# Spark

## Basic Working

1. Lazy Evaluation - Procastinate steps for all sub function, to avoid runnin gout of memory
2. Collect function to gather data from all clusters
3. Fault Tolerant Way

### Fault Tolerant and HDFS

1. Hadoop saves data in 64-129 MB chunks across clusters.
2. Also could be saved on AWS s3

### Spark Context

1. From pyspark import SparkContext, SparkConf
2. Conf = SparkConf().SetApp

### Spark SQL Session

##### Tip

If Spark is used in a cluster mode all the worker nodes need to have access to the input data source. If you're trying to import a file saved only on the local disk of the driver node you'll receive an error message similar to this:

AnalysisException: u'Path does not exist: file:/home/ubuntu/test.csv;'

Loading the file should work if all the nodes have it saved under the same path.

## Import Spark Context and Parallelize

1. import pyspark
2. sc = pyspark.SparkContext(appName="maps\_and\_lazy\_evaluation\_example")
3. distributed\_song\_log = sc.parallelize(log\_of\_songs)

# Imperative and Declarative Programming

Imperative = Spark DataFrame and Python – cares How – Focus on steps to ge result

Declarative Programing – SQL – What - We focus on outcome

# 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).

For further information see the [**Spark SQL, DataFrames and Datasets Guide**](https://spark.apache.org/docs/latest/sql-programming-guide.html) and the [**Spark Python API Docs**](https://spark.apache.org/docs/latest/api/python/index.html).

# Spark SQL resources

Here are a few resources that you might find helpful when working with Spark SQL

* [**Spark SQL built-in functions**](https://spark.apache.org/docs/latest/api/sql/index.html)
* [**Spark SQL guide**](https://spark.apache.org/docs/latest/sql-getting-started.html)

From SparkSQL to PostGreSQL View

User\_log.createOrReplaceTempView(‘user\_log\_table’)

# Setting Spark on AWS

Three modes:

1. Standalone mode
2. MESOS
3. YARN

## 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.