What is Spark?

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?

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 *slaves*. The master sends the slaves 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.

Examining The SparkContext

```
# Verify SparkContext
print(sc)

# Print Spark version
print(sc.version)
```

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.

Creating a SparkSession

```
# Import SparkSession from pyspark.sql
from pyspark.sql import SparkSession

# Create my_spark
my_spark = SparkSession.builder.getOrCreate()

#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!

# Print my_spark
print(my_spark)

# Print the tables in the catalog
print(spark.catalog.listTables())
```

```
query1 = "FROM flights SELECT * LIMIT 10"

# Get the first 10 rows of flights
flights10 = spark.sql(query1)

# Show the results
flights10.show()

query2= "SELECT origin, dest, COUNT(*) as N FROM flights
GROUP BY origin, dest"

# Run the query
flight_counts = spark.sql(query2)

# Convert the results to a pandas DataFrame
pd_counts = flight_counts.toPandas()

# Print the head of pd_counts
print(pd_counts)
```

Put some Spark from Pandas

```
# 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())
```

Use Spark to read csv file directly as Spark DataFrame

```
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()
```