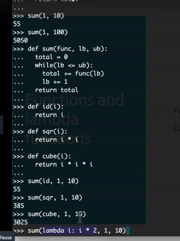
Python Data Engineering:

Import sys

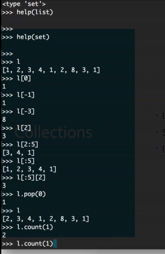
With above library we n pass arguments into python using the argv array

* Spark uses hadoop HDFS API to read data from files and also to write data into files.

You can either pass functions directly inside a function like lambda function or define fucntions separately to get called from another function:

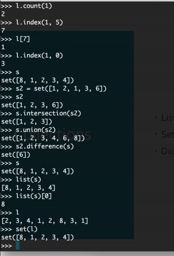


**Pop,append,count etc on a List :**

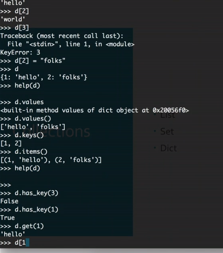


**Set Operations:**

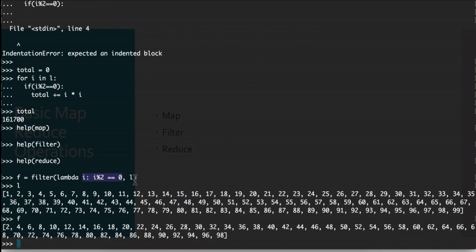
Intersection,difference,union etc between sets.



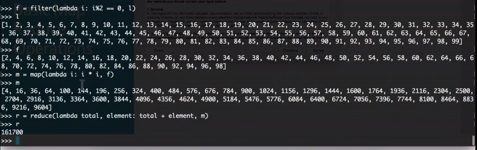
Dictionaries:



Filter and Reduce:



Map Reduce perspective:



To write a first python shell script which will take 2 arguments by consuming a file and produce an output





Hadoop file system ls:

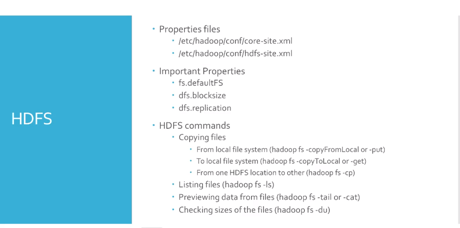
hadoop fs –ls /public



**Hadoop file system information:**

Below ee shall see basic Hadoop commands to list the file systems in hdfs etc.

Hadoop configuration files.



Hadoop commands:

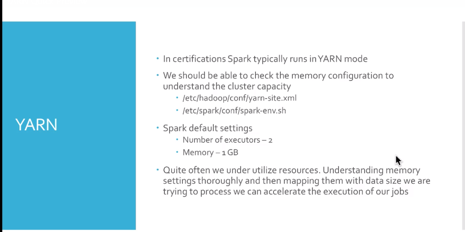
hadoop fs –ls

hadoop fs –help copyFromLocal

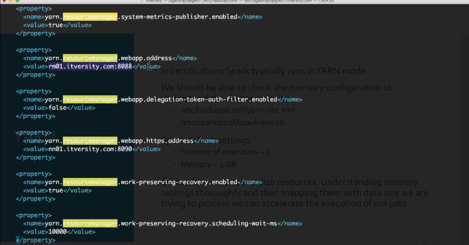
To copy a file form local to inside Hadoop file system:

Hadoop fs –copyFromLocal /sourcepath /destinationpath

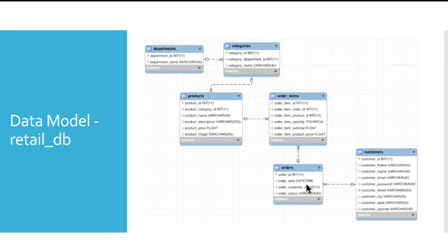
YARN ( Yet Another Resource Negotiator) :

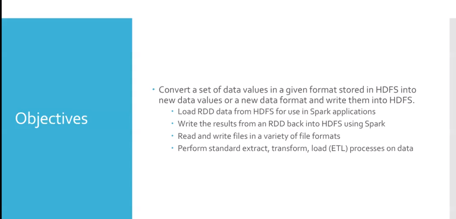


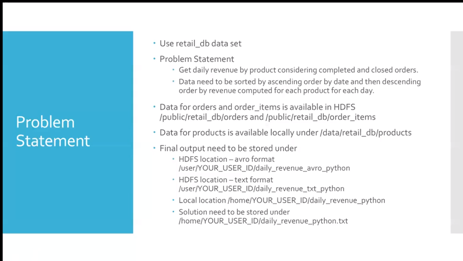
YARN Resource namager uri

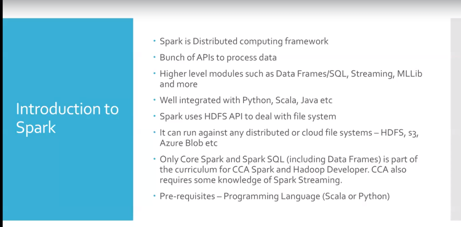


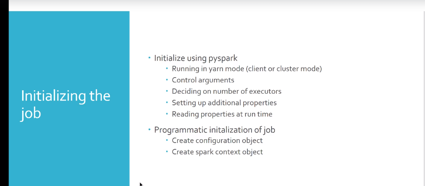
**A atypical E-commerce Data model:**

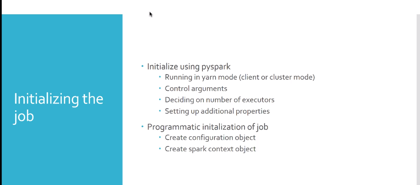












Pyspark will launch python console with spark modules and on top of it, it will create a spark Webservice modules and that Webservice is assigned to a variable called sc.

Below will insitialize pyspark in yarn mode:

Pyspark –master –conf spark.ui.port= 12888



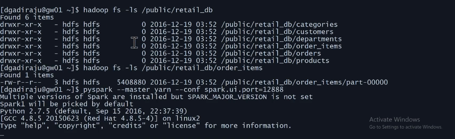
This will launch spark



In actual world, we need to know how to initialize the job using programmatic approach rather than from console mode:

Now let’s get into the job:

We can launch pyspark job to read data inside a Hadoop file system and create a RDD out of it.



To Open a file and create a RDD:

* help(sc)
* help(sc.textFile)
* orderItems = sc.textFile(“/public/retail\_db/order\_tems”)

This will create a RDD

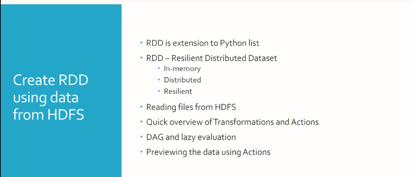
* type(orderItems)
* help(orderItems)
* orderItems

TO see the first element in the file

ordeItems.first()

To see first 10 itesms in the list then below:

>>> for I in orderItems.take(10): print(i)

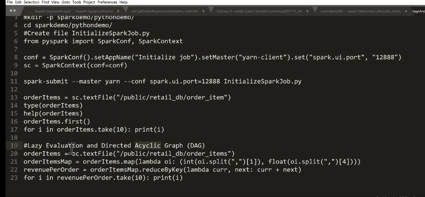


**RDD’s** have **actions** and **transformations** and you can see the list in below url

<https://spark.apache.org/docs/latest/rdd-programming-guide.html>

you can click above link and go through all the API’s available under RDD’s under

Actions and transformations where are a total of 25 Core Spark API’s , if we master those then we are all good to use spark:

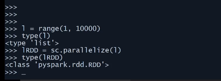


In Above code, the first lines will not run a job it just does evaluation, te last for loop is the one which actually creates and run a job.

To convert RDD inti a apythin collection:

>>> orderItemsList = orderItems.collect()

To convert collection into a RDD;



**We can directly read a file from fdfs and then convert into a RDD using below commands:**

**How to convert a raw hdfs file into a RDD**



**How to read data from different file formats:**

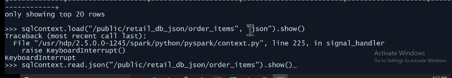
**Below are 4 ways on how we can read files from different formats:**

**Below shows how to open a a json file**

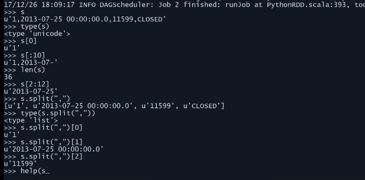


We can also do the same withour load and do read option.

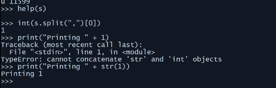
Load vs read



**How to split the first line from a file and convert the first line into an array:**



**Converting string to integer and from integer to string:**



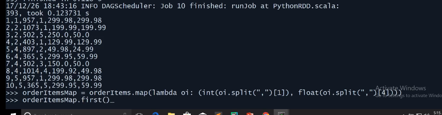
**Split and Replace manipulation:**



Covert first 10 elements through Map transformation from a RDD:



To extract by orderId and the corresponding revenue column:

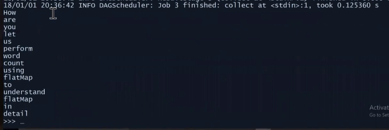


For first 10 records



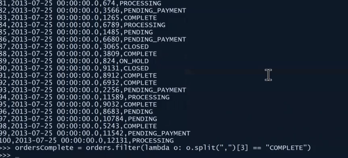
**flatMap() :**





**Filtering:**

Filtering by default gives horizontal filtering and below is an example of how to filter records for a particular value which will return true:



To print 100 records with status = ‘COMPLETE’



**Filter with IN clause:**



**Double filter with records for 2014**



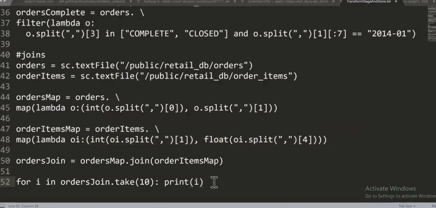
**Filter With IN Clause:**



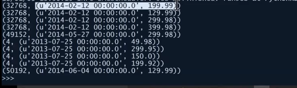
**Inner Join**

Create map for the 2 data sets that needs to be joined and then then do the join using

.join function

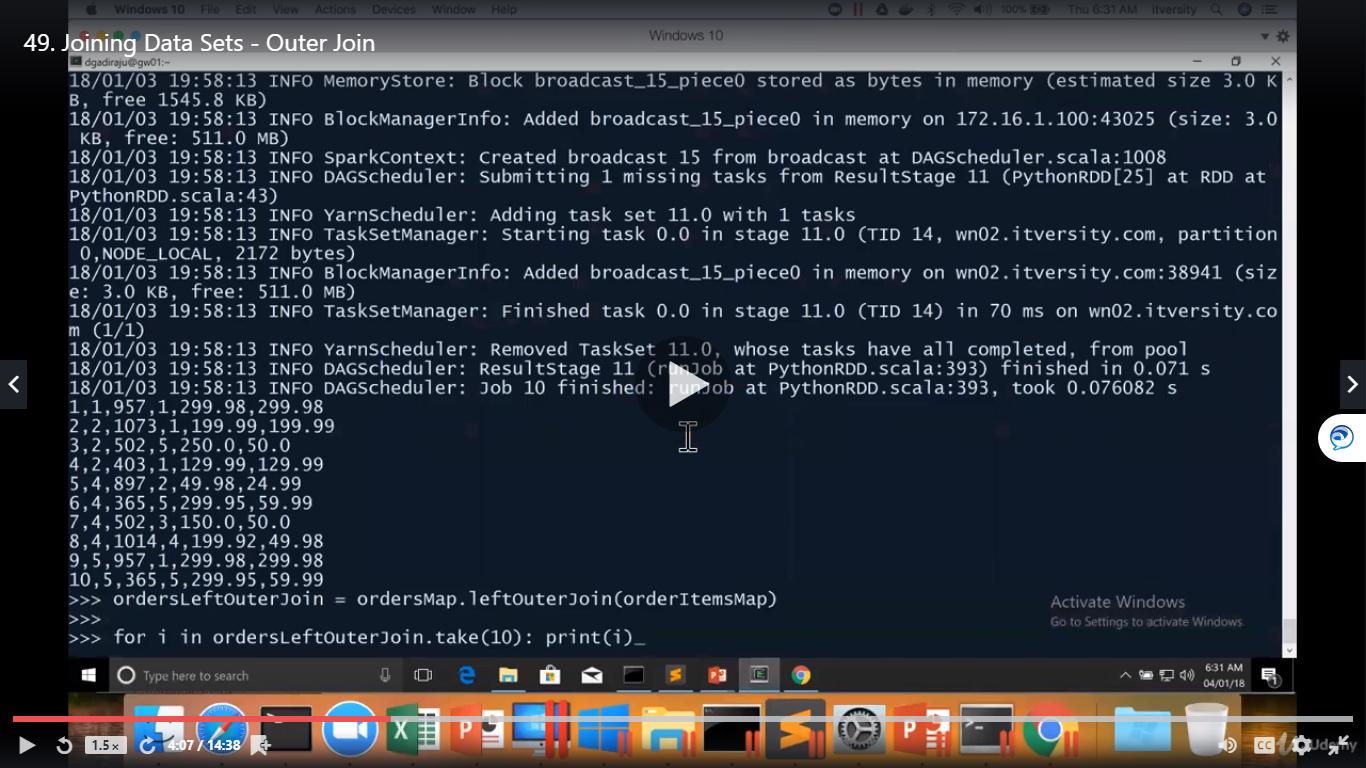


**Output:**

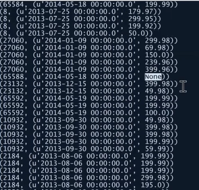


**Left outer join on 2 data sets or tables:**

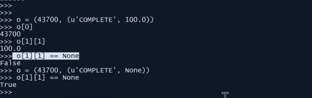


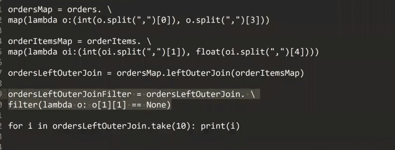


**Below output we can see data being none (nul) from the rght table.**



If we need to apply another filter from the output of a Left Outer Join then, we can apply below logic.

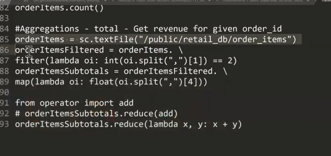




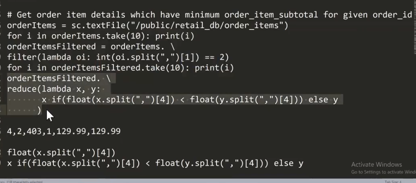
**Right Outer Join:**



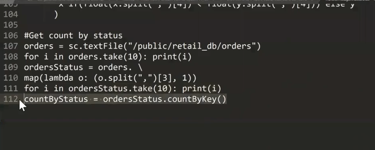
**Reduce function : ( Group by function):**



Rank by min or max function by using If logic:



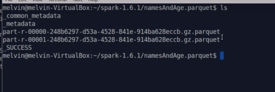
**CountByKey ( like Group by):**



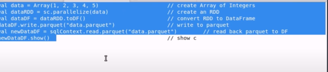
**How to convert a file to Parquet format :**



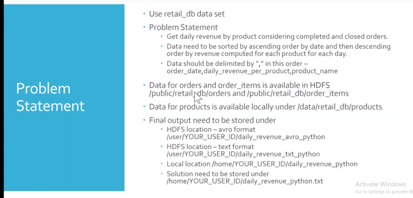
**Below is how Parquet file is stored in the Hadoop file system as a parquet directory:**

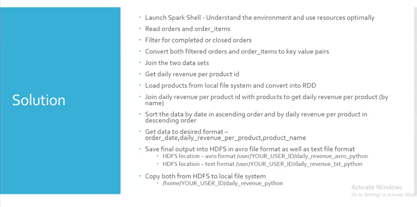


**Creating a parquet file and how to read from it:**

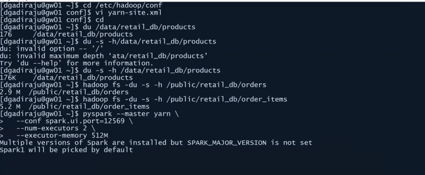


**Problem –Statement:**





**Launch Pythion spark shell with desired number of executors:**

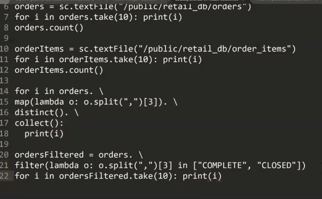


Enter spark-submit to evaluate all control arguments available in spark.

* spark-submit

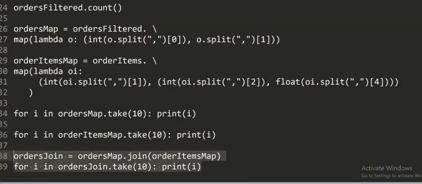
Now let’s work with the files:

We shall open the 2 files with sc.textfile and then can perform filters and map to work with the data.

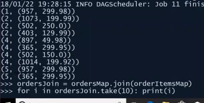


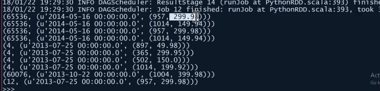
**Join:**

Now we need to join orders and orderItems for that we need to convert those 2 as key value pairs and then get the ouput through join



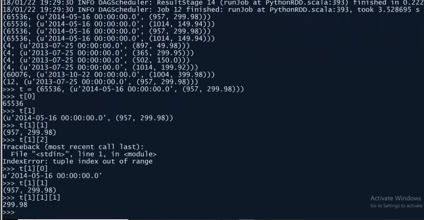
Output:





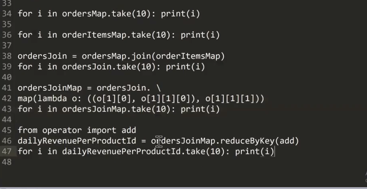
**Aggregate to get daily revenue by productid:**

We can use array positions of the data to perform the required aggregation as below where t[1][1][1] will give the revenue for that productid

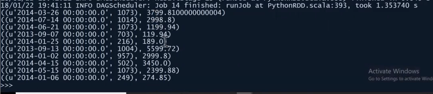


**Reduce:**

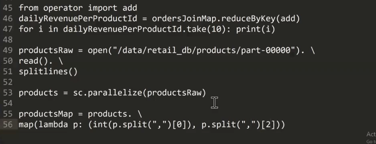
Now we can do aggregation based on reduce API on the desired keys:



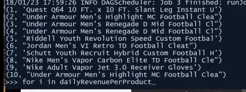
**Got the ouput as for each date for that productid what is the total revenuew:**



We can get the product name fronm the hdfs file like below where we are importing the hdfs file into a RDD



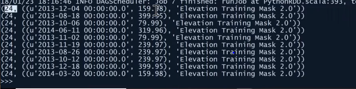
Output:

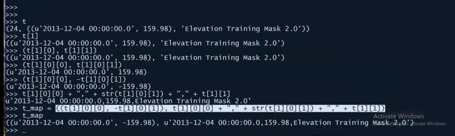


Now we need to group by:

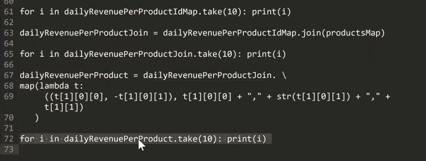


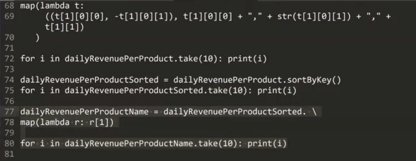
Output:



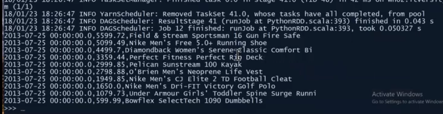


Below we can group by date and sort it by revenue

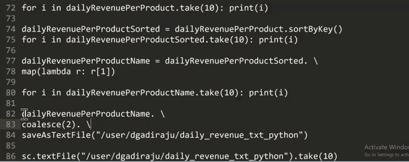




**To get the final output of revenue sorted by date in ascending order and revenue by descending order**

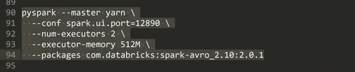


Save the file as textfile:

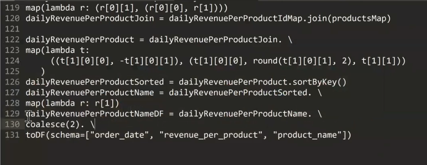


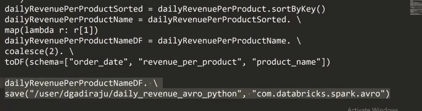
**To save the file as AVRO file format:**

We need to plugin the third party tool as jar file and then restart pyspark:



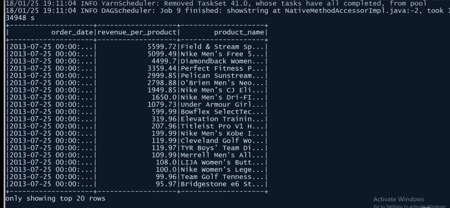
Modify the pipeline for Avro format:





Avro format dataframe output:

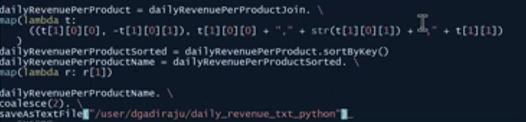
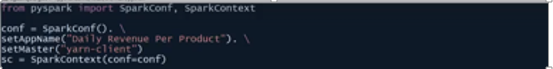




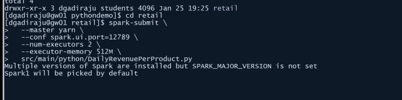
**Copy the file from hdfs to local:**



Final Python Pipeline:



**Now let’s submit the job:**



**Spark SQL:**



Below we can work with Data Frames and Hive tables:

