro!

**Exercise**

**Exercise**

**Fit the model(s)**

You're finally ready to fit the models and select the best one!

Unfortunately, cross validation is a very computationally intensive procedure. Fitting all the models would take too long on DataCamp.

To do this locally you would use the code:

# Fit cross validation models

models = cv.fit(training)

# Extract the best model

best\_lr = models.bestModel

Remember, the training data is called training and you're using lr to fit a logistic regression model. Cross validation selected the parameter values regParam=0 and elasticNetParam=0 as being the best. These are the default values, so you don't need to do anything else with lr before fitting the model.

**Instructions**

**100 XP**

* Create best\_lr by calling lr.fit() on the training data.
* Print best\_lr to verify that it's an object of the LogisticRegressionModel class.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Have you specified the arguments for lr.fit() using the right syntax?

Your code generated an error. Fix it and try again!

import numpy as np

# Call lr.fit()

best\_lr = lr.fit(training)

# Print best\_lr

print(best\_lr)

print(np.array(best\_lr))

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

Traceback (most recent call last):

File "script.py", line 2, in <module>

best\_lr = lr.fit()

TypeError: fit() missing 1 required positional argument: 'dataset'

<script.py> output:

LogisticRegression\_4cb9bd908b8f4b14181d

Traceback (most recent call last):

File "script.py", line 6, in <module>

print(np.array(best\_lr))

NameError: name 'np' is not defined

<script.py> output:

LogisticRegression\_4cb9bd908b8f4b14181d

LogisticRegression\_4cb9bd908b8f4b14181d

In [1]:

+100 XP

Wow! You fit your first Spark model!

**Evaluating binary classifiers**

For this course we'll be using a common metric for binary classification algorithms call the *AUC*, or area under the curve. In this case, the curve is the ROC, or receiver operating curve. The details of what these things actually measure isn't important for this course. All you need to know is that for our purposes, the closer the AUC is to one (1), the better the model is!

If you've created a perfect binary classification model, what would the AUC be?

**Answer the question**

**50 XP**

**Possible Answers**

-1

press

1

1

press

2

0

press

3

.5

press

4

+50 XP

Great job! An AUC of one represents a model that always perfectly classifies observations.

**Exercise**

**Exercise**

**Evaluate the model**

Remember the test data that you set aside waaaaaay back in chapter 3? It's finally time to test your model on it! You can use the same evaluator you made to fit the model.

**Instructions**

**100 XP**

* Use your model to generate predictions by applying best\_lr.transform() to the test data. Save this as test\_results.
* Call evaluator.evaluate() on test\_results to compute the AUC. Print the output.

[**Take Hint (-30 XP)**](javascript:void(0))

# Use the model to predict the test set

test\_results = best\_lr.transform(test)

# Evaluate the predictions

print(evaluator.evaluate(test\_results))

<https://campus.datacamp.com/courses/introduction-to-pyspark/model-tuning-and-selection?ex=9>

+100 XP

Congratulations! What do you think of the AUC? Your model isn't half bad! You went from knowing nothing about Spark to doing advanced machine learning. Great job on making it to the end of the course! The next steps are learning how to create large scale Spark clusters and manage and submit jobs so that you can use models in the real world. Check out some of the other DataCamp courses that use Spark! And remember, Spark is still being actively developed, so there's new features coming all the time!

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.3.1

/\_/

Using Python version 3.5.2 (default, Nov 23 2017 16:37:01)

SparkSession available as 'spark'.

<script.py> output:

0.7125950520013029

In [1]:

<script.py> output:

0.7125950520013008

In [1]:

<script.py> output:

0.7125950520013022

In [1]: