**Functions**

In the previous video, we've used a number of functions to manipulate our dataframe. Let's take a look at the different type of functions and their potential pitfalls.

**General functions**

We have used the following general functions that are quite similar to methods of pandas dataframes:

* select(): returns a new DataFrame with the selected columns
* filter(): filters rows using the given condition
* where(): is just an alias for filter()
* groupBy(): groups the DataFrame using the specified columns, so we can run aggregation on them
* sort(): returns a new DataFrame sorted by the specified column(s). By default the second parameter 'ascending' is True.
* dropDuplicates(): returns a new DataFrame with unique rows based on all or just a subset of columns
* withColumn(): returns a new DataFrame by adding a column or replacing the existing column that has the same name. The first parameter is the name of the new column, the second is an expression of how to compute it.

**Aggregate functions**

Spark SQL provides built-in methods for the most common aggregations such as count(), countDistinct(), avg(), max(), min(), etc. in the pyspark.sql.functions module. These methods are not the same as the built-in methods in the Python Standard Library, where we can find min() for example as well, hence you need to be careful not to use them interchangeably.

In many cases, there are multiple ways to express the same aggregations. For example, if we would like to compute one type of aggregate for one or more columns of the DataFrame we can just simply chain the aggregate method after a groupBy(). If we would like to use different functions on different columns, agg()comes in handy. For example agg({"salary": "avg", "age": "max"}) computes the average salary and maximum age.

**User defined functions (UDF)**

In Spark SQL we can define our own functions with the udf method from the pyspark.sql.functions module. The default type of the returned variable for UDFs is string. If we would like to return an other type we need to explicitly do so by using the different types from the pyspark.sql.types module.

**Window functions**

Window functions are a way of combining the values of ranges of rows in a DataFrame. When defining the window we can choose how to sort and group (with the partitionBy method) the rows and how wide of a window we'd like to use (described by rangeBetween or rowsBetween).

**Lesson Overview**

By the end of the lesson, you will be able to:

* Distinguish between setting up a Spark Cluster using both Local and Standalone Mode
* Set up Spark Cluster in AWS
* Use Spark UI
* Use AWS CLI
* Create EMR using AWS CLI
* Create EMR Cluster
* Test Port Forwarding
* Use Notebooks on your Spark Cluster
* Write Spark Scripts
* Store and Retrieve data on the Cloud
* Read and Write to Amazon S3
* Understand the distinction between HDFS and S3
* Reading and Writing Data to HDFS

**Overview of the Set up of a Spark Cluster**

1. **Amazon S3** will store the dataset.
2. We rent a cluster of machines, i.e., our **Spark Cluster**, and iti s located in AWS data centers. We rent these using AWS service called **Elastic Compute Cloud (EC2)**.
3. We log in from your local computer to this Spark cluster.
4. Upon running our Spark code, the cluster will load the dataset from **Amazon S3** into the cluster’s memory distributed across each machine in the cluster.

**New Terms:**

* **Local mode**: You are running a Spark program on your laptop like a single machine.
* **Standalone mode**: You are defining Spark Primary and Secondary to work on your (virtual) machine. You can do this on EMR or your machine. Standalone mode uses a resource manager like YARN or Mesos.

### EC2 vs EMR

|  |  |  |
| --- | --- | --- |
|  | **AWS EMR** | **AWS EC2** |
| **Distributed computing** | Yes | Yes |
| **Node categorization** | Categorizes secondary nodes into core and task nodes as a result of which data can be lost in case a data node is removed. | Does not use node categorization |
| **Can support HDFS?** | Yes | Only if you configure HDFS on EC2 yourself using multi-step process. |
| **What protocol can be used?** | Uses S3 protocol over AWS S3, which is faster than s3a protocol | ECS uses s3a |
| **Comparison cost** | Bit higher | Lower |

### Circling back about HDFS

Previously we have looked over the Hadoop Ecosystem. To refresh those concepts, we have provided reference material here. HDFS (Hadoop Distributed File System) is the file system. HDFS uses MapReduce system as a resource manager.

Spark can replace the MapReduce algorithm. Since Spark does not have its own distributed storage system, it leverages using HDFS or AWS S3, or any other distributed storage. Primarily in this course, we will be using AWS S3, but let’s review the advantages of using HDFS over AWS S3.

### What is HDFS?

HDFS (Hadoop Distributed File System) is the file system in the Hadoop ecosystem. Hadoop and Spark are two frameworks providing tools for carrying out big-data related tasks. While Spark is faster than Hadoop, Spark has one drawback. It lacks a distributed storage system. In other words, Spark lacks a system to organize, store and process data files.

### MapReduce System

HDFS uses MapReduce system as a resource manager to allow the distribution of the files across the hard drives within the cluster. Think of it as the MapReduce System storing the data back on the hard drives after completing all the tasks.

Spark, on the other hand, runs the operations and holds the data in the RAM memory rather than the hard drives used by HDFS. Since Spark lacks a file distribution system to organize, store and process data files, Spark tools are often installed on Hadoop because Spark can then use the Hadoop Distributed File System (HDFS).

### Why do you need ****EMR Cluster****?

Since a Spark cluster includes multiple machines, in order to use Spark code on each machine, we would need to download and install Spark and its dependencies. This is a manual process. **Elastic Map Reduce** is a service offered by AWS that negates the need for you, the user, to go through the manual process of installing Spark and its dependencies for each machine.