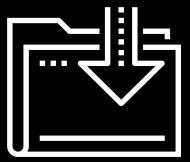
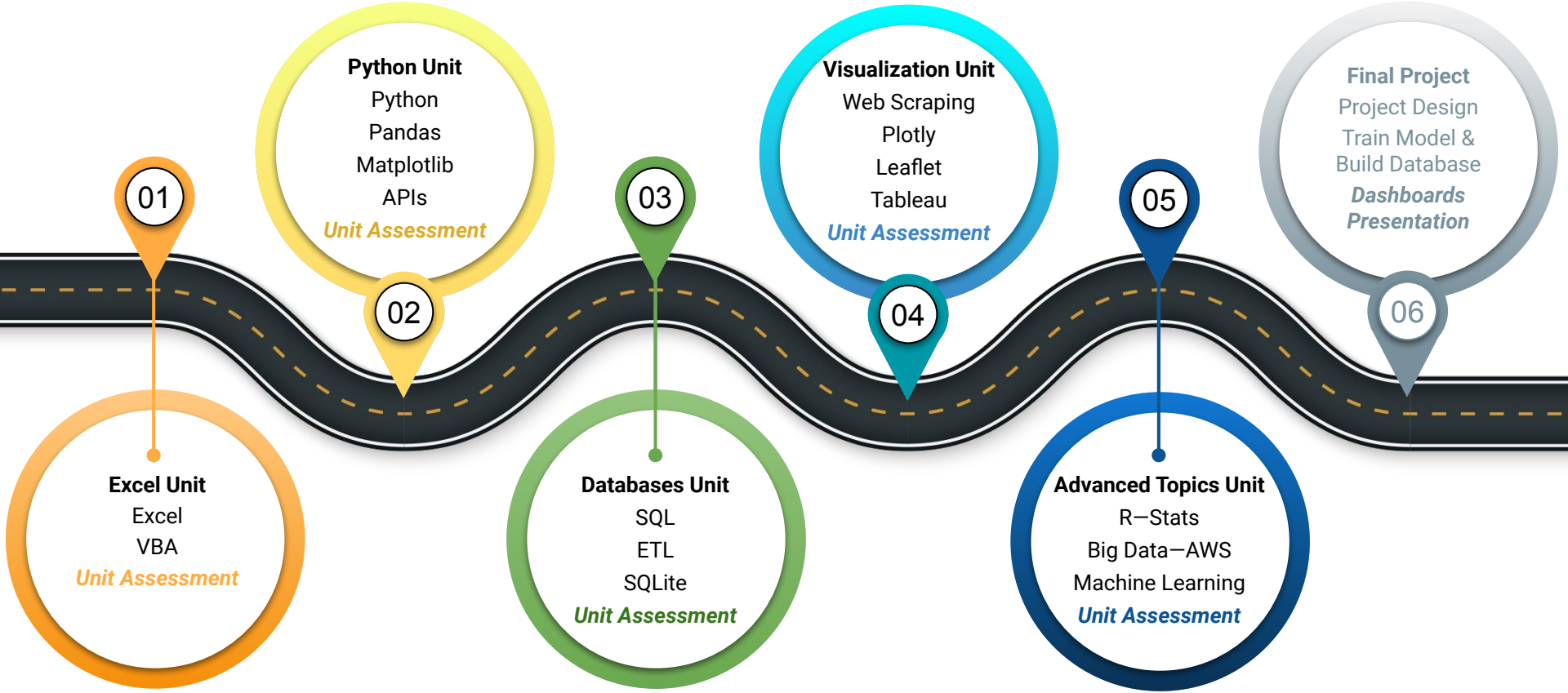
PySpark Fundamentals

Data Boot Camp

Lesson 16.1



The Big Picture



Quick Tip for Success:

Welcome the opportunity for a challenge, and embrace each learning opportunity as we near closer to the end of class!

Module 16

This Week: Big Data

This Week: Big Data

By the end of this week, you'll know how to:

- Define big data and describe the challenges associated with it
- Define Hadoop and name the main elements of its ecosystem
- Explain how MapReduce processes data
- Define Spark and explain how it processes data
- Describe how NLP collects and analyzes text data
- Use AWS Simple Storage Service (S3) and relational databases for basic cloud storage

This Week's Challenge

Using the skills learned throughout the week, connect to an AWS RDS instance and perform ETL on an Amazon customer-review dataset to determine if Vine reviews show bias towards being more favorable. .

Career Connection

How will you use this module's content in your career?

Module 16

How to Succeed This Week

Quick Tip for Success:

There can be a steep learning curve when it comes to Big Data and we'll only just be scratching the surface of what we can learn. Use this as a starting point!

Module 16

Today’s Agenda

Today's Agenda

By completing today's activities, you'll learn the following skills:

01 Using Google Colab notebooks

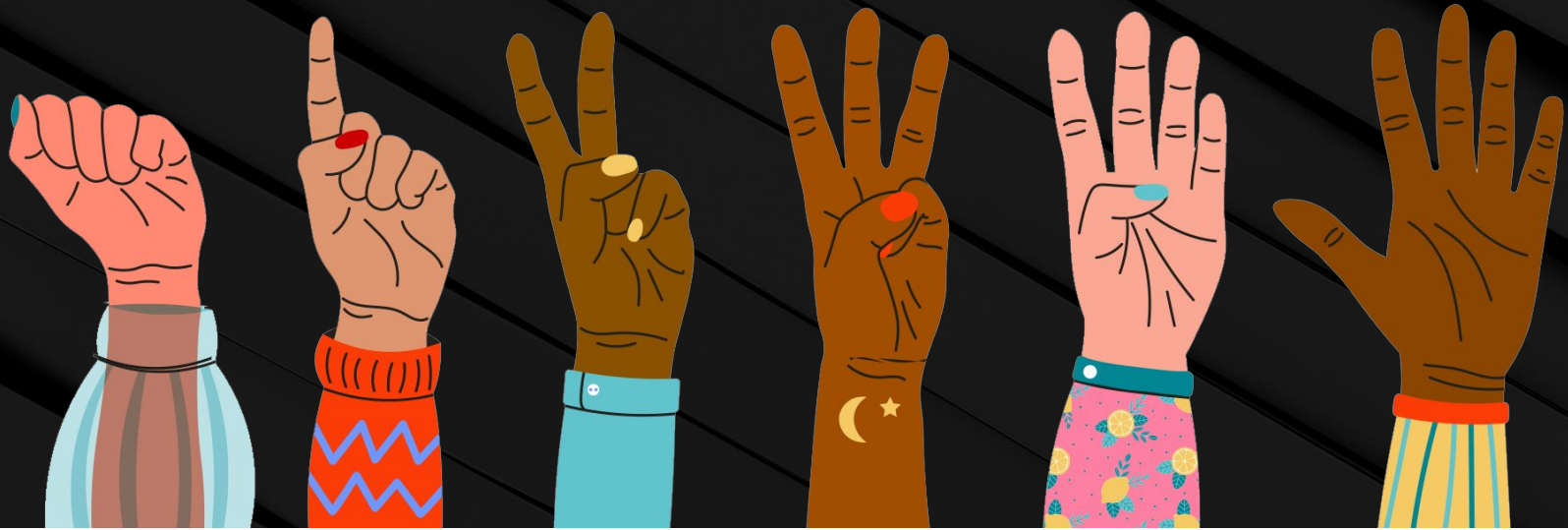
02 Storing and filtering datasets in PySpark DataFrames

03 Working with dates in data and plotting results

Make sure you've downloaded any relevant class files!

FIST TO FIVE:

How comfortable do you feel with this topic?



Spark Overview

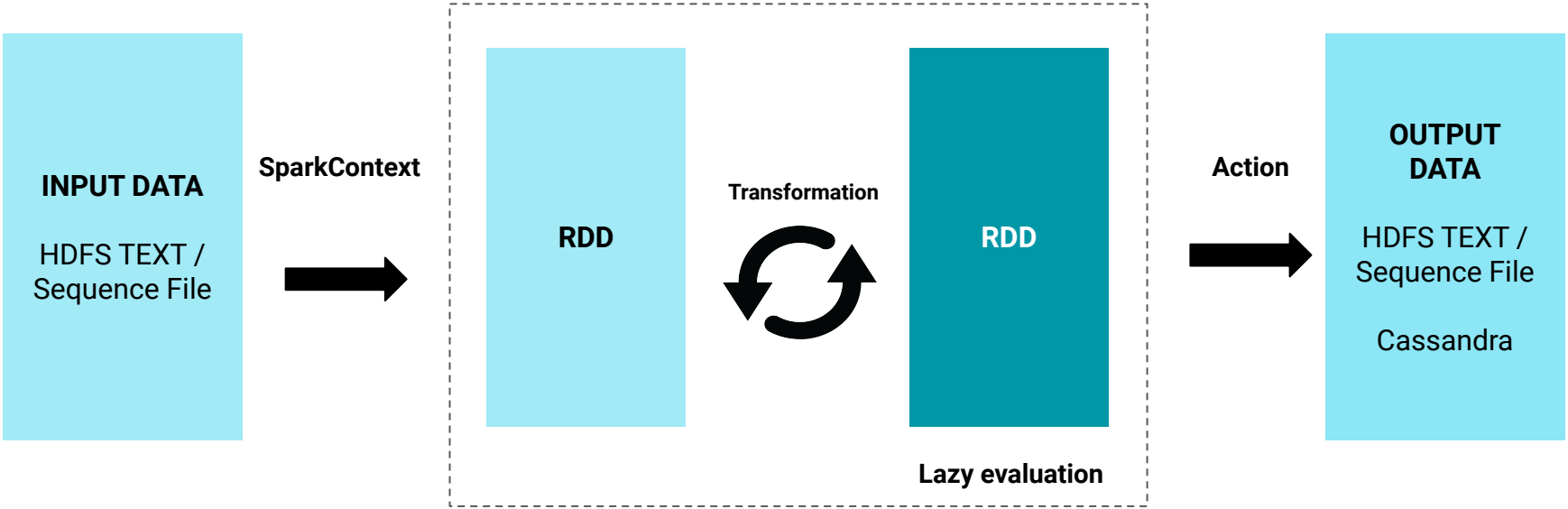
Spark is a fast and general engine for large-scale data processing. Spark uses scripts from real programming languages, has a rich ecosystem, and is very scalable.

Hadoop is a buzzword in the big data industry, but many businesses are relying on Spark to solve their big data problems. Spark runs on Hadoop, but it doesn't have to.

Differences between Spark and Hadoop

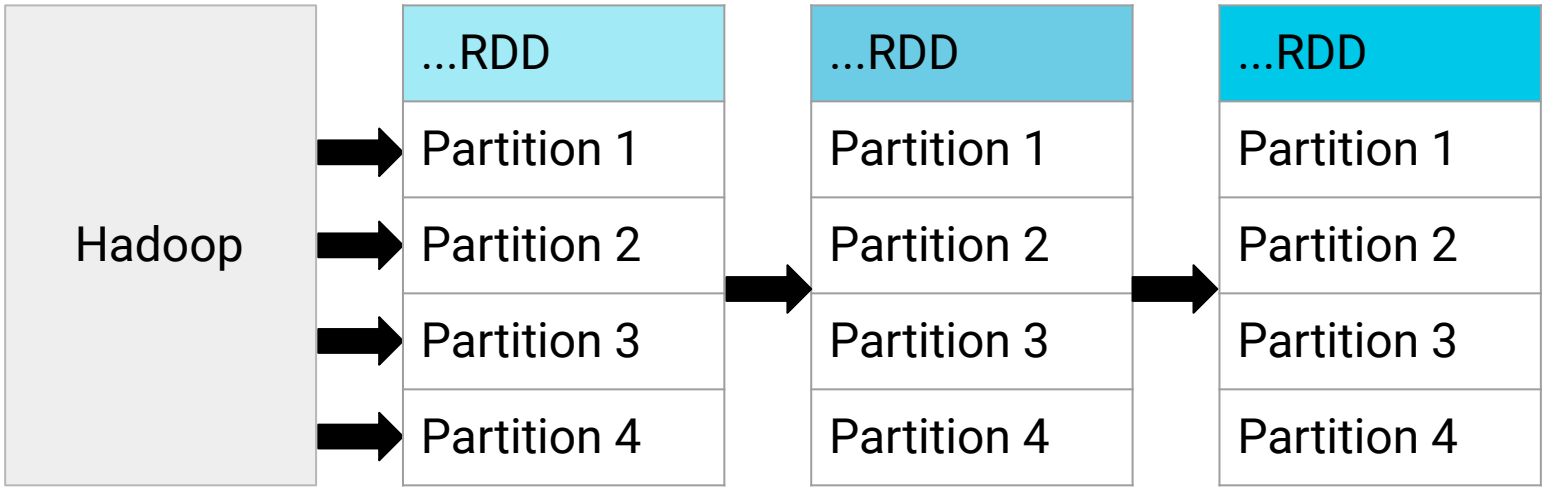
What Is Spark?

Spark lets you **write applications** in **Java, Scala, Python, R,** and **SQL**, and it can **run standalone, on Hadoop,** or **in the cloud** (and many other platforms).



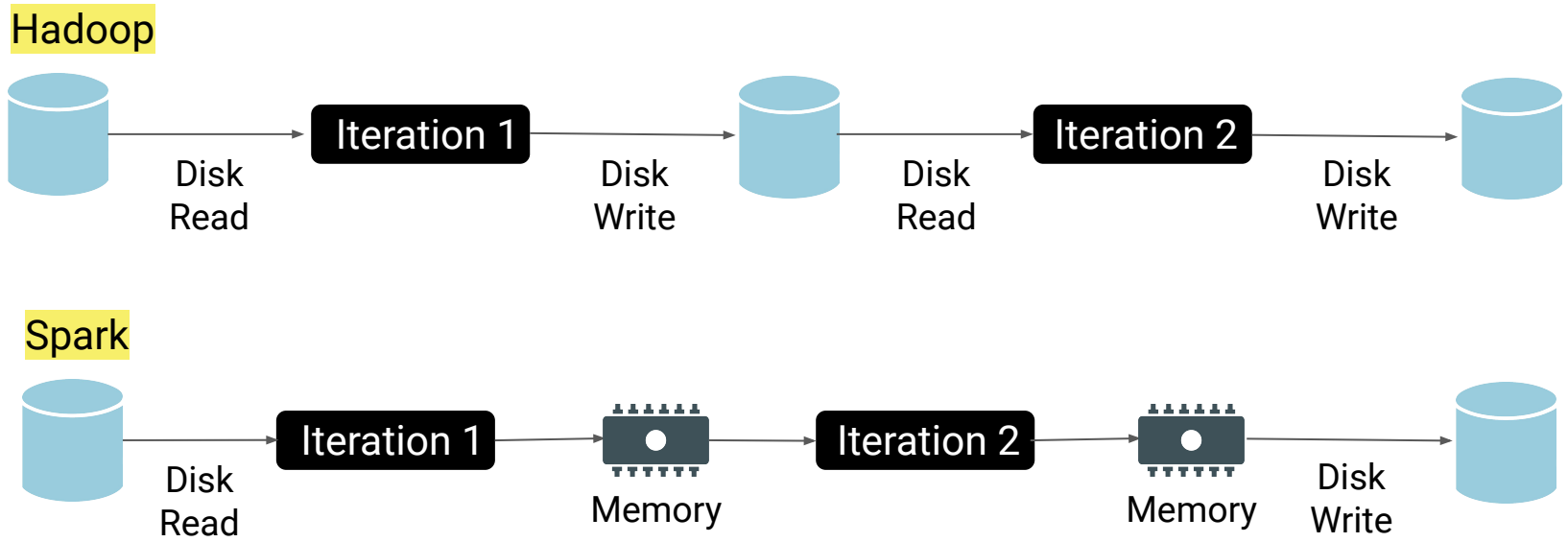
What Is Hadoop?

Hadoop is a file system that is used to store data across server clusters, and it is **scalable, fault-tolerant,** and **distributed.**



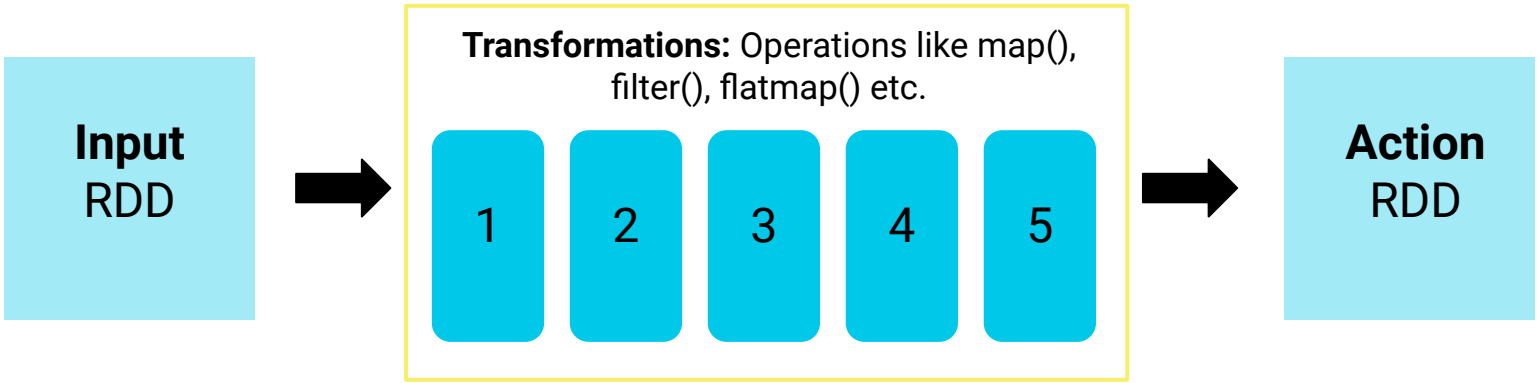
Spark versus Hadoop

Spark uses **in-memory computation** instead of a disk-based solution, which means it doesn't need to talk to the Hadoop Distributed File System (HDFS) each time; instead, **Spark retains as much as it can in memory.**



Spark versus Hadoop

Spark uses lazy evaluation, which delays the evaluation of an expression until its value is needed.



Spark **maintains the lineage of transformations** but does not evaluate them until an action is called.

Set Up Google Colab

Suggested Time:

10 minutes

Spark DataFrames

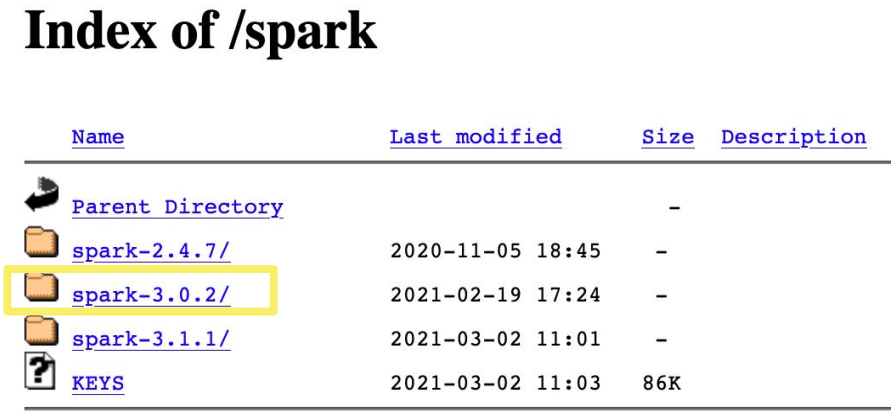
Instructor Demonstration

PySpark DataFrame Basics

When using Colab, each notebook will need to install Spark and create a SparkSession.

Installing Spark

Note: Spark is constantly being updated, and the version used in the code below may be outdated. If you run into installation issues, **visit the Spark distribution to find the most recent version of Spark 3.X.X, then update the version in the variable below**. You will need to update this for all notebooks.



Index of /spark

Name	Last modified	Size	Description
Parent Directory		-	
spark-2.4.7/	2020-11-05 18:45	-	
spark-3.0.2/	2021-02-19 17:24	-	
spark-3.1.1/	2021-03-02 11:01	-	
KEYS	2021-03-02 11:03	86K	

```
Import os

# Find the latest version of spark
# 3.0 from http://www-us.apache.org/dist/spark/ and
# enter as the spark version
# For example:
# spark_version  = 'spark-3.0.2'

Spark_version = 'spark-3.0.<enter version>'
os.environ['SPARK_VERSION']=spark_version
```

Creating a SparkSession

A **SparkSession is created to control your Spark Application**. Before interacting with Spark, a session is started and the app is named; this can be any name, but it is **usually good to associate the app with what you are doing**.

```
# Start Spark session
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("DataFrameBasics").getOrCreate()
```

Creating DataFrames

Spark can create DataFrames manually.

```
# Create DataFrame manually
dataframe = spark.createDataFrame([
                                   (0, "Here is our DataFrame"),
                                   (1, "We are making one from
scratch"),
                                   (2, "This will look very similar to
a Pandas DataFrame")
], ["id", "words"])

dataframe.show()
```

Reading Datasets

Since **Colab is hosted in the cloud, it's much easier to read datasets directly from the cloud as well**, compared to reading from your local files. In this code block, Colab will **pull data from Amazon's Simple Storage Service (S3)**. This boilerplate code can be used to read other public files hosted on Amazon's services.

```
# Read in data from S3 Buckets
from pyspark import SparkFiles
url = "https://s3.amazonaws.com/dataviz-curriculum/day_1/food.csv"
spark.sparkContext.addFile(url)
df = spark.read.csv(SparkFiles.get("food.csv"), sep=",", header=True)
```

Displaying Data

Spark uses the **show()** method to display the data from DataFrames.

```
# Read our data with our new schema
dataframe = spark.read.csv(SparkFiles.get("food.csv"), schema=final,
sep=",", header=True)
dataframe.show()
```

Accessing DataFrames

Spark can access the DataFrame in many different ways:

```
dataframe['price']

type(dataframe['price'])

dataframe.select('price')

type(dataframe.select('price'))

dataframe.select('price').show()
```

Methods for Columns

1. Columns can be manipulated using the **withColumn()** method.

```
# Add new column
dataframe.withColumn('newprice', dataframe['price']).show()
```

2. Columns can be renamed using **withColumnRenamed()**.

```
# Update column name
dataframe.withColumnRenamed('price','newerprice').show()
```

Methods for Columns

3. Column data can be changed.

```
# Double the price
dataframe.withColumn('doubleprice',dataframe['price']*2).show()

# Add a dollar to the price
dataframe.withColumn('add_one_dollar',dataframe['price']+1).show()

# Halve the price
dataframe.withColumn('half_price',dataframe['price']/2).show()
```

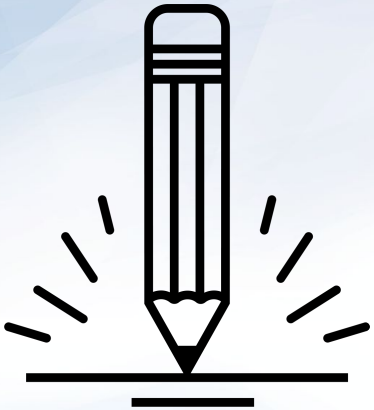
4. A list can be made out of columns with **collect()**.

```
# Collecting a column as a list
dataframe.select("price").collect()
```

Converting a PySpark DataFrame to a Pandas DataFrame

Use **toPandas()** to convert a **PySpark DataFrame to a Pandas DataFrame**. This should only be done for summarized or aggregated subsets of the original Spark DataFrame.

```
import pandas as pd
pandas_df = dataframe.toPandas()
```



Demographic DataFrame Basics

In this activity, you will get the chance to explore Spark DataFrames. Follow the comments in the notebook to clean and display stock data using Spark DataFrames. Remember to consult the documentation.

Suggested Time:
15 minutes

Let's Review

Filtering DataFrames

Instructor Demonstration

PySpark DataFrame Filtering

Ordering DataFrames

Spark can order DataFrames by using the **orderBy()** method.

Pass in the column name and either **asc()** for ascending order or **desc()** for descending order.

```
# Order a DataFrame by ascending values
df.orderBy(df["points"].asc()).show(5)

# Order a DataFrame by descending values
df.orderBy(df["points"].desc()).show(5)
```

Importing Helper Functions

1. **avg()** finds the average of the values in the input column.

```
# Import average function
from pyspark.sql.functions import avg
df.select(avg("points")).show()
```

2. The **filter()** method allows more data manipulation, similar to the WHERE clause in SQL. Here, it is filtering for all wine that has a price lower than $20.

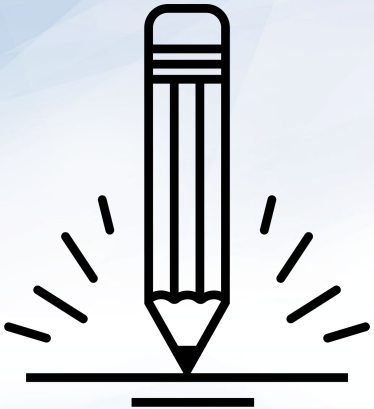
```
# Using filter
df.filter("price<20").show()
```

Importing Helper Functions

3. The exact columns can be used by combining the **select** method with **filter**.

```
# Filter by price on certain columns
df.filter("price<20").select(['points','country',
'winery','price']).show()
```

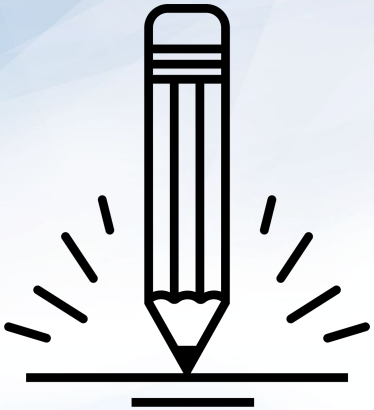
PySpark Demographic Filtering



Using PySpark methods and the demographics dataset, answer the following questions:

1. Which occupation had the **highest salary?**
2. Which occupation had the **lowest salary?**
3. What is the **mean salary** of this dataset?
4. What is the **max and min of the Salary column?**
5. Which **occupations have salaries above 80k? List all of them.**

Suggested Time:
20 minutes



BONUS

What is the average age and height in people with each academic degree type?

Hint: You will need to use groupBy to answer this question.

Suggested Time:

Let's Review

Dates and Plotting

Instructor Demonstration

PySpark DataFrame Dates

Handling Data Formats and Plotting Data with PySpark

- To avoid errors in reading the data,
  `inferSchema=True, timestampFormat="yyyy/MM/dd HH:mm:ss"`
  is used to tell Spark to infer the schema and use this format for handling timestamps.

- It's **common to encounter a variety of date and timestamp formats**.

- Spark provides a **functions library** with date and timestamp conversion functions.

Handling Data Formats and Plotting Data with PySpark

The **year function** is imported, which allows you to select the year from a timestamp column.

```
# Import date time functions
from pyspark.sql.functions import year

# Show the year for the date column
df.select(year(df["date"])).show()
```

A new **column storing only the year** can be created.

```
# Save the year as a new column
df = df.withColumn("year", year(df['date']))
df.show()
```

Handling Data Formats and Plotting Data with PySpark

With the new column, we can now **group by the year** and **find the average precipitation**.

```
# Find the average precipitation per year
averages = df.groupBy("year").avg()
averages.orderBy("year").select("year", "avg(prcp)").show()
```

The same can be done with the month function, except this time we'll use the **max()** function. The DataFrame can also be exported to a **Pandas DataFrame**.

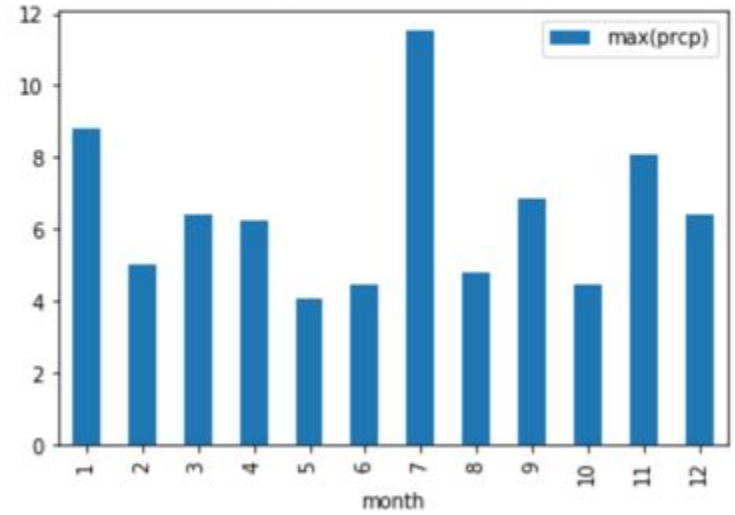
```
# Import the summarized data to a pandas dataframe for plotting
# Note: If your summarized data is still too big for your local memory
then your notebook may crash

pandas_df = averages.orderBy("month").select("month",
"max(prcp)").toPandas()
pandas_df.head()
```

Handling Data Formats and Plotting Data with PySpark

From the Pandas DataFrame, we can use **Matplotlib to chart the data**.

```
import matplotlib.pyplot as plt
pandas_df.set_index("month", inplace=True)
pandas_df.plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f256b512860>
```



Time to Code

Plotting BigFoot

Suggested Time:

15 minutes

Questions?