Introduction into Big Data analytics

Lecture 8 – Apache Spark

Janusz Szwabiński

Outlook:

1. In-memory computation/processing/analytics
2. Why Spark?
3. Installation
4. Data model
5. Architecture
6. First steps with Spark

Further reading:

- https://www.datacamp.com/community/tutorials/apache-spark-python (https://www.datacamp.com/community/tutorials/apache-spark-python)
- W. Feng, *Learning Apache Spark with Python*
- https://mingchen0919.github.io/learning-apache-spark/ (https://mingchen0919.github.io/learning-apache-spark/)

In-memory computation/processing/analytics

- processing of data stored in an in-memory database
- advantages:
  - no slow I/O necessary
  - fast response times
  - real-time analysis
  - faster reporting and decision making
  - short implementation times compared to traditional BI tools
- increasing popularity due to:
  - cheaper and higher performing hardware
  - multicore-architecture
  - advent of 64-bit operating systems
    - theoretically, up to 16 EB memory possible
    - in modern AMD64 architectures, memory limited to 4 PB (52-bit)
  - NAND flash memory
    - cheaper scaling of memory to many Terabytes
- disadvantages:
  - data must fit in the memory of the distributed system
  - additional persistency (with asynchronous flushing) usually required
    - goal is to delegate the operations conducted with the in-memory solution into a backend database
  - fault-tolerance is mandatory
- solutions:
  - BI: SAP Hana
  - Big data: Apache Spark, Apache Flink

Why Spark?

Lightning-fast unified analytics engine

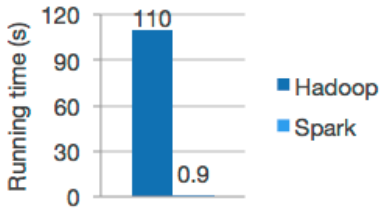
Download | Libraries | Documentation | Examples | Community | Developers

Apache Spark™ is a unified analytics engine for large-scale data processing.

Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.



Logistic regression in Hadoop and Spark

Ease of Use

Write applications quickly in Java, Scala, Python, R, and SQL.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python, R, and SQL shells.

```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API
Read JSON files with automatic schema inference

Spark overview

- https://spark.apache.org/ (https://spark.apache.org/)
- a unified analytics in-memory engine for large-scale data processing
  - loads data from HDFS, Cassandra, HBase
  - resource management via YARN, Mesos, Spark, Amazon EC2
  - it can use Hadoop but also works **standalone**!
- task scheduling and monitoring
- rich APIs:
  - APIs for Java, Scala, Python, R
  - Thrift JDBC/ODBC server for SQL
  - high-level domain-specific tools/languages
    - advanced APIs simplify typical computation tasks
- interactive shells with tight integration
  - `spark-shell`: Scala (object-oriented functional language running on JVM)
  - `pyspark`: Python
  - `sparkR`: R (basic support)
- execution in either **local** (single node) or **cluster** mode

Scala or Python?

- Spark performance and concurency $\longrightarrow$ Scala
  - Scala make it easy to write clean and performant async code
  - Python's user defined functions less efficient that their Scala equivalent
  - avoid passing data between data frames and RDDs while working with Python (serialization and deserialization of data is expensive)
- learning Spark $\longrightarrow$ Python
  - Python is less verbose and more readable than Scala
- type safety $\longrightarrow$ Scala
  - Python is a good choice when you're doing smaller ad hoc experiments
  - Scala is great when you're working on bigger projects in production
  - a statically typed language like Scala is much easier and hassle-free when you're refactoring
- advanced features $\longrightarrow$ Python
  - Python provides the user with a lot of great tools for machine learning and natural language processing

Installation

Ubuntu and Mac OS

Prerequisites

- Java JDK (if not already install, check the Hadoop installation notes)

In [1]:

```
!java -version
```

```
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.
04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

- install Scala

```
sudo apt-get install scala #Ubuntu
```

- install `py4j` for Python-Java integration:

```
sudo pip3 install py4j
```

Installation

- download the package from https://spark.apache.org/downloads.html (https://spark.apache.org/downloads.html)
- untar the package and move it to a directory of your choice
- test Spark with

```
$SPARK_HOME/bin/pyspark
```

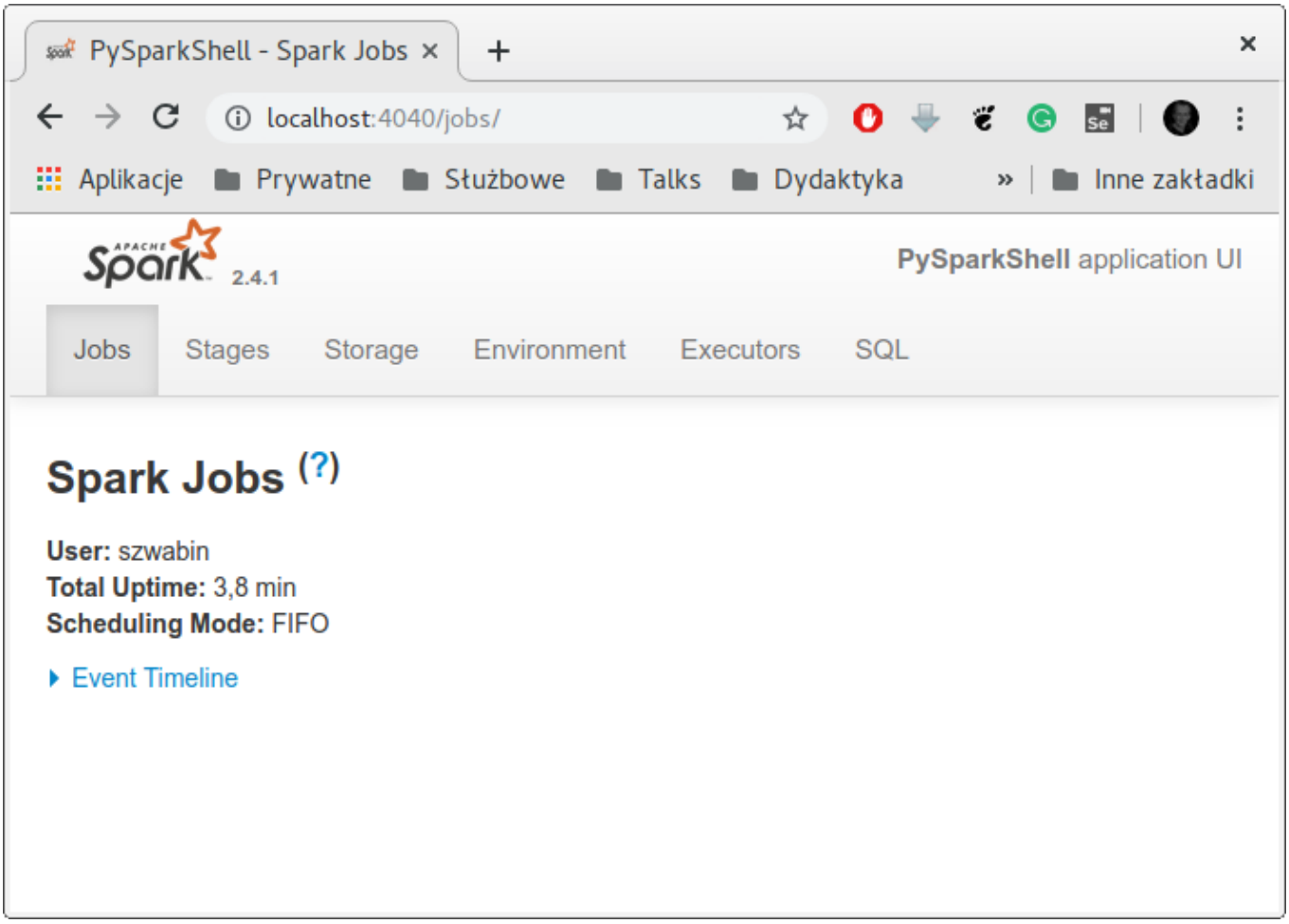
where $SPARK_HOME stands for the installation directory.

You should see something like

```
Python 2.7.12 (default, Nov 12 2018, 14:36:49)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
19/04/11 11:38:36 WARN Utils: Your hostname, enterprise resolves to a loo
pback address: 127.0.1.1; using 10.111.2.9 instead (on interface enp0s31f
6)
19/04/11 11:38:36 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to a
nother address
19/04/11 11:38:36 WARN NativeCodeLoader: Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.prop
erties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use set
LogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Python version 2.7.12 (default, Nov 12 2018 14:36:49)
SparkSession available as 'spark'.
>>>
```

The application UI is available at `localhost:4040`:



Configuration (to work with Python 3.X)

Add the following lines to your `~/.bashrc` (on Ubuntu) or `~/.bash_profile` (on Mac OS) file:

```
# add for spark
export SPARK_HOME=/home/szwabin/Tools/spark
export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_PYTHON=python3
```

Remember to source the file with:

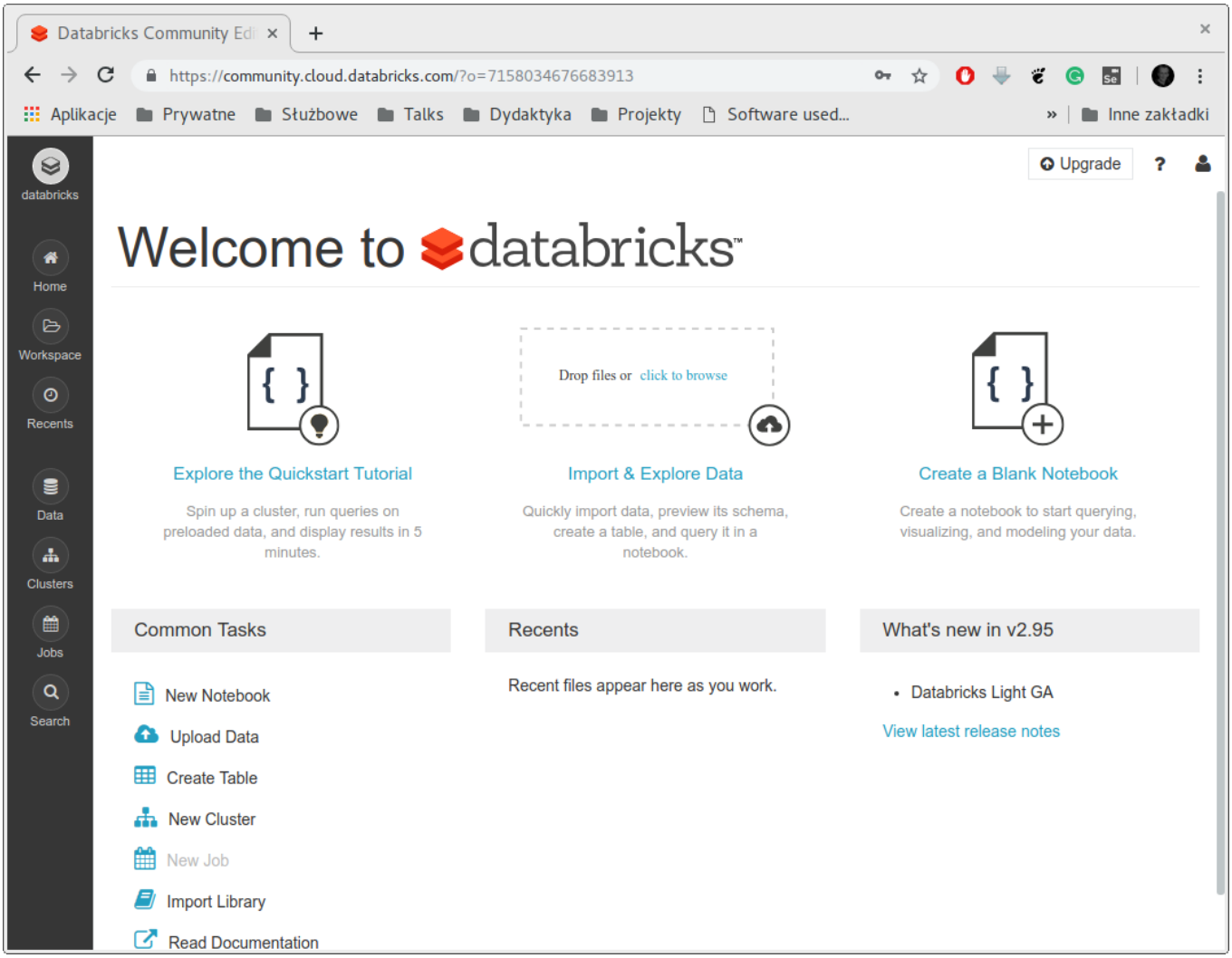
```
source ~/.bashrc #Ubuntu
source ~/.bash_profile #Mac
```

Windows

If you still insist to use Windows as your big data platform, visit http://deelesh.github.io/pyspark-windows.html (http://deelesh.github.io/pyspark-windows.html)

Do I really have to install it?

- no :)
- Databricks Community Cloud (https://databricks.com/try-databricks (https://databricks.com/try-databricks))
  - for students and educational institutions
  - single cluster limited to 6GB and no worker nodes
  - basic notebook without collaboration
  - limited to 3 max users
  - public environment to share your work



Data model

Resilient Distributed Datasets (RDDs)

- the building blocks of Spark
- the original API that Spark exposed
- named collections of elements distributed in partitions, for instance $X = [1, 2, 3, 4 \ldots, 1000]$ distributed in 4 partitions:

- immutable, i.e. it cannot be changed
- typically a list or a map (key-value pairs)
- three main characteristics:
  - compile-time type safe (they have a type!)
  - lazy
  - based on the Scala collections API
- durability of data
  - RDDs live until the SparkContext is terminated
  - to keep them, they need to be persisted (e.g., to HDFS)
- **fault-tolerance** is provided by re-computing data (if an error occurs)
- problems:
  - easy to build inefficient transformation chains
  - slow with non-JVM languages such as Python
  - transformation chains are not very readable
- best practices:
  - **do not** call `collect()` on large RDDs
    - it drags data back into your applications from the nodes
    - each RDD element will be copied onto the single driver program, which will run out of memory and crash
    - if you really need to take a look at the complete data, write out the RDD to files or export it to a database that is large enough to keep the data
  - reduce your RDD before joining
    - to avoid sending too much data over the network that you'll throw away after the join
  - avoid `groupByKey()` on large RDDs
    - on big data sets, `reduceByKey()`, `combineByKey()` or `foldByKey()` are more useful
    - when using `groupByKey()`, all key-value pairs are shuffled around in the cluster and a lot of unnecessary data is being transferred over the network
    - if more data is shuffled onto a single machine than can fit in memory, the data will be spilled to disk
    - this heavily impacts the performance of the Spark job.
    - in `reduceByKey()`, the pairs with the same key are already combined before the data is shuffled (less data sent)
  - broadcast variables
    - it can reduce the cost of launching a job over the cluster
  - avoid `flatmap()`, `join()` and `groupBy()` pattern

    - when you have two datasets that are grouped by key and you want to join them, but still keep them grouped, use `cogroup()` instead of the above pattern

DataFrames

- a high level API providing a representation of data
- built on top of RDDs
- it allows to use a query language to manipulate the data
- a logical plan that represents data and a schema
    - it will be converted to a physical plan for execution
- faster than RDDs
    - custom memory management (project Tungsten)
    - optimized execution plans (Catalyst optimizer)
- consider switching your RDD to a DataFrame especially when working with structured data

Datasets

- downside of DataFrames - no compile-time type safety (code more prone to errors)
- Datasets restory some type safety and allow for lambda functions
- a collection of strongly-typed JVM objects
-
- the second main Spark API, besides the RDD API
    - DataFrame is still there conceptually
    - a collection of strongly-typed JVM objects
- DataSet API includes both DataSets and DataFrames
- DataSet API basically forces you to work with more structured data

When to use which?

- it is generally advised to use DataFrames while working with PySpark
    - similarity to Pandas' dataframe
- use RDDs when you want to perform low-level transformations and actions on your unstructured data
- use RDDs when you want to manipulate the data with functional programming constructs rather than domain specific expressions

Spark vs Python dataframes

- DataFrames have a scheme, with column names and types and logic for rows and columns
- this mimics the implementation of DataFrames in Pandas
- main differences:
    - Spark DataFrames carry the specific optimalization under the hood and can use distributed memory to handle big data
    - Pandas and R dataframes can only run on one computer
- you can convert your Spark DataFrame to a Pandas one when you expect the resulting dataframe to be small enough:

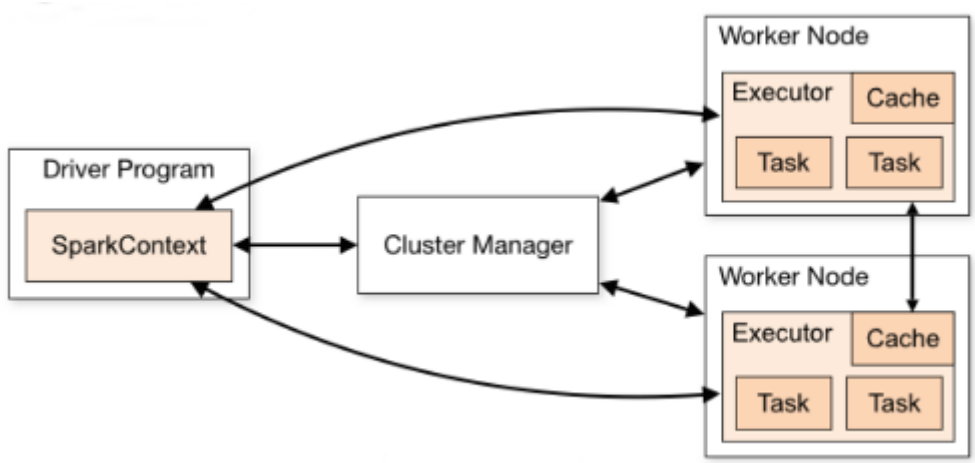
```
df.toPandas()
```

Persistent RDDs

- by default, each transformed RDD may be recomputed each time you run an action on it

- it is however possible to persist an RDD in memory or on disk (replicas across multiple nodes included)

Architecture



- **Driver**
  - a separate process to execute user applications
  - creates/uses **SparkContext**
    - represents the connection to a Spark cluster
    - can be used to create RDDs, accumulators and broadcast variables
- **Cluster** manager: allocates cluster resources and runs executor
  - Yarn
  - Mesos
  - Spark standalone
- **Job**
  - a piece of code which reads some input from HDFS or local file system, performs some computation on the data and writes some output data
- **Task**
  - a unit of work processed by one executor
- **Stage**
  - collection of tasks executing the same code
  - tasks run concurrently
- **Executor**
  - process responsible for executing a task
  - isolates apps (data cannot be shared between them)
  - stores computation results in memory or on disk
  - in local mode only one executor is created

First steps with Spark

Working with PySpark in Jupyter notebook

Jupyter installation

- Jupyter Notebook is a popular application that allows to edit, run and share Python code into a web view
- usually used to test and prototype programs

If you are using Anaconda, Jupyter is already installed on your system. If not, issue the following commands as superuser:

```
python3 -m pip install --upgrade pip
python3 -m pip install jupyter
```

To run the Jupyter notebook, run the following command at the terminal (Mac/Linux) or command prompt (Windows):

```
jupyter notebook
```

PySpark in a Jupyter Notebook

The are 2 ways to get PySpark available in a Jupyter notebook:

- configure PySpark driver to use Jupyter notebook
  - running `pyspark` will automatically open a Jupyter Notebook
  - quicker but specific to Jupyter
- load a regular Jupyter notebook and load PySpark using `findSpark` package
  - works with any IDE

Configuring PySpark driver

Add the following lines to your `~/.bashrc` file (on Linux):

```
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS='notebook'
```

and source the file

```
source ~/.bashrc
```

From now on, the `pyspark` command will start a Jupyter notebook in your web browser.

`findspark` module

To install `findspark`, issue the command

```
pip3 install findspark
```

Then, do the following in a regular Jupyter notebook (or within your favourite IDE):

In [1]:

```
import findspark
findspark.init()

import pyspark
import random

sc = pyspark.SparkContext(appName="Pi")
num_samples = 100000000

def inside(p):
  x, y = random.random(), random.random()
  return x*x + y*y < 1

count = sc.parallelize(range(0, num_samples)).filter(inside).count()
pi = 4 * count / num_samples

print(pi)
sc.stop()
```

```
3.1413712
```

RDDs and DataFrames

`parallelize()` function:

In [2]:

```
sc = pyspark.SparkContext(appName="RDD example")
```

In [3]:

```
rdd = sc.parallelize([(1, 2, 3, 'a b c'),
                      (4, 5, 6, 'd e f'),
                      (7, 8, 9, 'g h i')])
```

In [4]:

```
rdd.collect()
```

Out[4]:

```
[(1, 2, 3, 'a b c'), (4, 5, 6, 'd e f'), (7, 8, 9, 'g h i')]
```

It is possible to convert the RDD into a dataframe:

In [5]:

```
from pyspark.sql import SparkSession

spark = SparkSession(sc)
rdd.toDF().show()
```

```
+---+---+---+-----+
| _1| _2| _3|   _4|
+---+---+---+-----+
|  1|  2|  3|a b c|
|  4|  5|  6|d e f|
|  7|  8|  9|g h i|
+---+---+---+-----+
```

Creation of a dataframe directly:

In [6]:

```
employee = spark.createDataFrame([
('1', 'Joe','70000', '1'),
('2', 'Henry', '80000', '2'),
('3', 'Sam','60000', '2'),
('4', 'Max','90000', '1')],
['Id', 'Name', 'Sallary','DepartmentId']
)
```

In [7]:

```
employee
```

Out[7]:

```
DataFrame[Id: string, Name: string, Sallary: string, DepartmentId:
string]
```

In [8]:

```
employee.show()
```

```
+---+-----+-------+------------+
| Id| Name|Sallary|DepartmentId|
+---+-----+-------+------------+
|  1|  Joe|  70000|           1|
|  2|Henry|  80000|           2|
|  3|  Sam|  60000|           2|
|  4|  Max|  90000|           1|
+---+-----+-------+------------+
```

Reading data from file:

In [9]:

```
cars = sc.textFile('mtcars.csv')
```

In [10]:

```
cars.take(5)
```

Out[10]:

```
['model,mpg,cyl,disp,hp,drat,wt,qsec,vs,am,gear,carb',
 'Mazda RX4,21,6,160,110,3.9,2.62,16.46,0,1,4,4',
 'Mazda RX4 Wag,21,6,160,110,3.9,2.875,17.02,0,1,4,4',
 'Datsun 710,22.8,4,108,93,3.85,2.32,18.61,1,1,4,1',
 'Hornet 4 Drive,21.4,6,258,110,3.08,3.215,19.44,1,0,3,1']
```

- each line of text file becomes an element in the resulting RDD
- usually, a map function will be required to transform elements of RDD into a form suitable for data analysis

In [11]:

```
#save the first row to a variable
header = cars.map(lambda x: x.split(',')).filter(lambda x: x[1] == 'mpg').collec
t()[0]
header
```

Out[11]:

```
['model',
 'mpg',
 'cyl',
 'disp',
 'hp',
 'drat',
 'wt',
 'qsec',
 'vs',
 'am',
 'gear',
 'carb']
```

In [12]:

```
#save the rest to a new RDD
rdd = cars.map(lambda x: x.split(',')).filter(lambda x: x[1] != 'mpg')
rdd.take(2)
```

Out[12]:

```
[['Mazda RX4',
  '21',
  '6',
  '160',
  '110',
  '3.9',
  '2.62',
  '16.46',
  '0',
  '1',
  '4',
  '4'],
 ['Mazda RX4 Wag',
  '21',
  '6',
  '160',
  '110',
  '3.9',
  '2.875',
  '17.02',
  '0',
  '1',
  '4',
  '4']]
```

In [13]:

```
#convert RDD elements to Row objects
from pyspark.sql import Row

def list_to_row(keys, values):
    row_dict = dict(zip(keys, values))
    return Row(**row_dict)
```

In [14]:

```
rdd_rows = rdd.map(lambda x: list_to_row(header, x))
rdd_rows.take(3)
```

Out[14]:

```
[Row(am='1', carb='4', cyl='6', disp='160', drat='3.9', gear='4', h
p='110', model='Mazda RX4', mpg='21', qsec='16.46', vs='0', wt='2.6
2'),
 Row(am='1', carb='4', cyl='6', disp='160', drat='3.9', gear='4', h
p='110', model='Mazda RX4 Wag', mpg='21', qsec='17.02', vs='0', wt
='2.875'),
 Row(am='1', carb='1', cyl='4', disp='108', drat='3.85', gear='4',
hp='93', model='Datsun 710', mpg='22.8', qsec='18.61', vs='1', wt
='2.32')]
```

In [15]:

```
#now we can convert to a dataframe
df = spark.createDataFrame(rdd_rows)
df.show(5)
```

```
+---+----+---+----+----+----+---+----------------+----+-----+---+-
----+
| am|carb|cyl|disp|drat|gear| hp|           model| mpg| qsec| vs|
wt|
+---+----+---+----+----+----+---+----------------+----+-----+---+-
----+
|  1|   4|  6| 160| 3.9|   4|110|       Mazda RX4|  21|16.46|  0|
2.62|
|  1|   4|  6| 160| 3.9|   4|110|   Mazda RX4 Wag|  21|17.02|  0|
2.875|
|  1|   1|  4| 108|3.85|   4| 93|      Datsun 710|22.8|18.61|  1|
2.32|
|  0|   1|  6| 258|3.08|   3|110|  Hornet 4 Drive|21.4|19.44|  1|
3.215|
|  0|   2|  8| 360|3.15|   3|175|Hornet Sportabout|18.7|17.02|  0|
3.44|
+---+----+---+----+----+----+---+----------------+----+-----+---+-
----+
only showing top 5 rows
```

Reading data from csv:

In [16]:

```
mtcars = spark.read.csv(path='mtcars.csv',
                        sep=',',
                        encoding='UTF-8',
                        comment=None,
                        header=True,
                        inferSchema=True)
mtcars.show(n=5, truncate=False)
```

```
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
|model            |mpg |cyl|disp |hp |drat|wt   |qsec |vs |am |gear
|carb|
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
|Mazda RX4        |21.0|6  |160.0|110|3.9 |2.62 |16.46|0  |1  |4
|4   |
|Mazda RX4 Wag    |21.0|6  |160.0|110|3.9 |2.875|17.02|0  |1  |4
|4   |
|Datsun 710       |22.8|4  |108.0|93 |3.85|2.32 |18.61|1  |1  |4
|1   |
|Hornet 4 Drive   |21.4|6  |258.0|110|3.08|3.215|19.44|1  |0  |3
|1   |
|Hornet Sportabout|18.7|8  |360.0|175|3.15|3.44 |17.02|0  |0  |3
|2   |
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
only showing top 5 rows
```

From pandas dataframe:

In [18]:

```
import pandas as pd
pdf = pd.DataFrame({'x': [[1,2,3], [4,5,6]],'y': [['a','b','c'], ['e','f','g'
]]})
pdf
```

Out[18]:

	x	y
0	[1, 2, 3]	[a, b, c]
1	[4, 5, 6]	[e, f, g]

In [19]:

```
df = spark.createDataFrame(pdf)
df.show()
```

```
+---------+---------+
|        x|        y|
+---------+---------+
|[1, 2, 3]|[a, b, c]|
|[4, 5, 6]|[e, f, g]|
+---------+---------+
```

From a list:

In [20]:

```
#each element in the list becomes a row in the dataframe
my_list = [['a', 1], ['b', 2]]
df = spark.createDataFrame(my_list, ['letter', 'number'])
df.show()
```

```
+------+------+
|letter|number|
+------+------+
|     a|     1|
|     b|     2|
+------+------+
```

In [21]:

```
df.dtypes
```

Out[21]:

```
[('letter', 'string'), ('number', 'bigint')]
```

Conversion from dataframe to RDD:

In [22]:

```
df.rdd.collect()
```

Out[22]:

```
[Row(letter='a', number=1), Row(letter='b', number=2)]
```

Merge and split columns

In [23]:

```
mtcars.show(5)
```

```
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
|            model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear
|carb|
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
|        Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4
|   4|
|    Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4
|   4|
|       Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4
|   1|
|   Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|  0|   3
|   1|
|Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|  0|   3
|   2|
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
only showing top 5 rows
```

In [24]:

```
#We convert DataFrame to RDD and then apply the map function
#to merge values and convert elements to Row objects
from pyspark.sql import Row
mtcars_rdd = mtcars.rdd.map(lambda x: Row(model=x[0], values=x[1:]))
mtcars_rdd.take(5)
```

Out[24]:

```
[Row(model='Mazda RX4', values=(21.0, 6, 160.0, 110, 3.9, 2.62, 16.
46, 0, 1, 4, 4)),
 Row(model='Mazda RX4 Wag', values=(21.0, 6, 160.0, 110, 3.9, 2.87
5, 17.02, 0, 1, 4, 4)),
 Row(model='Datsun 710', values=(22.8, 4, 108.0, 93, 3.85, 2.32, 1
8.61, 1, 1, 4, 1)),
 Row(model='Hornet 4 Drive', values=(21.4, 6, 258.0, 110, 3.08, 3.2
15, 19.44, 1, 0, 3, 1)),
 Row(model='Hornet Sportabout', values=(18.7, 8, 360.0, 175, 3.15,
3.44, 17.02, 0, 0, 3, 2))]
```

In [25]:

```
#Create a new dataframe
mtcars_df = spark.createDataFrame(mtcars_rdd)
mtcars_df.show(5, truncate=False)
```

```
+-----------------+-----------------------------------------------
-----+
|model            |values
|
+-----------------+-----------------------------------------------
-----+
|Mazda RX4        |[21.0, 6, 160.0, 110, 3.9, 2.62, 16.46, 0, 1, 4,
4]  |
|Mazda RX4 Wag    |[21.0, 6, 160.0, 110, 3.9, 2.875, 17.02, 0, 1,
4, 4] |
|Datsun 710       |[22.8, 4, 108.0, 93, 3.85, 2.32, 18.61, 1, 1, 4,
1]  |
|Hornet 4 Drive   |[21.4, 6, 258.0, 110, 3.08, 3.215, 19.44, 1, 0,
3, 1]|
|Hornet Sportabout|[18.7, 8, 360.0, 175, 3.15, 3.44, 17.02, 0, 0,
3, 2] |
+-----------------+-----------------------------------------------
-----+
only showing top 5 rows
```

In [26]:

```
#Now we want to split 'values' column into two ones
mtcars_rdd_2 = mtcars_df.rdd.map(lambda x: Row(model=x[0], x1=x[1][:5], x2=x[1][
5:]))
# convert RDD back to DataFrame
mtcars_df_2 = spark.createDataFrame(mtcars_rdd_2)
mtcars_df_2.show(5, truncate=False)
```

```
+-----------------+---------------------------+-------------------
------+
|model            |x1                         |x2
|
+-----------------+---------------------------+-------------------
------+
|Mazda RX4        |[21.0, 6, 160.0, 110, 3.9] |[2.62, 16.46, 0, 1,
4, 4] |
|Mazda RX4 Wag    |[21.0, 6, 160.0, 110, 3.9] |[2.875, 17.02, 0, 1,
4, 4]|
|Datsun 710       |[22.8, 4, 108.0, 93, 3.85] |[2.32, 18.61, 1, 1,
4, 1] |
|Hornet 4 Drive   |[21.4, 6, 258.0, 110, 3.08]|[3.215, 19.44, 1, 0,
3, 1]|
|Hornet Sportabout|[18.7, 8, 360.0, 175, 3.15]|[3.44, 17.02, 0, 0,
3, 2] |
+-----------------+---------------------------+-------------------
------+
only showing top 5 rows
```

Map functions

- `map()`
  - applies a function to each element of an RDD
  - each element has to be a valid input to the function
  - the function outputs as the elements of the new RDD
- `mapValues()`
  - requirement: each element in the RDD has a key/value pair structure
  - it applies a function to each of the element values
  - the element keys remain unchanged
- `flatMap()`
  - applies a function to each elements of an RDD and then flattens the result
- `flatMapValues()`
  - requirement: each element in the RDD has a key/value pair structure
  - it applies a function to each element value of the RDD object and then flattens the results

map()

In [27]:

```
map_exp_rdd = sc.textFile('mtcars.csv')
map_exp_rdd.take(4)
```

Out[27]:

```
['model,mpg,cyl,disp,hp,drat,wt,qsec,vs,am,gear,carb',
 'Mazda RX4,21,6,160,110,3.9,2.62,16.46,0,1,4,4',
 'Mazda RX4 Wag,21,6,160,110,3.9,2.875,17.02,0,1,4,4',
 'Datsun 710,22.8,4,108,93,3.85,2.32,18.61,1,1,4,1']
```

In [28]:

```
# split auto model from other feature values
map_exp_rdd_1 = map_exp_rdd.map(lambda x: x.split(',')).map(lambda x: (x[0], x[1
:]))
map_exp_rdd_1.take(4)
```

Out[28]:

```
[('model',
  ['mpg',
   'cyl',
   'disp',
   'hp',
   'drat',
   'wt',
   'qsec',
   'vs',
   'am',
   'gear',
   'carb']),
 ('Mazda RX4',
  ['21', '6', '160', '110', '3.9', '2.62', '16.46', '0', '1', '4',
'4']),
 ('Mazda RX4 Wag',
  ['21', '6', '160', '110', '3.9', '2.875', '17.02', '0', '1', '4',
'4']),
 ('Datsun 710',
  ['22.8', '4', '108', '93', '3.85', '2.32', '18.61', '1', '1',
'4', '1'])]
```

In [29]:

```
# remove the header row
header = map_exp_rdd_1.first()
# the filter method apply a function to each element
#  - the function output is a boolean value (TRUE or FALSE)
#  - elements that have output TRUE will be kept
map_exp_rdd_2 = map_exp_rdd_1.filter(lambda x: x != header)
map_exp_rdd_2.take(4)
```

Out[29]:

```
[('Mazda RX4',
  ['21', '6', '160', '110', '3.9', '2.62', '16.46', '0', '1', '4',
'4']),
 ('Mazda RX4 Wag',
  ['21', '6', '160', '110', '3.9', '2.875', '17.02', '0', '1', '4',
'4']),
 ('Datsun 710',
  ['22.8', '4', '108', '93', '3.85', '2.32', '18.61', '1', '1',
'4', '1']),
 ('Hornet 4 Drive',
  ['21.4', '6', '258', '110', '3.08', '3.215', '19.44', '1', '0',
'3', '1'])]
```

In [30]:

```
# convert string values to numeric values
map_exp_rdd_3 = map_exp_rdd_2.map(lambda x: (x[0], list(map(float, x[1]))))
map_exp_rdd_3.take(4)
```

Out[30]:

```
[('Mazda RX4',
  [21.0, 6.0, 160.0, 110.0, 3.9, 2.62, 16.46, 0.0, 1.0, 4.0, 4.0]),
 ('Mazda RX4 Wag',
  [21.0, 6.0, 160.0, 110.0, 3.9, 2.875, 17.02, 0.0, 1.0, 4.0, 4.
0]),
 ('Datsun 710',
  [22.8, 4.0, 108.0, 93.0, 3.85, 2.32, 18.61, 1.0, 1.0, 4.0, 1.0]),
 ('Hornet 4 Drive',
  [21.4, 6.0, 258.0, 110.0, 3.08, 3.215, 19.44, 1.0, 0.0, 3.0, 1.
0])]
```

mapValues()

In [31]:

```
mapValues_exp_rdd = map_exp_rdd_3
mapValues_exp_rdd.take(4)
```

Out[31]:

```
[('Mazda RX4',
  [21.0, 6.0, 160.0, 110.0, 3.9, 2.62, 16.46, 0.0, 1.0, 4.0, 4.0]),
 ('Mazda RX4 Wag',
  [21.0, 6.0, 160.0, 110.0, 3.9, 2.875, 17.02, 0.0, 1.0, 4.0, 4.
0]),
 ('Datsun 710',
  [22.8, 4.0, 108.0, 93.0, 3.85, 2.32, 18.61, 1.0, 1.0, 4.0, 1.0]),
 ('Hornet 4 Drive',
  [21.4, 6.0, 258.0, 110.0, 3.08, 3.215, 19.44, 1.0, 0.0, 3.0, 1.
0])]
```

In [32]:

```
import numpy as np
mapValues_exp_rdd_1 = mapValues_exp_rdd.mapValues(lambda x: np.mean(x))
mapValues_exp_rdd_1.take(4)
```

Out[32]:

```
[('Mazda RX4', 29.90727272727273),
 ('Mazda RX4 Wag', 29.98136363636364),
 ('Datsun 710', 23.59818181818182),
 ('Hornet 4 Drive', 38.73954545454546)]
```

flatMap()

In [33]:

```
x = [('a', 'b', 'c'), ('a', 'a'), ('c', 'c', 'c', 'd')]
flatMap_exp_rdd = sc.parallelize(x)
flatMap_exp_rdd.collect()
```

Out[33]:

```
[('a', 'b', 'c'), ('a', 'a'), ('c', 'c', 'c', 'd')]
```

In [34]:

```
flatMap_exp_rdd_1 = flatMap_exp_rdd.flatMap(lambda x: x)
flatMap_exp_rdd_1.collect()
```

Out[34]:

```
['a', 'b', 'c', 'a', 'a', 'c', 'c', 'c', 'd']
```

flatMapValues()

In [35]:

```
# example data
my_data = [
    [1, (23, 28, 32)],
    [2, (18, 29, 31)],
    [3, (34, 21, 18)]
]
for i in my_data:
        print(*i)
```

```
1 (23, 28, 32)
2 (18, 29, 31)
3 (34, 21, 18)
```

In [36]:

```
flatMapValues_exp_rdd = sc.parallelize(my_data)
flatMapValues_exp_rdd.collect()
```

Out[36]:

```
[[1, (23, 28, 32)], [2, (18, 29, 31)], [3, (34, 21, 18)]]
```

In [37]:

```
# merge A,B,and C columns into one column and add the type column
flatMapValues_exp_rdd_1 = flatMapValues_exp_rdd.flatMapValues(lambda x: list(zip
(list('ABC'), x)))
flatMapValues_exp_rdd_1.collect()
```

Out[37]:

```
[(1, ('A', 23)),
 (1, ('B', 28)),
 (1, ('C', 32)),
 (2, ('A', 18)),
 (2, ('B', 29)),
 (2, ('C', 31)),
 (3, ('A', 34)),
 (3, ('B', 21)),
 (3, ('C', 18))]
```

In [38]:

```
# unpack the element values
flatMapValues_exp_rdd_2 = flatMapValues_exp_rdd_1.map(lambda x: [x[0]] + list(x[
1]) )
flatMapValues_exp_rdd_2.collect()
```

Out[38]:

```
[[1, 'A', 23],
 [1, 'B', 28],
 [1, 'C', 32],
 [2, 'A', 18],
 [2, 'B', 29],
 [2, 'C', 31],
 [3, 'A', 34],
 [3, 'B', 21],
 [3, 'C', 18]]
```

Aggregate functions

`aggregate(zeroValue, seqOp, combOp)`

- `zeroValue`
  - like a data container
  - its structure should match with the data structure of the values returned by from the `seqOp` function
- `seqOp`
  - two argument function
  - the first argument is the `zeroValue`
  - the second argument is an element from an RDD
  - `zeroValue` gets updated with the returned value after every run
- `combOp`
  - two argument function
  - the first argument is the final `zeroValue` from one partition
  - the other argument is another final `zeroValue` from another partition

In [39]:

```
#we want to calculate the total sum of squares for mpg and disp in data set mtcars
mtcars_df = mtcars.select(['mpg','disp'])
mtcars_df.take(5)
```

Out[39]:

```
[Row(mpg=21.0, disp=160.0),
 Row(mpg=21.0, disp=160.0),
 Row(mpg=22.8, disp=108.0),
 Row(mpg=21.4, disp=258.0),
 Row(mpg=18.7, disp=360.0)]
```

In [40]:

```
#calculate averages
mpg_mean = mtcars_df.select('mpg').rdd.map(lambda x: x[0]).mean()
disp_mean = mtcars_df.select('disp').rdd.map(lambda x: x[0]).mean()
print('mpg mean = ', mpg_mean, '; '
      'disp mean = ', disp_mean)
```

```
mpg mean =  20.090625000000003 ; disp mean =  230.721875
```

In [41]:

```
#prepare for aggregation
zeroValue = (0,0)
# z below refers to zeroValue,
#x refers to an element in an RDD partition
seqOp = lambda z, x: (z[0] + (x[0] - mpg_mean)**2, z[1] + (x[1] - disp_mean)**2)
combOp = lambda px, py: ( px[0] + py[0], px[1] + py[1] )
```

In [42]:

```
res = mtcars_df.rdd.aggregate(zeroValue, seqOp, combOp)
```

In [43]:

```
res
```

Out[43]:

```
(1126.0471874999998, 476184.7946875)
```

aggregateByKey(zeroValue, seqOp, combOp)

- similar to `aggregate`, but merges results by key

Continuous to categorical variables

- `pyspark.ml.feature.Binarizer` - split a column of continuous features given a threshold
- `pyspark.ml.feature.Bucktizer` - split a column of continuous features into categories given several breaking points (with n+1 split points, there are n categories aka buckets)

In [44]:

```
#create some data
import numpy as np
import pandas as pd
np.random.seed(seed=1234)
pdf = pd.DataFrame({
        'x1': np.random.randn(10),
        'x2': np.random.rand(10)*10
    })
np.random.seed(seed=None)
df = spark.createDataFrame(pdf)
df.show()
```

```
+--------------------+------------------+
|                  x1|                x2|
+--------------------+------------------+
| 0.47143516373249306| 6.834629351721363|
| -1.1909756947064645| 7.127020269829002|
|  1.4327069684260973|3.7025075479039495|
| -0.3126518960917129| 5.611961860656249|
| -0.7205887333650116| 5.030831653078097|
|  0.8871629403077386|0.1376844959068224|
|  0.8595884137174165| 7.728266216123741|
| -0.6365235044173491| 8.826411906361166|
|0.015696372114428918| 3.648859839013723|
| -2.2426849541854055| 6.153961784334937|
+--------------------+------------------+
```

In [45]:

```
#binarize the column x1 and bucketize the column x2
from pyspark.ml.feature import Binarizer, Bucketizer
# threshold = 0 for binarizer
binarizer = Binarizer(threshold=0, inputCol='x1', outputCol='x1_new')
# provide 5 split points to generate 4 buckets
bucketizer = Bucketizer(splits=[0, 2.5, 5, 7.5, 10], inputCol='x2', outputCol='x
2_new')

# pipeline stages
from pyspark.ml import Pipeline
stages = [binarizer, bucketizer]
pipeline = Pipeline(stages=stages)

# fit the pipeline model and transform the data
pipeline.fit(df).transform(df).show()
```

```
+--------------------+------------------+------+------+
|                  x1|                x2|x1_new|x2_new|
+--------------------+------------------+------+------+
| 0.47143516373249306| 6.834629351721363|   1.0|   2.0|
| -1.1909756947064645| 7.127020269829002|   0.0|   2.0|
|  1.4327069684260973|3.7025075479039495|   1.0|   1.0|
| -0.3126518960917129| 5.611961860656249|   0.0|   2.0|
| -0.7205887333650116| 5.030831653078097|   0.0|   2.0|
|  0.8871629403077386|0.1376844959068224|   1.0|   0.0|
|  0.8595884137174165| 7.728266216123741|   1.0|   3.0|
| -0.6365235044173491| 8.826411906361166|   0.0|   3.0|
|0.015696372114428918| 3.648859839013723|   1.0|   1.0|
| -2.2426849541854055| 6.153961784334937|   0.0|   2.0|
+--------------------+------------------+------+------+
```

First data checks

In [46]:

```
titanic = spark.read.csv('titanic.csv', header=True, inferSchema=True)
titanic.show(5)
```

```
+-----------+--------+------+--------------------+------+----+-----
+-----+----------------+-------+-----+--------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp
|Parch|          Ticket|   Fare|Cabin|Embarked|
+-----------+--------+------+--------------------+------+----+-----
+-----+----------------+-------+-----+--------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|22.0|    1
|    0|       A/5 21171|   7.25| null|       S|
|          2|       1|     1|Cumings, Mrs. Joh...|female|38.0|    1
|    0|        PC 17599|71.2833|  C85|       C|
|          3|       1|     3|Heikkinen, Miss. ...|female|26.0|    0
|    0|STON/O2. 3101282|  7.925| null|       S|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|35.0|    1
|    0|          113803|   53.1| C123|       S|
|          5|       0|     3|Allen, Mr. Willia...|  male|35.0|    0
|    0|          373450|   8.05| null|       S|
+-----------+--------+------+--------------------+------+----+-----
+-----+----------------+-------+-----+--------+
only showing top 5 rows
```

In [47]:

```
titanic.printSchema()
```

```
root
 |-- PassengerId: integer (nullable = true)
 |-- Survived: integer (nullable = true)
 |-- Pclass: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- SibSp: integer (nullable = true)
 |-- Parch: integer (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: double (nullable = true)
 |-- Cabin: string (nullable = true)
 |-- Embarked: string (nullable = true)
```

In [48]:

```
#number of variables
len(titanic.columns)
```

Out[48]:

```
12
```

In [49]:

```
#number of observations
titanic.count()
```

Out[49]:

891

In [50]:

```
#summarize columns
def describe_columns(df):
    for i in df.columns:
        print('Column: ' + i)
        titanic.select(i).describe().show()
```

In [51]:

```
describe_columns(titanic)
```

```
Column: PassengerId
+-------+-----------------+
|summary|      PassengerId|
+-------+-----------------+
|  count|              891|
|   mean|            446.0|
| stddev|257.3538420152301|
|    min|                1|
|    max|              891|
+-------+-----------------+

Column: Survived
+-------+-------------------+
|summary|           Survived|
+-------+-------------------+
|  count|                891|
|   mean| 0.3838383838383838|
| stddev|0.4865924542648575 3|
|    min|                  0|
|    max|                  1|
+-------+-------------------+

Column: Pclass
+-------+------------------+
|summary|            Pclass|
+-------+------------------+
|  count|               891|
|   mean| 2.308641975308642|
| stddev|0.8360712409770491|
|    min|                 1|
|    max|                 3|
+-------+------------------+

Column: Name
+-------+--------------------+
|summary|                Name|
+-------+--------------------+
|  count|                 891|
|   mean|                null|
| stddev|                null|
|    min|"Andersson, Mr. A...|
|    max|van Melkebeke, Mr...|
+-------+--------------------+

Column: Sex
+-------+------+
|summary|   Sex|
+-------+------+
|  count|   891|
|   mean|  null|
| stddev|  null|
|    min|female|
|    max|  male|
+-------+------+

Column: Age
+-------+-----------------+
|summary|              Age|
+-------+-----------------+
|  count|              714|
|   mean| 29.69911764705882|
```

```
| stddev|14.526497332334035|
|    min|              0.42|
|    max|              80.0|
+-------+------------------+

Column: SibSp
+-------+------------------+
|summary|             SibSp|
+-------+------------------+
|  count|               891|
|   mean|0.5230078563411896|
| stddev|1.1027434322934315|
|    min|                 0|
|    max|                 8|
+-------+------------------+

Column: Parch
+-------+-------------------+
|summary|              Parch|
+-------+-------------------+
|  count|                891|
|   mean|0.38159371492704824|
| stddev| 0.8060572211299488|
|    min|                  0|
|    max|                  6|
+-------+-------------------+

Column: Ticket
+-------+------------------+
|summary|            Ticket|
+-------+------------------+
|  count|               891|
|   mean|260318.54916792738|
| stddev|471609.26868834975|
|    min|            110152|
|    max|         WE/P 5735|
+-------+------------------+

Column: Fare
+-------+-----------------+
|summary|             Fare|
+-------+-----------------+
|  count|              891|
|   mean| 32.2042079685746|
| stddev|49.69342859718089|
|    min|              0.0|
|    max|         512.3292|
+-------+-----------------+

Column: Cabin
+-------+-----+
|summary|Cabin|
+-------+-----+
|  count|  204|
|   mean| null|
| stddev| null|
|    min|  A10|
|    max|    T|
+-------+-----+

Column: Embarked
```

```
+-------+--------+
|summary|Embarked|
+-------+--------+
|  count|     889|
|   mean|    null|
| stddev|    null|
|    min|       C|
|    max|       S|
+-------+--------+
```

In [52]:

```
#find columns with missing values
def find_missing_values_columns(df):
    nrow = df.count()
    for v in df.columns:
        summary_df = df.select(v).describe()
        v_count = int(summary_df.collect()[0][v])
        if v_count < nrow:
            missing_percentage = (1 - v_count/nrow) * 100
            print("Total observations: " + str(nrow) + "\n"
                 "Total observations of " + v + ": " + str(v_count) + "\n"
                 "Percentage of missing values: " + str(missing_percentage) +
"%" + "\n"
                 "----------------------------")
```

In [53]:

```
find_missing_values_columns(titanic)
```

```
Total observations: 891
Total observations of Age: 714
Percentage of missing values: 19.865319865319865%
----------------------------
Total observations: 891
Total observations of Cabin: 204
Percentage of missing values: 77.1043771043771%
----------------------------
Total observations: 891
Total observations of Embarked: 889
Percentage of missing values: 0.22446689113355678%
----------------------------
```

Subsets of dataframes

In [54]:

```
mtcars.show(5)
```

```
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
|            model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear
|carb|
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
|        Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4
|   4|
|    Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4
|   4|
|       Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4
|   1|
|   Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|  0|   3
|   1|
|Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|  0|   3
|   2|
+-----------------+----+---+-----+---+----+-----+-----+---+---+----
+----+
only showing top 5 rows
```

In [55]:

```
#select rows by index
mtcars.rdd.zipWithIndex().take(5)
```

Out[55]:

```
[(Row(model='Mazda RX4', mpg=21.0, cyl=6, disp=160.0, hp=110, drat=
3.9, wt=2.62, qsec=16.46, vs=0, am=1, gear=4, carb=4),
  0),
 (Row(model='Mazda RX4 Wag', mpg=21.0, cyl=6, disp=160.0, hp=110, d
rat=3.9, wt=2.875, qsec=17.02, vs=0, am=1, gear=4, carb=4),
  1),
 (Row(model='Datsun 710', mpg=22.8, cyl=4, disp=108.0, hp=93, drat=
3.85, wt=2.32, qsec=18.61, vs=1, am=1, gear=4, carb=1),
  2),
 (Row(model='Hornet 4 Drive', mpg=21.4, cyl=6, disp=258.0, hp=110,
drat=3.08, wt=3.215, qsec=19.44, vs=1, am=0, gear=3, carb=1),
  3),
 (Row(model='Hornet Sportabout', mpg=18.7, cyl=8, disp=360.0, hp=17
5, drat=3.15, wt=3.44, qsec=17.02, vs=0, am=0, gear=3, carb=2),
  4)]
```

In [56]:

```
#we can apply the map function to modify the structure of each element
mtcars.rdd.zipWithIndex().map(lambda x: [x[1]] + list(x[0])).take(5)
```

Out[56]:

```
[[0, 'Mazda RX4', 21.0, 6, 160.0, 110, 3.9, 2.62, 16.46, 0, 1, 4,
4],
 [1, 'Mazda RX4 Wag', 21.0, 6, 160.0, 110, 3.9, 2.875, 17.02, 0, 1,
4, 4],
 [2, 'Datsun 710', 22.8, 4, 108.0, 93, 3.85, 2.32, 18.61, 1, 1, 4,
1],
 [3, 'Hornet 4 Drive', 21.4, 6, 258.0, 110, 3.08, 3.215, 19.44, 1,
0, 3, 1],
 [4, 'Hornet Sportabout', 18.7, 8, 360.0, 175, 3.15, 3.44, 17.02,
0, 0, 3, 2]]
```

In [57]:

```
header = ['index'] + mtcars.columns
mtcars_df = mtcars.rdd.zipWithIndex().map(lambda x: [x[1]] + list(x[0])).toDF(he
ader)
```

In [58]:

```
mtcars_df.show(5)
```

```
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
|index|            model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| a
m|gear|carb|
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
|    0|        Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|
1|   4|   4|
|    1|    Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|
1|   4|   4|
|    2|       Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|
1|   4|   1|
|    3|   Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|
0|   3|   1|
|    4|Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|
0|   3|   2|
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
only showing top 5 rows
```

In [59]:

```
#select specific rows
mtcars_df.filter(mtcars_df.index.isin([1,2,4,6,9])).show()
```

```
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
|index|            model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| a
m|gear|carb|
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
|    1|    Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|
1|   4|   4|
|    2|       Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|
1|   4|   1|
|    4|Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|
0|   3|   2|
|    6|       Duster 360|14.3|  8|360.0|245|3.21| 3.57|15.84|  0|
0|   3|   4|
|    9|         Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|
0|   4|   4|
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
```

In [60]:

```
#select rows from a given range
mtcars_df.filter(mtcars_df.index.between(5, 10)).show()
```

```
+-----+----------+----+---+-----+---+----+----+-----+---+---+----+-
---+
|index|     model| mpg|cyl| disp| hp|drat|  wt| qsec| vs| am|gear|c
arb|
+-----+----------+----+---+-----+---+----+----+-----+---+---+----+-
---+
|    5|   Valiant|18.1|  6|225.0|105|2.76|3.46|20.22|  1|  0|   3|
1|
|    6|Duster 360|14.3|  8|360.0|245|3.21|3.57|15.84|  0|  0|   3|
4|
|    7| Merc 240D|24.4|  4|146.7| 62|3.69|3.19| 20.0|  1|  0|   4|
2|
|    8|  Merc 230|22.8|  4|140.8| 95|3.92|3.15| 22.9|  1|  0|   4|
2|
|    9|  Merc 280|19.2|  6|167.6|123|3.92|3.44| 18.3|  1|  0|   4|
4|
|   10| Merc 280C|17.8|  6|167.6|123|3.92|3.44| 18.9|  1|  0|   4|
4|
+-----+----------+----+---+-----+---+----+----+-----+---+---+----+-
---+
```

In [61]:

```
#select rows by a cutoff index
mtcars_df.filter(mtcars_df.index < 9).show()
```

```
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
|index|            model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| a
m|gear|carb|
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
|    0|        Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|
1|   4|   4|
|    1|    Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|
1|   4|   4|
|    2|       Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|
1|   4|   1|
|    3|   Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|
0|   3|   1|
|    4|Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|
0|   3|   2|
|    5|          Valiant|18.1|  6|225.0|105|2.76| 3.46|20.22|  1|
0|   3|   1|
|    6|       Duster 360|14.3|  8|360.0|245|3.21| 3.57|15.84|  0|
0|   3|   4|
|    7|        Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|
0|   4|   2|
|    8|         Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|
0|   4|   2|
+-----+-----------------+----+---+-----+---+----+-----+-----+---+--
-+----+----+
```

In [62]:

```
#select rows by logical criteria
mtcars_df.filter(mtcars_df.cyl == 4).show()
```

```
+-----+--------------+----+---+-----+---+----+-----+-----+---+---+-
---+----+
|index|         model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|g
ear|carb|
+-----+--------------+----+---+-----+---+----+-----+-----+---+---+-
---+----+
|    2|    Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|
4|   1|
|    7|     Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|  0|
4|   2|
|    8|      Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|  0|
4|   2|
|   17|      Fiat 128|32.4|  4| 78.7| 66|4.08|  2.2|19.47|  1|  1|
4|   1|
|   18|   Honda Civic|30.4|  4| 75.7| 52|4.93|1.615|18.52|  1|  1|
4|   2|
|   19|Toyota Corolla|33.9|  4| 71.1| 65|4.22|1.835| 19.9|  1|  1|
4|   1|
|   20| Toyota Corona|21.5|  4|120.1| 97| 3.7|2.465|20.01|  1|  0|
3|   1|
|   25|     Fiat X1-9|27.3|  4| 79.0| 66|4.08|1.935| 18.9|  1|  1|
4|   1|
|   26| Porsche 914-2|26.0|  4|120.3| 91|4.43| 2.14| 16.7|  0|  1|
5|   2|
|   27|  Lotus Europa|30.4|  4| 95.1|113|3.77|1.513| 16.9|  1|  1|
5|   2|
|   31|    Volvo 142E|21.4|  4|121.0|109|4.11| 2.78| 18.6|  1|  1|
4|   2|
+-----+--------------+----+---+-----+---+----+-----+-----+---+---+-
---+----+
```

In [63]:

```
#select column by name
mtcars.select(['hp', 'disp']).show(5)
```

```
+---+-----+
| hp| disp|
+---+-----+
|110|160.0|
|110|160.0|
| 93|108.0|
|110|258.0|
|175|360.0|
+---+-----+
only showing top 5 rows
```

In [64]:

```
#select columns by index
indices = [0,3,4,7]
selected_columns =  [mtcars.columns[index] for index in indices]
selected_columns
```

Out[64]:

```
['model', 'disp', 'hp', 'qsec']
```

In [65]:

```
mtcars.select(selected_columns).show(5)
```

```
+-----------------+-----+---+-----+
|            model| disp| hp| qsec|
+-----------------+-----+---+-----+
|        Mazda RX4|160.0|110|16.46|
|    Mazda RX4 Wag|160.0|110|17.02|
|       Datsun 710|108.0| 93|18.61|
|   Hornet 4 Drive|258.0|110|19.44|
|Hornet Sportabout|360.0|175|17.02|
+-----------------+-----+---+-----+
only showing top 5 rows
```

In [66]:

```
#select columns by pattern
import re
selected_columns = [x for x in mtcars.columns if re.compile('^d').match(x) is no
t None]
selected_columns
```

Out[66]:

```
['disp', 'drat']
```

In [67]:

```
mtcars.select(selected_columns).show(5)
```

```
+-----+----+
| disp|drat|
+-----+----+
|160.0| 3.9|
|160.0| 3.9|
|108.0|3.85|
|258.0|3.08|
|360.0|3.15|
+-----+----+
only showing top 5 rows
```

Column expressions

In [68]:

```
#use dot ('.') to select column from DataFrame
mpg_col = mtcars.mpg
mpg_col
```

Out[68]:

```
Column<b'mpg'>
```

In [69]:

```
mtcars.select(mpg_col * 100).show(5)
```

```
+-----------+
|(mpg * 100)|
+-----------+
|     2100.0|
|     2100.0|
|     2280.0|
|     2140.0|
|     1870.0|
+-----------+
only showing top 5 rows
```

In [70]:

```
mtcars.select(mpg_col +1).show(5)
```

```
+---------+
|(mpg + 1)|
+---------+
|     22.0|
|     22.0|
|     23.8|
|     22.4|
|     19.7|
+---------+
only showing top 5 rows
```

In [71]:

```
mtcars.select(mtcars.gear.isin([2,3])).show(5)
```

```
+----------------+
|(gear IN (2, 3))|
+----------------+
|           false|
|           false|
|           false|
|            true|
|            true|
+----------------+
only showing top 5 rows
```

Boolean column expressions

between()

In [77]:

```
#: true/false if the column value is between given values
mpg_between = mtcars.cyl.between(4,6)
mpg_between
```

Out[77]:

```
Column<b'((cyl >= 4) AND (cyl <= 6))'>
```

In [78]:

```
mtcars.select(mtcars.cyl, mpg_between).show(5)
```

```
+---+--------------------------+
|cyl|((cyl >= 4) AND (cyl <= 6))|
+---+--------------------------+
|  6|                      true|
|  6|                      true|
|  4|                      true|
|  6|                      true|
|  8|                     false|
+---+--------------------------+
only showing top 5 rows
```

contains()

In [79]:

```
#true/false if the column value contains a string
model_contains = mtcars.model.contains('Ho')
model_contains
```

Out[79]:

```
Column<b'contains(model, Ho)'>
```

In [80]:

```
mtcars.select(mtcars.model, model_contains).show(5)
```

```
+-----------------+------------------+
|            model|contains(model, Ho)|
+-----------------+------------------+
|        Mazda RX4|             false|
|    Mazda RX4 Wag|             false|
|       Datsun 710|             false|
|   Hornet 4 Drive|              true|
|Hornet Sportabout|              true|
+-----------------+------------------+
only showing top 5 rows
```

endswith()

In [81]:

```
#true/false if the column value ends with a string
model_endswith = mtcars.model.endswith('t')
model_endswith
```

Out[81]:

```
Column<b'endswith(model, t)'>
```

In [82]:

```
mtcars.select(mtcars.model, model_endswith).show(6)
```

```
+-----------------+------------------+
|            model|endswith(model, t)|
+-----------------+------------------+
|        Mazda RX4|             false|
|    Mazda RX4 Wag|             false|
|       Datsun 710|             false|
|   Hornet 4 Drive|             false|
|Hornet Sportabout|              true|
|          Valiant|              true|
+-----------------+------------------+
only showing top 6 rows
```

isNotNull()

In [83]:

```
#true/false if the column value is not Null
from pyspark.sql import Row
df = spark.createDataFrame([Row(name='Tom', height=80), Row(name='Alice', height
=None)])
df.show()
```

```
+------+-----+
|height| name|
+------+-----+
|    80|  Tom|
|  null|Alice|
+------+-----+
```

In [84]:

```
height_isNotNull = df.height.isNotNull()
height_isNotNull
```

Out[84]:

```
Column<b'(height IS NOT NULL)'>
```

In [85]:

```
df.select(df.height, height_isNotNull).show()
```

```
+------+--------------------+
|height|(height IS NOT NULL)|
+------+--------------------+
|    80|                true|
|  null|               false|
+------+--------------------+
```

isNull()

In [86]:

```
#true/false if the column value is Null
height_isNull = df.height.isNull()
height_isNull
```

Out[86]:

```
Column<b'(height IS NULL)'>
```

In [87]:

```
df.select(df.height, height_isNull).show()
```

```
+------+----------------+
|height|(height IS NULL)|
+------+----------------+
|    80|           false|
|  null|            true|
+------+----------------+
```

isin()

In [88]:

```
#true/false if the column value is contained in the given sequence
carb_isin = mtcars.carb.isin([2, 3])
carb_isin
```

Out[88]:

```
Column<b'(carb IN (2, 3))'>
```

In [89]:

```
mtcars.select(mtcars.carb, carb_isin).show(10)
```

```
+----+----------------+
|carb|(carb IN (2, 3))|
+----+----------------+
|   4|           false|
|   4|           false|
|   1|           false|
|   1|           false|
|   2|            true|
|   1|           false|
|   4|           false|
|   2|            true|
|   2|            true|
|   4|           false|
+----+----------------+
only showing top 10 rows
```

like()

In [90]:

```
#true/false if the column value matches a pattern (similar to SQL's LIKE)
model_like = mtcars.model.like('Ho%')
model_like
```

Out[90]:

```
Column<b'model LIKE Ho%'>
```

In [91]:

```
mtcars.select(mtcars.model, model_like).show(10)
```

```
+-----------------+--------------+
|            model|model LIKE Ho%|
+-----------------+--------------+
|        Mazda RX4|         false|
|    Mazda RX4 Wag|         false|
|       Datsun 710|         false|
|   Hornet 4 Drive|          true|
|Hornet Sportabout|          true|
|          Valiant|         false|
|       Duster 360|         false|
|        Merc 240D|         false|
|         Merc 230|         false|
|         Merc 280|         false|
+-----------------+--------------+
only showing top 10 rows
```

rlike()

In [92]:

```
#true/false if the column value matches a pattern (similar to like, but with reg exp)
model_rlike = mtcars.model.rlike('t$') #'t' at the end of the model
model_rlike
```

Out[92]:

```
Column<b'model RLIKE t$'>
```

In [93]:

```
mtcars.select(mtcars.model, model_rlike).show()
```

```
+------------------+--------------+
|             model|model RLIKE t$|
+------------------+--------------+
|         Mazda RX4|         false|
|     Mazda RX4 Wag|         false|
|        Datsun 710|         false|
|    Hornet 4 Drive|         false|
| Hornet Sportabout|          true|
|           Valiant|          true|
|         Duster 360|         false|
|          Merc 240D|         false|
|           Merc 230|         false|
|           Merc 280|         false|
|          Merc 280C|         false|
|         Merc 450SE|         false|
|         Merc 450SL|         false|
|        Merc 450SLC|         false|
| Cadillac Fleetwood|         false|
|Lincoln Continental|         false|
|  Chrysler Imperial|         false|
|           Fiat 128|         false|
|        Honda Civic|         false|
|    Toyota Corolla|         false|
+------------------+--------------+
only showing top 20 rows
```

startswith()

In [94]:

```
#true/false if the column value starts with a given string
model_startswith = mtcars.model.startswith('Merc')
model_startswith
```

Out[94]:

```
Column<b'startswith(model, Merc)'>
```

In [95]:

```
mtcars.select(mtcars.model, model_startswith).show()
```

```
+------------------+----------------------+
|             model|startswith(model, Merc)|
+------------------+----------------------+
|         Mazda RX4|                 false|
|     Mazda RX4 Wag|                 false|
|        Datsun 710|                 false|
|    Hornet 4 Drive|                 false|
| Hornet Sportabout|                 false|
|           Valiant|                 false|
|         Duster 360|                 false|
|          Merc 240D|                  true|
|           Merc 230|                  true|
|           Merc 280|                  true|
|          Merc 280C|                  true|
|         Merc 450SE|                  true|
|         Merc 450SL|                  true|
|        Merc 450SLC|                  true|
| Cadillac Fleetwood|                 false|
|Lincoln Continental|                 false|
|  Chrysler Imperial|                 false|
|           Fiat 128|                 false|
|        Honda Civic|                 false|
|     Toyota Corolla|                 false|
+------------------+----------------------+
only showing top 20 rows
```

SQL functions

In [96]:

```
from pyspark.sql import functions as F
from pyspark.sql import Row
df = sc.parallelize([Row(x=1), Row(x=-1), Row(x=-2)]).toDF()
df.show()
```

```
+---+
|  x|
+---+
|  1|
| -1|
| -2|
+---+
```

In [97]:

```
#absolute value of column elements
x_abs = F.abs(df.x)
x_abs
```

Out[97]:

```
Column<b'abs(x)'>
```

In [98]:

```
df.select(df.x, x_abs).show()
```

```
+---+------+
|  x|abs(x)|
+---+------+
|  1|     1|
| -1|     1|
| -2|     2|
+---+------+
```

Renaming columns

In [99]:

```
mtcars.show(3)
```

```
+-------------+----+---+-----+---+----+-----+-----+---+---+----+---
-+
|        model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|car
b|
+-------------+----+---+-----+---+----+-----+-----+---+---+----+---
-+
|    Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4|
4|
|Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4|
4|
|   Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4|
1|
+-------------+----+---+-----+---+----+-----+-----+---+---+----+---
-+
only showing top 3 rows
```

In [100]:

```
new_col_names = [ 'x_' + x for x in mtcars.columns]
new_col_names
```

Out[100]:

```
['x_model',
 'x_mpg',
 'x_cyl',
 'x_disp',
 'x_hp',
 'x_drat',
 'x_wt',
 'x_qsec',
 'x_vs',
 'x_am',
 'x_gear',
 'x_carb']
```

In [101]:

```
mtcars2 = mtcars.rdd.toDF(new_col_names)
mtcars2.show(3)
```

```
+-------------+-----+-----+------+----+------+-----+------+----+---
-+------+------+
|      x_model|x_mpg|x_cyl|x_disp|x_hp|x_drat| x_wt|x_qsec|x_vs|x_a
m|x_gear|x_carb|
+-------------+-----+-----+------+----+------+-----+------+----+---
-+------+------+
|    Mazda RX4| 21.0|    6| 160.0| 110|   3.9| 2.62| 16.46|   0|
1|     4|     4|
|Mazda RX4 Wag| 21.0|    6| 160.0| 110|   3.9|2.875| 17.02|   0|
1|     4|     4|
|   Datsun 710| 22.8|    4| 108.0|  93|  3.85| 2.32| 18.61|   1|
1|     4|     1|
+-------------+-----+-----+------+----+------+-----+------+----+---
-+------+------+
only showing top 3 rows
```

Exporting data

We need to coalesce data into one partition before saving to file:

In [102]:

```
from pyspark.sql import DataFrameWriter
mtcars = mtcars.coalesce(numPartitions=1)
mtcars.write.csv('saved-mtcars', header=True)
```

In [103]:

```
!cat saved-mtcars/part-00000-eb6eead5-c1e1-4720-bd8a-4e67eaf95964-c000.csv
```

```
model,mpg,cyl,disp,hp,drat,wt,qsec,vs,am,gear,carb
Mazda RX4,21.0,6,160.0,110,3.9,2.62,16.46,0,1,4,4
Mazda RX4 Wag,21.0,6,160.0,110,3.9,2.875,17.02,0,1,4,4
Datsun 710,22.8,4,108.0,93,3.85,2.32,18.61,1,1,4,1
Hornet 4 Drive,21.4,6,258.0,110,3.08,3.215,19.44,1,0,3,1
Hornet Sportabout,18.7,8,360.0,175,3.15,3.44,17.02,0,0,3,2
Valiant,18.1,6,225.0,105,2.76,3.46,20.22,1,0,3,1
Duster 360,14.3,8,360.0,245,3.21,3.57,15.84,0,0,3,4
Merc 240D,24.4,4,146.7,62,3.69,3.19,20.0,1,0,4,2
Merc 230,22.8,4,140.8,95,3.92,3.15,22.9,1,0,4,2
Merc 280,19.2,6,167.6,123,3.92,3.44,18.3,1,0,4,4
Merc 280C,17.8,6,167.6,123,3.92,3.44,18.9,1,0,4,4
Merc 450SE,16.4,8,275.8,180,3.07,4.07,17.4,0,0,3,3
Merc 450SL,17.3,8,275.8,180,3.07,3.73,17.6,0,0,3,3
Merc 450SLC,15.2,8,275.8,180,3.07,3.78,18.0,0,0,3,3
Cadillac Fleetwood,10.4,8,472.0,205,2.93,5.25,17.98,0,0,3,4
Lincoln Continental,10.4,8,460.0,215,3.0,5.424,17.82,0,0,3,4
Chrysler Imperial,14.7,8,440.0,230,3.23,5.345,17.42,0,0,3,4
Fiat 128,32.4,4,78.7,66,4.08,2.2,19.47,1,1,4,1
Honda Civic,30.4,4,75.7,52,4.93,1.615,18.52,1,1,4,2
Toyota Corolla,33.9,4,71.1,65,4.22,1.835,19.9,1,1,4,1
Toyota Corona,21.5,4,120.1,97,3.7,2.465,20.01,1,0,3,1
Dodge Challenger,15.5,8,318.0,150,2.76,3.52,16.87,0,0,3,2
AMC Javelin,15.2,8,304.0,150,3.15,3.435,17.3,0,0,3,2
Camaro Z28,13.3,8,350.0,245,3.73,3.84,15.41,0,0,3,4
Pontiac Firebird,19.2,8,400.0,175,3.08,3.845,17.05,0,0,3,2
Fiat X1-9,27.3,4,79.0,66,4.08,1.935,18.9,1,1,4,1
Porsche 914-2,26.0,4,120.3,91,4.43,2.14,16.7,0,1,5,2
Lotus Europa,30.4,4,95.1,113,3.77,1.513,16.9,1,1,5,2
Ford Pantera L,15.8,8,351.0,264,4.22,3.17,14.5,0,1,5,4
Ferrari Dino,19.7,6,145.0,175,3.62,2.77,15.5,0,1,5,6
Maserati Bora,15.0,8,301.0,335,3.54,3.57,14.6,0,1,5,8
Volvo 142E,21.4,4,121.0,109,4.11,2.78,18.6,1,1,4,2
```

In [104]:

```
sc.stop()
```