PYSPARK TUTORIAL

APACHE SPARK

Spark is an open source software developed by UC Berkley in 2009 and it was made available to the public in 2010 and since then has been used within the industry with an unprecedented scale. Earlier tools like MapReduce were used by Data Scientist to process streaming of data, however they were slow. To overcome this issue, spark offers a solution that is both fast and general purpose. The main difference between Spark and MapReduce is that Spark runs computations in memory while the later on the hard disk and allows high-speed access and data processing, reducing times from hours to minutes.

PYSPARK

Spark is the name of the engine to realise cluster computing while PySpark is the Python's library to use Spark

WORKING MECHANISM

Spark takes care of scheduling , distributing and monitoring application as it is based on computational engine. Each task is done across various worker machines called computing cluster. A computing cluster refers to the division of tasks. One machine performs one task, while the others contribute to the final output through a different task. In the end, all the tasks are aggregated to produce an output.

ADVANTAGES OF SPARK:

- Pyspark handles the complexities of multiprocessing, such as distributing the data, distributing code and collecting output from the workers on a cluster of machines.
- It can build an architecture that encompasses data streaming management, seamlessly data queries, machine learning prediction and real-time access to various analysis.
- Spark works closely with SQL language, i.e., structured data. It allows querying the data in real time.

INSTALLATION

We assume that you have already installed anaconda and have some basic knowledge of python and sql to follow along with the tutorial.

1. Go to Apache Spark website.
2. Choose a spark release. In our case we have used 2.4.3 (May 07 2019).
3. Choose a package type. It will be selected by default.
4. Click the Download spark link.

You can follow the link to install Spark.

Credits: Michael Galarnyk.

Note: Remember to change the spark-2.1.0-bin-hadoop2.7 to spark-2.4.3-bin-hadoop2.7 where it is applicable as we are using the latest version of spark.

SparkContext

Spark Context is the internal engine that that initiates connection with clusters. It sets up internal services and establishes connection with spark execution environment. Through SparkContext you can create RDDs, broadcast variables, etc.

In [ ]:
```
!conda install nbconvert
```

In [1]:
```
import pyspark
from pyspark import SparkContext
```

In [22]:
```
spark  #Gives the name of the Spark version running, Master and AppName
```

Out[22]: **SparkSession - hive**

SparkContext

Spark UI

Version

`v2.4.3`

Master

`local[4]`

AppName

`PySparkShell`

RDD

Once the SparkContext is initialised, you can create a RDD. RDDs are said to be the building blocks for Spark. The fullform is: R - Ressilient: It is fault tolerant and can rebuild data on failure D - Distributed: Data is distributed among multiple nodes on a cluster D - Dataset: Collection of partioned data with values

In [2]:
```
nums= sc.parallelize([1,2,3,4])
```

nums is a RDD and the code uses parallelize to tranfer chunks of data to various nodes. By default the total data is divided into 128MB per node. Example: If we have 1GB of data then it is divided into (1000/128) approximately 8 nodes.

TRANSFORMATION AND ACTION

Spark Transformation is a function that produces new RDD from the existing RDDs. It takes RDD as input and produces one or more RDD as output. Each time it creates new RDD when we apply any transformation. Thus, the so input RDDs, cannot be changed since RDD are immutable in nature.

Transformations are lazy in nature i.e., they get execute when we call an action. They are not executed immediately.

Common Transformations are map, flatMap, filter, distinct, reduceByKey, sortBy

Common Actions are collect, reduce, take, countByKey/countByValue, show

You can add a tranformation using a lambda function. In the example below, you are returning the square of nums. Here map is the tranformation and collect is the action.

In [50]:
```
squared = nums.map(lambda x: x*x).collect()
for num in squared:
    print(num)
```

```
1
4
9
16
```

SqlContext

SQLContext allows connecting the engine with different data sources. It is used to initiate the functionalities of Spark SQL. Once SqlContext is intialised, you can create a Data Frame in Pyspark. Data Frames are Built on top of RDDs and are different from panda data frames as they are immutable.

In [8]:
```
from pyspark.sql import Row
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

LOADING DATA

Using textFile method and passing the text file to read as an argument we can get the contents of the file into rdd.

type function is used to check the datatype of the variable that is passed as an argument to it.

Note: The output will be a RDD.

In [18]:
```
rdd = sc.textFile("alice_in_wonderland.txt")
type(rdd)
```

Out[18]: `pyspark.rdd.RDD`

While reading csv if InferSchema is not set to true, then Spark does not guess the data type and by default shows String for all columns of the dataset.

Note: The output will be a dataframe

In [16]:
```
df1= sqlContext.read.csv("adult.csv", header=True)
type(df1)
```

Out[16]: `pyspark.sql.dataframe.DataFrame`

printSchema() method is used to display the schema in a tree format where we can know the datatypes of all the columns and whether they can accept null values or not.

In [17]:
```
df1.printSchema()
```

```
root
 |-- age: string (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: string (nullable = true)
 |-- education: string (nullable = true)
 |-- education-num: string (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital-gain: string (nullable = true)
 |-- capital-loss: string (nullable = true)
 |-- native-country: string (nullable = true)
 |-- class: string (nullable = true)
 |-- label: string (nullable = true)
```

To **solve** this, we you put **inferSchema= True**, then Spark automatically guesses the type of data. By **default**, it is taken as **false**.

In [11]:
```
df = sqlContext.read.csv("adult.csv", header=True, inferSchema= True)
```

In [6]:
```
df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: integer (nullable = true)
 |-- education: string (nullable = true)
 |-- education-num: integer (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital-gain: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- native-country: integer (nullable = true)
 |-- class: string (nullable = true)
 |-- label: string (nullable = true)
```

SHOW DATA

show() method is used to show the contents of the dataframe. By **default** it will display only **20 rows**.

In [21]:
```
df.show()
```

```
+---+----------------+------+------------+-------------+-------------------+-----------------
-+-------------+------------------+-------+------------+------------+-------------+----------
----+------+
|age|       workclass|fnlwgt|   education|education-num|     marital-status|       occupatio
n|  relationship|              race|    sex|capital-gain|capital-loss|native-country|         c
lass| label|
+---+----------------+------+------------+-------------+-------------------+-----------------
-+-------------+------------------+-------+------------+------------+-------------+----------
----+------+
| 39|       State-gov| 77516|   Bachelors|           13|      Never-married|      Adm-clerica
l| Not-in-family|             White|   Male|        2174|           0|            40| United-St
ates| <=50K|
| 50|Self-emp-not-inc| 83311|   Bachelors|           13| Married-civ-spouse|   Exec-manageria
l|       Husband|             White|   Male|           0|           0|            13| United-St
ates| <=50K|
| 38|         Private|215646|     HS-grad|            9|           Divorced| Handlers-cleaner
s| Not-in-family|             White|   Male|           0|           0|            40| United-St
ates| <=50K|
| 53|         Private|234721|        11th|            7| Married-civ-spouse| Handlers-cleaner
s|       Husband|             Black|   Male|           0|           0|            40| United-St
ates| <=50K|
| 28|         Private|338409|   Bachelors|           13| Married-civ-spouse|    Prof-specialt
y|          Wife|             Black| Female|           0|           0|            40|
Cuba| <=50K|
| 37|         Private|284582|     Masters|           14| Married-civ-spouse|   Exec-manageria
l|          Wife|             White| Female|           0|           0|            40| United-St
ates| <=50K|
| 49|         Private|160187|         9th|            5| Married-spouse-a...|    Other-servic
e| Not-in-family|             Black| Female|           0|           0|            16|        Jam
aica| <=50K|
| 52|Self-emp-not-inc|209642|     HS-grad|            9| Married-civ-spouse|   Exec-manageria
l|       Husband|             White|   Male|           0|           0|            45| United-St
ates|  >50K|
| 31|         Private| 45781|     Masters|           14|      Never-married|    Prof-specialt
y| Not-in-family|             White| Female|       14084|           0|            50| United-St
ates|  >50K|
| 42|         Private|159449|   Bachelors|           13| Married-civ-spouse|   Exec-manageria
l|       Husband|             White|   Male|        5178|           0|            40| United-St
ates|  >50K|
| 37|         Private|280464|Some-college|           10| Married-civ-spouse|   Exec-manageria
l|       Husband|             Black|   Male|           0|           0|            80| United-St
ates|  >50K|
| 30|       State-gov|141297|   Bachelors|           13| Married-civ-spouse|    Prof-specialt
y|       Husband|Asian-Pac-Islander|   Male|           0|           0|            40|          I
ndia|  >50K|
| 23|         Private|122272|   Bachelors|           13|      Never-married|      Adm-clerica
l|     Own-child|             White| Female|           0|           0|            30| United-St
ates| <=50K|
| 32|         Private|205019|  Assoc-acdm|           12|      Never-married|             Sale
s| Not-in-family|             Black|   Male|           0|           0|            50| United-St
ates| <=50K|
| 40|         Private|121772|   Assoc-voc|           11| Married-civ-spouse|      Craft-repai
r|       Husband|Asian-Pac-Islander|   Male|           0|           0|            40|
?|  >50K|
| 34|         Private|245487|     7th-8th|            4| Married-civ-spouse| Transport-movin
g|       Husband|Amer-Indian-Eskimo|   Male|           0|           0|            45|         Me
xico| <=50K|
| 25|Self-emp-not-inc|176756|     HS-grad|            9|      Never-married|   Farming-fishin
g|     Own-child|             White|   Male|           0|           0|            35| United-St
ates| <=50K|
| 32|         Private|186824|     HS-grad|            9|      Never-married| Machine-op-inspc
t|     Unmarried|             White|   Male|           0|           0|            40| United-St
ates| <=50K|
| 38|         Private| 28887|        11th|            7| Married-civ-spouse|             Sale
s|       Husband|             White|   Male|           0|           0|            50| United-St
ates| <=50K|
| 43|Self-emp-not-inc|292175|     Masters|           14|           Divorced|   Exec-manageria
l|     Unmarried|             White| Female|           0|           0|            45| United-St
ates|  >50K|
+---+----------------+------+------------+-------------+-------------------+-----------------
-+-------------+------------------+-------+------------+------------+-------------+----------
----+------+
only showing top 20 rows
```

To show particular number of rows we can pass that **number** as an argument to **show()** method.

In [22]:
```
df.show(5)
```

```
+---+----------------+------+----------+-------------+-------------------+------------------+--
------------+------+-------+------------+------------+--------------+-------------+------+
|age|       workclass|fnlwgt| education|education-num|     marital-status|        occupation|
relationship|  race|    sex|capital-gain|capital-loss|native-country|        class| label|
+---+----------------+------+----------+-------------+-------------------+------------------+--
------------+------+-------+------------+------------+--------------+-------------+------+
| 39|       State-gov| 77516| Bachelors|           13|      Never-married|      Adm-clerical| N
ot-in-family| White|   Male|        2174|           0|            40| United-States| <=50K|
| 50|Self-emp-not-inc| 83311| Bachelors|           13| Married-civ-spouse|   Exec-managerial|
Husband| White|   Male|           0|           0|            13| United-States| <=50K|
| 38|         Private|215646|   HS-grad|            9|           Divorced| Handlers-cleaners| N
ot-in-family| White|   Male|           0|           0|            40| United-States| <=50K|
| 53|         Private|234721|      11th|            7| Married-civ-spouse| Handlers-cleaners|
Husband| Black|   Male|           0|           0|            40| United-States| <=50K|
| 28|         Private|338409| Bachelors|           13| Married-civ-spouse|    Prof-specialty|
Wife| Black| Female|           0|           0|            40|          Cuba| <=50K|
+---+----------------+------+----------+-------------+-------------------+------------------+--
------------+------+-------+------------+------------+--------------+-------------+------+
only showing top 5 rows
```

SELECT COLUMNS

select() method is used to select only the specified columns which are passed as an argument.

Here we can see **select** working as a **transformation** which transform df to a **smaller version** of **df** and we apply **action** to the **smaller version of df** which is **show**.

In [45]:
```
df.select('age','fnlwgt').show(5)
```

```
+---+------+
|age|fnlwgt|
+---+------+
| 39| 77516|
| 50| 83311|
| 38|215646|
| 53|234721|
| 28|338409|
+---+------+
only showing top 5 rows
```

GROUPING VALUES AND APPLYING COUNT

groupBy() method allows you to group **rows** of **data together** and then apply **functions** such as **count()** or **sum()**.

In [25]:
```
df.groupBy("education").count().show()
```

```
+------------+-----+
|   education|count|
+------------+-----+
| Prof-school|  576|
|        10th|  933|
|     7th-8th|  646|
|     5th-6th|  333|
|  Assoc-acdm| 1067|
|   Assoc-voc| 1382|
|     Masters| 1723|
|        12th|  433|
|   Preschool|   51|
|         9th|  514|
|   Bachelors| 5355|
|   Doctorate|  413|
|     HS-grad|10501|
|        11th| 1175|
|Some-college| 7291|
|     1st-4th|  168|
+------------+-----+
```

As we can see the values of count are not easy to read. To **solve** this we can simply sort the values in either ascending or descending order using **sort()** method and passing the column to be sorted as the first argument.

By default **sort()** method will **sort** the column in **ascending** order. To **change** it to **descending** order we simply set **ascending = False** in the second argument.

In [30]:
```
df.groupBy("education").count().sort("count",ascending=False).show()
```

```
+------------+-----+
|   education|count|
+------------+-----+
|     HS-grad|10501|
|Some-college| 7291|
|   Bachelors| 5355|
|     Masters| 1723|
|   Assoc-voc| 1382|
|        11th| 1175|
|  Assoc-acdm| 1067|
|        10th|  933|
|     7th-8th|  646|
| Prof-school|  576|
|         9th|  514|
|        12th|  433|
|   Doctorate|  413|
|     5th-6th|  333|
|     1st-4th|  168|
|   Preschool|   51|
+------------+-----+
```

Below code, will **group** the **data** based on **martial status** and then we applied **aggregate function** named **mean** on the column **captial gain** using **agg()** method.

In [34]:
```
df.groupby('marital-status').agg({'capital-gain': 'mean'}).show()
```

```
+--------------------+-----------------+
|      marital-status|avg(capital-gain)|
+--------------------+-----------------+
|             Widowed|571.0715005035247|
|Married-spouse-a...|653.9832535885167|
|   Married-AF-spouse|432.6521739130435|
|  Married-civ-spouse|1764.8595085470085|
|            Divorced|728.4148098131893|
|       Never-married|376.58831788823363|
|           Separated|535.5687804878049|
+--------------------+-----------------+
```

DESCRIBE DATA

describe() method gives **summary statistics** of the **data**:

1. **count**,
2. **mean**,
3. **standard deviation**,
4. **min**,
5. **max**.

In [30]:
```
df.describe().show()
```

```
+-------+-----------------+------------+-----------------+------------+-----------------+----
----------+-----------------+------------+------------------+-------+-----------------+-------
---------+-----------------+-----------+
|summary|              age|   workclass|           fnlwgt|   education|    education-num|mari
tal-status|       occupation|relationship|              race|    sex|     capital-gain|    cap
ital-loss|   native-country|      class|
+-------+-----------------+------------+-----------------+------------+-----------------+----
----------+-----------------+------------+------------------+-------+-----------------+-------
---------+-----------------+-----------+
|  count|            32561|       32561|            32561|       32561|            32561|
32561|            32561|       32561|             32561|  32561|            32561|           3
2561|            32561|      32561|
|   mean| 38.58164675532078|        null|189778.36651208502|        null| 10.0806793403151|
null|             null|        null|              null|   null|1077.6488437087312| 87.303829734
959|40.437455852092995|       null|
| stddev|13.640432553581356|        null|105549.97769702227|        null|2.572720332067397|
null|             null|        null|              null|   null| 7385.292084840354|402.960218649
002|12.347428681731838|       null|
|    min|               17|           ?|          12285.0|        10th|              1.0|
Divorced|                ?|     Husband|Amer-Indian-Eskimo| Female|              0.0|
0.0|              1.0|          ?|
|    max|               90| Without-pay|        1484705.0|Some-college|             16.0|
Widowed| Transport-moving|        Wife|             White|   Male|          99999.0|
4356.0|             99.0| Yugoslavia|
+-------+-----------------+------------+-----------------+------------+-----------------+----
----------+-----------------+------------+------------------+-------+-----------------+-------
---------+-----------------+-----------+
```

To **view** the **statistics** of a **particular** column we pass that **column name** as **argument** into the **describe()** method.

In [32]:
```
df.describe('capital-gain').show()
```

```
+-------+------------------+
|summary|      capital-gain|
+-------+------------------+
|  count|             32561|
|   mean|1077.6488437087312|
| stddev| 7385.292084840354|
|    min|                 0|
|    max|             99999|
+-------+------------------+
```

FILTER DATA

filter() method is used to **filter** the **orginial dataframe** on the basis of an **condition**.

- Suppose we are to find the number of people who are aged greater than 40.

In [40]:
```
df.filter(df.age > 40).show(5)
```

```
+---+----------------+------+----------+-------------+-------------------+------------------+-
------------+------+-------+------------+------------+--------------+-------------+------+
|age|       workclass|fnlwgt| education|education-num|     marital-status|        occupation|
relationship|  race|    sex|capital-gain|capital-loss|native-country|        class| label|
+---+----------------+------+----------+-------------+-------------------+------------------+-
------------+------+-------+------------+------------+--------------+-------------+------+
| 50|Self-emp-not-inc| 83311| Bachelors|           13| Married-civ-spouse|   Exec-managerial|
Husband| White|   Male|           0|           0|            13| United-States| <=50K|
| 53|         Private|234721|      11th|            7| Married-civ-spouse| Handlers-cleaners|
Husband| Black|   Male|           0|           0|            40| United-States| <=50K|
| 49|         Private|160187|       9th|            5|Married-spouse-a...|     Other-service|
Not-in-family| Black| Female|           0|           0|            16|       Jamaica| <=50K|
| 52|Self-emp-not-inc|209642|   HS-grad|            9| Married-civ-spouse|   Exec-managerial|
Husband| White|   Male|           0|           0|            45| United-States|  >50K|
| 42|         Private|159449| Bachelors|           13| Married-civ-spouse|   Exec-managerial|
Husband| White|   Male|        5178|           0|            40| United-States|  >50K|
+---+----------------+------+----------+-------------+-------------------+------------------+-
------------+------+-------+------------+------------+--------------+-------------+------+
only showing top 5 rows
```

In [39]:
```
df.filter(df.age > 40).count()
```

Out[39]: `13443`

SQL Functionality of SPARK

Let's play with the famous **Flight Delays** dataset and try to answer the **two** main **questions**.

- Print the **Arrival Delay** data for the **flights** that **departs** from **San Fransico (SFO)**.
- List the **top 3 destination airports** (by no. of arriving planes) showing the **count** of **arriving planes** with **Destination code**.

In [2]:
```
nycflight_df = spark.read.csv("flightdelays.csv",inferSchema=True, header = True)
```

In [3]:
```
nycflight_df.show(5)
```

```
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-
------+-----------------+--------------+-------+--------+--------+------+----+--------+------+--
-----+---------+----------------+--------+------------+------------+--------+------------+-----
------------+
|Year|Month|DayofMonth|DayOfWeek|DepTime|CRSDepTime|ArrTime|CRSArrTime|UniqueCarrier|FlightNum|T
ailNum|ActualElapsedTime|CRSElapsedTime|AirTime|ArrDelay|DepDelay|Origin|Dest|Distance|TaxiIn|Ta
xiOut|Cancelled|CancellationCode|Diverted|CarrierDelay|WeatherDelay|NASDelay|SecurityDelay|LateA
ircraftDelay|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-
------+-----------------+--------------+-------+--------+--------+------+----+--------+------+--
-----+---------+----------------+--------+------------+------------+--------+------------+-----
------------+
|2008|    1|         3|        4|   2003|      1955|   2211|      2225|           WN|      335|
N712SW|              128|           150|    116|     -14|       8|   IAD| TPA|     810|     4|
8|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|    754|       735|   1002|      1000|           WN|     3231|
N772SW|              128|           145|    113|       2|      19|   IAD| TPA|     810|     5|
10|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|    628|       620|    804|       750|           WN|      448|
N428WN|               96|            90|     76|      14|       8|   IND| BWI|     515|     3|
17|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|    926|       930|   1054|      1100|           WN|     1746|
N612SW|               88|            90|     78|      -6|      -4|   IND| BWI|     515|     3|
7|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|   1829|      1755|   1959|      1925|           WN|     3920|
N464WN|               90|            90|     77|      34|      34|   IND| BWI|     515|     3|
10|        0|            null|       0|           2|           0|       0|            0|
32|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-
------+-----------------+--------------+-------+--------+--------+------+----+--------+------+--
-----+---------+----------------+--------+------------+------------+--------+------------+-----
------------+
only showing top 5 rows
```

Below code will show how many **observation** we have in our **dataset**.

In [4]:
```
nycflight_df.count()
```

Out[4]: `7009728`

The sql function on a SQLContext enables applications to run SQL queries programmatically and returns the result as a DataFrame.

To run sql queries on our dataset we first to create a temporary table which will accessed by the sqlContext.

We can achieve the above by using **registerTempTable()** method and **pass** the **name of the table** that we will **refer** to when **executing** different **queries**.

In [6]:
```
nycflight_df.registerTempTable('flighttable')
```

In [9]:
```
sqlContext.sql('select * from flighttable').show(5)
```

```
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-
------+-----------------+--------------+-------+--------+--------+------+----+--------+------+--
-----+---------+----------------+--------+------------+------------+--------+------------+-----
------------+
|Year|Month|DayofMonth|DayOfWeek|DepTime|CRSDepTime|ArrTime|CRSArrTime|UniqueCarrier|FlightNum|T
ailNum|ActualElapsedTime|CRSElapsedTime|AirTime|ArrDelay|DepDelay|Origin|Dest|Distance|TaxiIn|Ta
xiOut|Cancelled|CancellationCode|Diverted|CarrierDelay|WeatherDelay|NASDelay|SecurityDelay|LateA
ircraftDelay|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-
------+-----------------+--------------+-------+--------+--------+------+----+--------+------+--
-----+---------+----------------+--------+------------+------------+--------+------------+-----
------------+
|2008|    1|         3|        4|   2003|      1955|   2211|      2225|           WN|      335|
N712SW|              128|           150|    116|     -14|       8|   IAD| TPA|     810|     4|
8|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|    754|       735|   1002|      1000|           WN|     3231|
N772SW|              128|           145|    113|       2|      19|   IAD| TPA|     810|     5|
10|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|    628|       620|    804|       750|           WN|      448|
N428WN|               96|            90|     76|      14|       8|   IND| BWI|     515|     3|
17|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|    926|       930|   1054|      1100|           WN|     1746|
N612SW|               88|            90|     78|      -6|      -4|   IND| BWI|     515|     3|
7|        0|            null|       0|          NA|          NA|      NA|           NA|
NA|
|2008|    1|         3|        4|   1829|      1755|   1959|      1925|           WN|     3920|
N464WN|               90|            90|     77|      34|      34|   IND| BWI|     515|     3|
10|        0|            null|       0|           2|           0|       0|            0|
32|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-
------+-----------------+--------------+-------+--------+--------+------+----+--------+------+--
-----+---------+----------------+--------+------------+------------+--------+------------+-----
------------+
only showing top 5 rows
```

Below **query** will give the **result** to the **first** question.

In [10]:
```
sqlContext.sql("select origin,arrdelay from flighttable where origin = 'SFO' limit 3").show()
```

```
+------+--------+
|origin|arrdelay|
+------+--------+
|   SFO|      69|
|   SFO|      46|
|   SFO|     120|
+------+--------+
```

Below **query** will give the **result** to the **second** question.

In [11]:
```
sqlContext.sql("select Dest, Count(Dest) from flighttable where Cancelled = 0 Group By DEST order by
Count(Dest) desc limit 3").show()
```

```
+----+-----------+
|Dest|count(Dest)|
+----+-----------+
| ATL|     407816|
| ORD|     334358|
| DFW|     273685|
+----+-----------+
```

We noticed that the run time to get the same output using python script on command line shell was more than the time taken to execute the above code.

WHY SPARK??

First we will create a 1gb of text file named dummy.txt using the following command into the cmd:

- echo "This is just a sample line appended to create a big file.. " > dummy.txt for /L %i in (1,1,24) do type dummy.txt >> dummy.txt

Now that we have created the file we will read it using what we learnt.

In [23]:
```
contentRDD =sc.textFile("dummy.txt")
```

Below code, will **filter** the rows which has **length 0**. So all the **lines** having no **character** in it will be **filtered** out and the **remaining lines** will be **selected**.

In [25]:
```
nonempty_lines = contentRDD.filter(lambda x: len(x) > 0)
```

Now, we will use **split()** based on **space** character to get individual words of the whole file.

In [26]:
```
words = nonempty_lines.flatMap(lambda x: x.split(' '))
```

We now have the individual words, so we can now count the occurence of words.

- Apply the **map** method to create the following (individual word, 1) for all the words.
- Apply the **reduceByKey** method to add the values of individual word kind of grouping the words and use sum afterwards.

%%time is used to calcualte the time taken to run the code.

Note: The time taken to do the count is only 199ms for a 1 GB of text file.

In [27]:
```
%%time
wordcount = words.map(lambda x:(x,1)).reduceByKey(lambda x,y: x+y)
```

```
Wall time: 199 ms
```

To **save** the output of the above code to a text file we simply use **saveAsTextFile()**. **coalesce()** is used to tell spark to only create a single output file.

The reason why spark creates multiple output if **coalesce()** is not used is because the code is executed on blocks resulting into multiple outputs. Hence, we specify the **coalesce()** method and passing **1** stating that create only 1 output file and **shuffle=True** will store the output in a random order in the outputted file.

In [41]:
```
wordcount.coalesce(1,shuffle=True).saveAsTextFile("word")
```

Below code is another way to get the output. The output will be displayed here only.

In [31]:
```
for word in wordcount.collect():

     print(word)
```

```
('', 16777216)
('create', 16777216)
('big', 16777216)
('file..', 16777216)
('a', 33554432)
('just', 16777216)
('to', 16777216)
('is', 16777216)
('"', 16777216)
('sample', 16777216)
('appended', 16777216)
('"This', 16777216)
('line', 16777216)
```

Now, lets count the occurence of word without using spark.

In [88]:
```
%%time
file = open("dummy.txt")
wordcount1 = {}

for word in file.read().split():
    if word not in wordcount1:
        wordcount1[word] = 1
    else:
        wordcount1[word] += 1
```

```
Wall time: 1min 16s
```

As we can see the time taken to output the same result is 1 minute and 16s which is way more than 199ms.

In [89]:
```
wordcount1
```

Out[89]:
```
{'"This': 16777216,
 'is': 16777216,
 'just': 16777216,
 'a': 33554432,
 'sample': 16777216,
 'line': 16777216,
 'appended': 16777216,
 'to': 16777216,
 'create': 16777216,
 'big': 16777216,
 'file..': 16777216,
 '"': 16777216}
```

REFERENCES

To learn different terms such as **SparkContext**, **RDDs**, **Transformation/Actions** and using methods like **show()**, **groupBy()**, etc. we have used the **guru99** website.

To learn the **word count** example we have used the **geoinsyssoft's** website.