

Contents

Exercise 1: Setting up the Environment	2
Exercise 2: Practicing HDFS Commands.....	4
Exercise 3: Running MapReduce Job.....	Error! Bookmark not defined.
Exercise 4: Launching Spark	6
Exploring Data using RDD operations:.....	9
Transformations.....	9
Action.....	14
PairRDD	19
Developing with Spark	24
REPL	24
Zeppelin	Error! Bookmark not defined.
Exercise 5: Building Spark Application using Eclipse IDE	Error! Bookmark not defined.
Additional Exercise	25
Analyze Movie Lens Dataset using RDD	25
Exercise 6: Dataframe and Spark SQL.....	30
Infer schema by Reflection (case class).....	Error! Bookmark not defined.
Programmatically	Error! Bookmark not defined.
UDF in Spark Dataframe	35
Exercise 7: Spark Streaming.....	48
Exercise 8: SparkML.....	55
Predicting power grid demand using Spark ML.....	55
Exercise9: Machine Learning	71
Linear Regression	72
Logistic Regression.....	84
Decision Tree	86
Random Forest.....	88
Classification	90
Clustering.....	99
PCA (Principle Component Analysis)	111
Exercise10: Graphx	Error! Bookmark not defined.
Analyzing Flight Data	Error! Bookmark not defined.

Exercise 1: Setting up the Environment

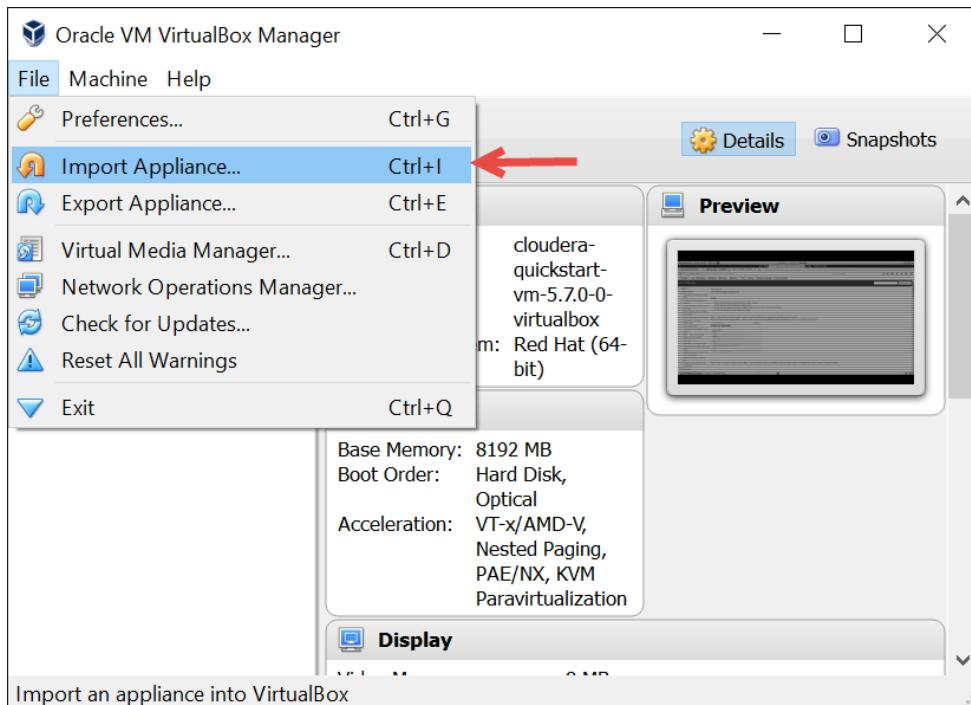
Step1: Installing Oracle VM virtual box from Training Bundle.

- Under ‘Training Bundle’ folder, navigate to “Software” folder.
- Find the executable file named ‘VirtualBox-5.0.16-105871-Win’ and complete the installation.

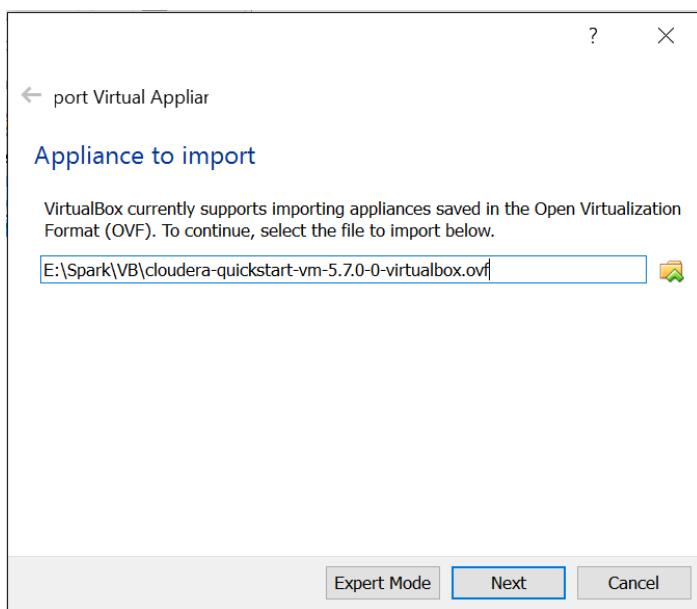
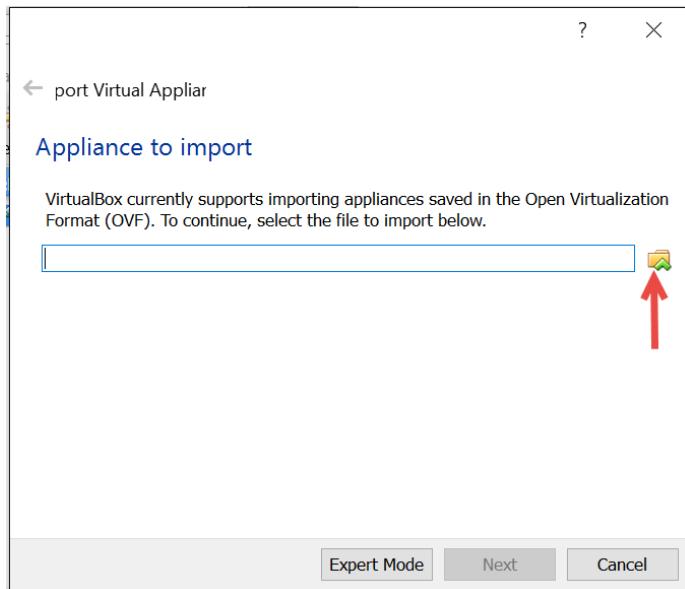
Step2: Extracting the Image

- From the same training bundle, locate the folder named VM image> ‘cloudera-quickstart-vm-5.12.0-0-virtualbox’
- Load the image to get started with the machine.

Step3: Importing the file



- Browse to the location under “Training Bundle” and select the OVF file.



- This will prepare your machine.
- After import, change the number of processors to two before starting the machine.
- Verify all the running services with jps command

```
[cloudera@quickstart ~]$ sudo jps
6403 RESTServer
5709 SecondaryNameNode
5980 NodeManager
6878 RunJar
8543
10329 ZeppelinServer
9721 org.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar
5841 Bootstrap
5366 DataNode
5454 JournalNode
8584
7516 HistoryServer
8518 Bootstrap
5573 NameNode
7484 Bootstrap
5303 QuorumPeerMain
6659 RunJar
5897 JobHistoryServer
18565 Jps
6184 ResourceManager
6521 ThriftServer
8191 Bootstrap
[cloudera@quickstart ~]$
```

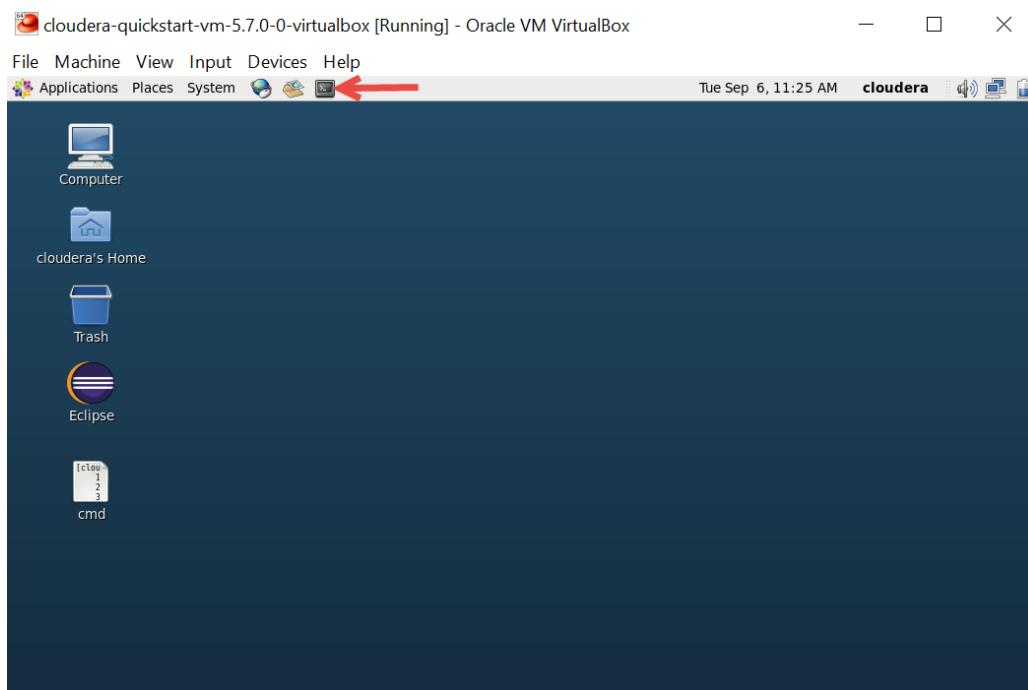
Exercise 2: Practicing HDFS Commands

Task 1: Using Hadoop Command Line Interface

Navigate to Application > System Tools > Terminal

OR

Use shortcut to open the terminal.



- Operation on Files and directories

To check the content of root directory in HDFS

```
hdfs dfs -ls /
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 5 items
drwxrwxrwx  - hdfs  supergroup          0 2016-04-05 23:56 /benchmarks
drwxr-xr-x  - hbase  supergroup          0 2016-09-04 10:14 /hbase
drwxrwxrwt  - hdfs  supergroup          0 2016-08-03 01:56 /tmp
drwxr-xr-x  - hdfs  supergroup          0 2016-08-02 08:08 /user
drwxr-xr-x  - hdfs  supergroup          0 2016-04-05 23:58 /var
[cloudera@quickstart ~]$ █
```

View the content of /user directory

```
hdfs dfs -ls /user
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user
Found 9 items
drwxr-xr-x  - cloudera  cloudera        0 2016-08-03 01:54 /user/cloudera
drwxr-xr-x  - hdfs      supergroup       0 2016-08-02 08:08 /user/hdfs
drwxr-xr-x  - mapred    hadoop          0 2016-04-05 23:56 /user/history
drwxrwxrwx   - hive      supergroup       0 2016-04-05 23:58 /user/hive
drwxrwxrwx   - hue       supergroup       0 2016-04-05 23:57 /user/hue
drwxrwxrwx   - jenkins   supergroup       0 2016-04-05 23:56 /user/jenkins
drwxrwxrwx   - oozie     supergroup       0 2016-04-05 23:57 /user/oozie
drwxrwxrwx   - root      supergroup       0 2016-04-05 23:57 /user/root
drwxr-xr-x  - hdfs      supergroup       0 2016-04-05 23:58 /user/spark
```

 For an empty directory the prompt does not show any error while querying whereas if the directory doesn't exist it will throw an error.

Example: Query a non-existing directory say /music

```
hdfs dfs -ls /music
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /music
ls: `/music': No such file or directory
[cloudera@quickstart ~]$ █
```



The directory structure of Hadoop filesystem is different from local filesystem i.e. Linux filesystem. To know the difference, list the files on your local filesystem.

Create a new directory named “hadoop” in HDFS

```
hdfs dfs -mkdir hadoop
```

Create a sample.txt file on local and upload the file into newly created directory

```
hdfs dfs -put sample.txt hadoop/sample.txt
```

View the content of new file

```
hdfs dfs -cat hadoop/sample.txt
```



In HDFS, any non-absolute path is considered relative to your home directory i.e. /user/cloudera. Unlike Linux filesystem concept of “current” or “present working directory”.

Download a file from HDFS to your local filesystem, specify HDFS path and local path to achieve the same.

```
hdfs dfs -get /user/cloudera/sample.txt /home/cloudera
```

Remove the directory along with its content (perform this step after completing Task 2)

```
hdfs dfs -rm -r hadoop
```

List all the shell commands supported by Hadoop

```
hdfs dfs
```

Exercise 3: Launching Spark

- Spark shell can be launched in two ways i.e. Scala and Python.
- To launch Scala spark shell, follow below steps

```
$ spark-shell
```

```
[cloudera@quickstart ~]$ spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/St
aticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to

    / \ / \
   / \ \ / \
  / \ \ / \
 / \ \ / \
version 1.6.0

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
```



You will see several INFO and WARNING message on the command prompt after launching spark-shell, which can be disregarded.

Logs will appear as below and finally you will get Scala Prompt

```
SQL context available as sqlContext.
```

```
scala> █
```

- Spark creates a SparkContext object called sc, verify that the object exists

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@a23c46d
```

- To know various SparkContext methods which are available, type sc. (sc followed by dot) and then TAB key

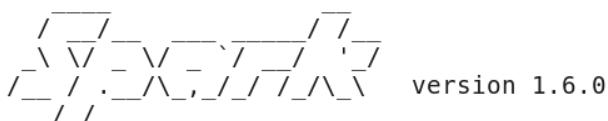
```
scala> sc.
accumulable          accumulableCollection
accumulator         addFile
addJar              addSparkListener
appName             applicationAttemptId
applicationId      asInstanceOf
binaryFiles         binaryRecords
broadcast           cancelAllJobs
cancelJobGroup     clearCallSite
clearFiles          clearJars
clearJobGroup       defaultMinPartitions
defaultMinSplits    defaultParallelism
emptyRDD            externalBlockStoreFolderName
files               getAllPools
getCheckpointDir   getConf
getExecutorMemoryStatus getExecutorStorageStatus
getLocalProperty   getPersistentRDDs
getPoolForName     getRDDStorageInfo
getSchedulingMode hadoopConfiguration
hadoopFile          hadoopRDD
initLocalProperties isInstanceOf
isLocal             isStopped
jars                killExecutor
killExecutors       makeRDD
```

Spark is based on the concept of Resilient Distributed Dataset (RDD), which is fault tolerant collection of elements that can be operated in parallel.

- Invoke python spark shell by using ‘pyspark’

```
[cloudera@quickstart ~]$ pyspark
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

Welcome to



version 1.6.0

```
Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>> █
```

Two ways to create RDDs:

- Parallelizing an existing collection
- Referencing a dataset from an External Storage System

Parallelized collection:

To create a parallelized collection holding numbers 1 to 6, use **sc.parallelize** method

```
data = [1, 2, 3, 4, 5, 6]
distData = sc.parallelize(data)
```

Once ‘distData’ RDD is created, we can perform **Action** such as:

- Finding the sum, mean & variance

```
distData.sum()
distData.mean()
distData.variance()
```

External Storage System:

Spark can create distributed datasets from any storage system supported by Hadoop, Spark supports text files, Sequence Files and any other Hadoop Input Format.

Load a file from your Local Filesystem say batting.txt using **sc.textFile** method.

```
textFile = sc.textFile("file:/home/cloudera/datasets/story.txt")
```

Operations on RDD will be discussed in next section.

Exploring Data using RDD operations:

Transformations

filter

Return a new dataset formed by selecting those elements of the source on which *func* returns true.

```
a = sc.parallelize(range(1,10))
b = a.filter(lambda x: x % 2 == 0)
b.collect
```

```
>>> a = sc.parallelize(range(1,10))
>>> b = a.filter(lambda x: x % 2 == 0)
>>> b.collect()
[2, 4, 6, 8]
>>> █
```

map

Return a new distributed dataset formed by passing each element of the source through a function *func*.

```
a = sc.parallelize(["dog", "salmon", "salmon", "rat", "elephant"])

b = a.map(lambda x:len(x))

c = a.zip(b)

c.collect()
```

```
>>> c = a.zip(b)
>>> c.collect()
[('dog', 3), ('salmon', 6), ('salmon', 6), ('rat', 3), ('elephant', 8)]
```

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

squared = nums.map(lambda x: x * x).collect()

for num in squared: print "%i " % (num)
```

```
>>> for num in squared: print "%i " % (num)
...
1
4
9
16
>>> █
```

distinct

Return a new dataset that contains the distinct elements of the source dataset.

```
c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"],2)

c.distinct().collect()
```

```
>>> c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"],2)
>>> c
ParallelCollectionRDD[27] at parallelize at PythonRDD.scala:423
>>> c.distinct().collect()
['Rat', 'Gnu', 'Dog', 'Cat']
>>> 
```

cartesian

When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).

```
x = sc.parallelize([1,2,3,4,5])
y = sc.parallelize([6,7,8,9,10])
x.cartesian(y).collect()
```

```
>>> x = sc.parallelize([1,2,3,4,5])
>>> x
ParallelCollectionRDD[33] at parallelize at PythonRDD.scala:423
>>> y = sc.parallelize([6,7,8,9,10])
>>> y
ParallelCollectionRDD[34] at parallelize at PythonRDD.scala:423
>>> x.cartesian(y).collect()
[(1, 6), (1, 7), (2, 6), (2, 7), (1, 8), (1, 9), (2, 8), (2, 9), (1, 10), (2, 10), (3, 6), (3, 7), (4, 6), (4, 7), (5, 6), (5, 7), (3, 8), (3, 9), (4, 8), (4, 9), (3, 10), (4, 10), (5, 8), (5, 9), (5, 10)]
>>> 
```

coalesce

Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.

```
y = sc.parallelize(range(1,10), 10)
z = y.coalesce(2, False)
z.glom().collect()
z.getNumPartitions()
```

```
>>> y = sc.parallelize(range(1,10), 10)
>>> y
ParallelCollectionRDD[41] at parallelize at PythonRDD.scala:423
>>> 
```

```
>>> z.glom().collect()
[[1, 2, 3, 4], [5, 6, 7, 8, 9]]
>>> z.getNumPartitions()
2
>>> █
```

flatMap

Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).

```
a = sc.parallelize(range(1,10), 5)
a.flatMap(lambda x: range(1, x)).collect()
```

```
>>> a = sc.parallelize(range(1,10), 5)
>>> a
ParallelCollectionRDD[52] at parallelize at PythonRDD.scala:423
>>> a.flatMap(lambda x: range(1, x)).collect()
[1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 8]
>>> █
```

```
b = sc.parallelize([1, 2, 3], 2).flatMap(lambda x: [x, x, x])
b.collect()
```

```
>>> b = sc.parallelize([1, 2, 3], 2).flatMap(lambda x: [x, x, x])
>>> b
PythonRDD[60] at RDD at PythonRDD.scala:43
>>> b.collect()
[1, 1, 1, 2, 2, 2, 3, 3, 3]
>>> █
```

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first() // returns "hello"
```

```
>>> lines = sc.parallelize(["hello world", "hi"])
>>> words = lines.flatMap(lambda line: line.split(" "))
>>> words.first()
'hello'
>>> █
```

```
odd_rdd = dist_numbers.filter(lambda x: x % 2 != 0)
```

groupBy

```
rdd = sc.parallelize([1, 1, 2, 3, 5, 8])
result = rdd.groupBy(lambda x: x % 2).collect()
sorted([(x, sorted(y)) for (x, y) in result])
```

```
>>> result = rdd.groupBy(lambda x: x % 2).collect()
>>> sorted([(x, sorted(y)) for (x, y) in result])
[(0, [2, 8]), (1, [1, 1, 3, 5])]
```

keys

Extracts the keys from all contained tuples and returns them in a new RDD.

```
a = sc.parallelize(["dog", "tiger", "lion", "cat", "panther", "eagle"], 2)
b = a.map(lambda x : (len(x), x))
b.keys().collect()
```

```
>>> b = a.map(lambda x : (len(x), x))
>>> b
PythonRDD[82] at RDD at PythonRDD.scala:43
>>> b.keys().collect()
[3, 5, 4, 3, 7, 5]
>>>
```

union

```
seta = sc.parallelize(range(1,10))
setb = sc.parallelize(range(5 ,15))
(seta.union(setb)).collect()
```

```
>>> seta = sc.parallelize(range(1,10))
>>> seta
ParallelCollectionRDD[84] at parallelize at PythonRDD.scala:423
>>> setb = sc.parallelize(range(5 ,15))
>>> (seta.union(setb)).collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>>
```

zip

Joins two RDDs by combining the i-th of either partition with each other. The resulting RDD will consist of two-component tuples which are interpreted as key-value pairs by the methods provided by the PairRDDFunctions extension.

```
a = sc.parallelize(range(1 , 100), 3)
b = sc.parallelize(range(101 , 200), 3)
a.zip(b).collect()
```

```
>>> a = sc.parallelize(range(1 , 100), 3)
>>> a
ParallelCollectionRDD[89] at parallelize at PythonRDD.scala:423
>>> b = sc.parallelize(range(101 , 200), 3)
>>> b
ParallelCollectionRDD[90] at parallelize at PythonRDD.scala:423
>>> a.zip(b).collect()
[(1, 101), (2, 102), (3, 103), (4, 104), (5, 105), (6, 106), (7, 107), (8, 108), (9, 109), (10, 110), (11, 111), (12, 112), (13, 113), (14, 114), (15, 115),
(16, 116), (17, 117), (18, 118), (19, 119), (20, 120), (21, 121), (22, 122), (23, 123), (24, 124), (25, 125), (26, 126), (27, 127), (28, 128), (29, 129), (30
, 130), (31, 131), (32, 132), (33, 133), (34, 134), (35, 135), (36, 136), (37, 137), (38, 138), (39, 139), (40, 140), (41, 141), (42, 142), (43, 143), (44, 1
44), (45, 145), (46, 146), (47, 147), (48, 148), (49, 149), (50, 150), (51, 151), (52, 152), (53, 153), (54, 154), (55, 155), (56, 156), (57, 157), (58, 158)
, (59, 159), (60, 160), (61, 161), (62, 162), (63, 163), (64, 164), (65, 165), (66, 166), (67, 167), (68, 168), (69, 169), (70, 170), (71, 171), (72, 172), (7
3, 173), (74, 174), (75, 175), (76, 176), (77, 177), (78, 178), (79, 179), (80, 180), (81, 181), (82, 182), (83, 183), (84, 184), (85, 185), (86, 186), (87,
187), (88, 188), (89, 189), (90, 190), (91, 191), (92, 192), (93, 193), (94, 194), (95, 195), (96, 196), (97, 197), (98, 198), (99, 199)]
>>> 
```

Action

collect

Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

```
c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"], 2)
c.collect
```

```
>>> c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"], 2)
>>> c
ParallelCollectionRDD[92] at parallelize at PythonRDD.scala:423
>>> c.collect()
['Gnu', 'Cat', 'Rat', 'Dog', 'Gnu', 'Rat']
>>> 
```

collectAsMap

Similar to collect, but works on key-value RDDs and converts them into Python maps to preserve their key-value structure.

```
a = sc.parallelize([1, 2, 1, 3], 1)
```

```
b = a.zip(a)
b.collectAsMap()

>>> a = sc.parallelize([1, 2, 1, 3], 1)
>>> b = a.zip(a)
>>> b.collectAsMap()
{1: 1, 2: 2, 3: 3}
>>> █
```

count

Return the number of elements in the dataset.

```
c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog"], 2)
c.count()
res2: Long = 4
```

```
>>> c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog"], 2)
>>> c.count()
4
>>> █
```

reduce

Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.

```
a = sc.parallelize([1, 2, 3, 4, 5]).reduce(lambda a,b : a+b)

>>> a = sc.parallelize([1, 2, 3, 4, 5]).reduce(lambda a,b : a+b)
>>> a
15
>>> █
```

take

Return an array with the first *n* elements of the dataset.

```
b = sc.parallelize(["dog", "cat", "ape", "salmon", "gnu"], 2)
b.take(2)
```

```
>>> b = sc.parallelize(["dog", "cat", "ape", "salmon", "gnu"], 2)
>>> b
ParallelCollectionRDD[103] at parallelize at PythonRDD.scala:423
>>> b.take(2)
['dog', 'cat']
>>> █
```

```
b = sc.parallelize(range(1 , 100), 5)
b.take(30)
```

```
>>> b = sc.parallelize(range(1 , 100), 5)
>>> b.take(30)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
>>> █
```

first

Return the first element of the dataset (similar to take(1)).

```
c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog"], 2)
c.first()
```

```
>>> c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog"], 2)
>>> c.first()
'Gnu'
>>> █
```

countByValue

Returns a map that contains all unique values of the RDD and their respective occurrence counts.

```
b = sc.parallelize([1,2,3,4,5,6,7,8,2,4,2,1,1,1,1,1])
b.countByValue()
```

```
>>> b = sc.parallelize([1,2,3,4,5,6,7,8,2,4,2,1,1,1,1,1])
>>> b.countByValue()
defaultdict(<type 'int'>, {1: 6, 2: 3, 3: 1, 4: 2, 5: 1, 6: 1, 7: 1, 8: 1})
>>> █
```

lookup

Scans the RDD for all keys that match the provided value and returns their values as a Python sequence.

```
a = sc.parallelize(["dog", "tiger", "lion", "cat", "panther", "eagle"], 2)
b = a.map(lambda x : (len(x), x))
b.lookup(5)
```

```
>>> a = sc.parallelize(["dog", "tiger", "lion", "cat", "panther", "eagle"], 2)
>>> b = a.map(lambda x : (len(x), x))
>>> b.lookup(5)
['tiger', 'eagle']
>>>
>>> type(b)
<class 'pyspark.rdd.PipelinedRDD'>
>>> ■
```

max

Returns the largest element in the RDD

```
y = sc.parallelize(range(10, 30))
y.max()
```

```
>>> y = sc.parallelize(range(10, 30))
>>> y.max()
29
>>> ■
```

```
a = sc.parallelize([(10, "dog"), (3, "tiger"), (9, "lion"), (18, "cat")])
a.max()
```

```
>>> a = sc.parallelize([(10, "dog"), (3, "tiger"), (9, "lion"), (18, "cat")])
>>> a.max()
(18, 'cat')
>>>
>>> type(a)
<class 'pyspark.rdd.RDD'>
>>> ■
```

min

Returns the smallest element in the RDD

```
y = sc.parallelize(range(10, 30))
y.min()
```

```
>>> y = sc.parallelize(range(10 , 30))
>>> y.min()
10
>>> ■
```

```
a = sc.parallelize([(10, "dog"), (3, "tiger"), (9, "lion"), (8, "cat")])
a.min()
```

```
**>>> a = sc.parallelize([(10, "dog"), (3, "tiger"), (9, "lion"), (8, "cat")])
>>> a.min()
(3, 'tiger')
>>> ■
```

mean

Calls stats and extracts the mean component.

```
a = sc.parallelize([9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0, 2.1, 7.4, 7.5, 7.6,
8.8, 10.0, 8.9, 5.5], 3)
a.mean()
```

```
>>> a = sc.parallelize([9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0, 2.1, 7.4, 7.5, 7.6, 8.8, 10.0, 8.9, 5.5], 3)
>>> a.mean()
5.299999999999998
>>> ■
```

variance

Calls stats and extracts either variance-component or corrected sampleVariance-component.

```
a = sc.parallelize([9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0, 2.1, 7.4, 7.5, 7.6,
8.8, 10.0, 8.9, 5.5], 3)
a.variance()
```

```
>>> a = sc.parallelize([9.1, 1.0, 1.2, 2.1, 1.3, 5.0, 2.0, 2.1, 7.4, 7.5, 7.6, 8.8, 10.0, 8.9, 5.5], 3)
>>> a
ParallelCollectionRDD[124] at parallelize at PythonRDD.scala:423
>>> a.variance()
10.60533333333332
>>> ■
```

PairRDD

countByKey

Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

```
c = sc.parallelize([(3, "Gnu"), (3, "Yak"), (5, "Mouse"), (3, "Dog")], 2)
c.countByKey()
```

```
=====
>>> c = sc.parallelize([(3, "Gnu"), (3, "Yak"), (5, "Mouse"), (3, "Dog")], 2)
>>> c.countByKey()
defaultdict(<type 'int'>, {3: 3, 5: 1})
>>> █
```

groupByKey

When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.

```
x = sc.parallelize([('a', 1), ('b', 1), ('a', 1)])
map((lambda (x,y): (x, list(y))), sorted(x.groupByKey().collect())))
```

```
=====
>>> x = sc.parallelize([('a', 1), ('b', 1), ('a', 1)])
>>> map((lambda (x,y): (x, list(y))), sorted(x.groupByKey().collect()))
[('a', [1, 1]), ('b', [1])]
>>> █
```

reduceByKey

When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V

```
a = sc.parallelize(["dog", "cat", "owl", "gnu", "ant"], 2)
b = a.map(lambda x : (len(x), x))
b.reduceByKey(lambda x,y : x + y).collect()
```

```
=====
>>> a = sc.parallelize(["dog", "cat", "owl", "gnu", "ant"], 2)
>>> b = a.map(lambda x : (len(x), x))
>>> b.reduceByKey(lambda x,y : x + y).collect()
[(3, 'dogcatowlgnuant')]
>>> █
```

foldByKey

Very similar to *fold*, but performs the folding separately for each key of the RDD. This function is only available if the RDD consists of two-component tuples.

```
rdd = sc.parallelize([('a', 1), ('b', 1), ('a', 1)])
from operator import add
rdd.foldByKey(0, add).collect()
```

```
>>> rdd = sc.parallelize([('a', 1), ('b', 1), ('a', 1)])
>>> from operator import add
>>> rdd.foldByKey(0, add).collect()
[('a', 2), ('b', 1)]
>>> █
```

cogroup

When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples.

```
x = sc.parallelize([('a', 1), ('b', 4)])
y = sc.parallelize([('a', 2)])
map((lambda (x,y): (x, (list(y[0]), list(y[1])))),
sorted(list(x.cogroup(y).collect())))
>>> █
```

```
>>> x = sc.parallelize([('a', 1), ('b', 4)])
>>> y = sc.parallelize([('a', 2)])
>>> map((lambda (x,y): (x, (list(y[0]), list(y[1])))), sorted(list(x.cogroup(y).collect())))
[('a', ([1], [2])), ('b', ([4], []))]
>>> █
```

```
x = sc.parallelize([(1, "apple"), (2, "banana"), (3, "orange"), (4, "kiwi")], 2)
y = sc.parallelize([(5, "computer"), (1, "laptop"), (1, "desktop"), (4, "iPad")], 2)
map((lambda (x,y): (x, (list(y[0]), list(y[1])))),
sorted(list(x.cogroup(y).collect())))
>>> █
```

```
>>> x = sc.parallelize([(1, "apple"), (2, "banana"), (3, "orange"), (4, "kiwi")], 2)
>>> y = sc.parallelize([(5, "computer"), (1, "laptop"), (1, "desktop"), (4, "iPad")], 2)
>>> map((lambda (x,y): (x, (list(y[0]), list(y[1])))), sorted(list(x.cogroup(y).collect())))
[(1, ['apple']), ('laptop', 'desktop')), (2, ['banana', []]), (3, ['orange', []]), (4, ['kiwi', 'iPad']), (5, ['computer'])]
>>> █
```

Join

Performs an inner join using two key-value RDDs.

```
a = sc.parallelize(["dog", "salmon", "salmon", "rat", "elephant"], 3)
b = a.keyBy(lambda x:len(x))
c =
sc.parallelize(["dog","cat","gnu","salmon","rabbit","turkey","wolf","bear",
"bee"], 3)
d = c.keyBy(lambda x:len(x))
b.join(d).collect()

>>> d = c.keyBy(lambda x:len(x))
>>> b.join(d).collect()
[(6, ('salmon', 'salmon')), (6, ('salmon', 'rabbit')), (6, ('salmon', 'turkey')), (6, ('salmon', 'salmon')), (6, ('salmon', 'rabbit')), (6, ('salmon', 'turkey')), (3, ('dog', 'dog')), (3, ('dog', 'cat')), (3, ('dog', 'gnu')), (3, ('dog', 'bee')), (3, ('rat', 'dog')), (3, ('rat', 'cat')), (3, ('rat', 'gnu')), (3, ('rat', 'bee'))]
>>> 
```

leftOuterJoin

Performs an left outer join using two key-value RDDs.

```
a = sc.parallelize(["dog", "salmon", "salmon", "rat", "elephant"], 3)
b = a.keyBy(lambda x:len(x))
c =
sc.parallelize(["dog","cat","gnu","salmon","rabbit","turkey","wolf","bear",
"bee"], 3)
d = c.keyBy(lambda x:len(x))
b.leftOuterJoin(d).collect()

>>> a = sc.parallelize(["dog", "salmon", "salmon", "rat", "elephant"], 3)
>>> b = a.keyBy(lambda x:len(x))
>>> c = sc.parallelize(["dog","cat","gnu","salmon","rabbit","turkey","wolf","bear",
"bee"], 3)
>>> d = c.keyBy(lambda x:len(x))
>>> b.leftOuterJoin(d).collect()
[(6, ('salmon', 'salmon')), (6, ('salmon', 'rabbit')), (6, ('salmon', 'turkey')), (6, ('salmon', 'salmon')), (6, ('salmon', 'rabbit')), (6, ('salmon', 'turkey')), (8, ('elephant', None)), (3, ('dog', 'dog')), (3, ('dog', 'cat')), (3, ('dog', 'gnu')), (3, ('dog', 'bee')), (3, ('rat', 'dog')), (3, ('rat', 'cat')), (3, ('rat', 'gnu')), (3, ('rat', 'bee'))]
>>> 
```

rightOuterJoin

Performs an right outer join using two key-value RDDs.

```
a = sc.parallelize(["dog", "salmon", "salmon", "rat", "elephant"], 3)
b = a.keyBy(lambda x:len(x))
```

```

c =
sc.parallelize(["dog", "cat", "gnu", "salmon", "rabbit", "turkey", "wolf", "bear",
"bee"], 3)

d = c.keyBy(lambda x:len(x))

b.rightOuterJoin(d).collect()

```

```

>>> b.rightOuterJoin(d).collect()
[(6, ('salmon', 'salmon')), (6, ('salmon', 'rabbit')), (6, ('salmon', 'turkey')), (6, ('salmon', 'salmon')), (6, ('salmon', 'rabbit')), (6, ('salmon', 'turkey')),
(3, ('dog', 'dog')), (3, ('dog', 'cat')), (3, ('dog', 'gnu')), (3, ('dog', 'bee')), (3, ('rat', 'dog')), (3, ('rat', 'cat')), (3, ('rat', 'gnu')), (3,
('rat', 'bee')), (4, (None, 'wolf')), (4, (None, 'bear'))]
>>> █

```

keyBy

Constructs two-component tuples (key-value pairs) by applying a function on each data item. The result of the function becomes the key and the original data item becomes the value of the newly created tuples.

```

a = sc.parallelize(["dog", "salmon", "salmon", "rat", "elephant"], 3)

b = a.keyBy(lambda x:len(x))

b.collect()

```

```

>>> a = sc.parallelize(["dog", "salmon", "salmon", "rat", "elephant"], 3)
>>> b = a.keyBy(lambda x:len(x))
>>> b.collect()
[(3, 'dog'), (6, 'salmon'), (6, 'salmon'), (3, 'rat'), (8, 'elephant')]
>>> █

```

mapValues

Takes the values of a RDD that consists of two-component tuples, and applies the provided function to transform each value. Then, it forms new two-component tuples using the key and the transformed value and stores them in a new RDD.

```

a = sc.parallelize(["dog", "tiger", "lion", "cat", "panther", "eagle"], 2)

b = a.map(lambda x : (len(x), x))

b.mapValues(lambda y :"x" + y + "x").collect()

```

```
>>> a = sc.parallelize(["dog", "tiger", "lion", "cat", "panther", "eagle"], 2)
>>> b = a.map(lambda x : (len(x), x))
>>> b.mapValues(lambda y :"x" + y + "x").collect()
[(3, 'xdogx'), (5, 'xtigerx'), (4, 'xlionx'), (3, 'xcatx'), (7, 'xpantherx'), (5, 'xeaglex')]
>>> █
```

cache

The cache() method is a shorthand for using the default storage level, which is StorageLevel.MEMORY_ONLY (store deserialized objects in memory)

```
c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"], 2)
c.getStorageLevel()
```

```
>>> c = sc.parallelize(["Gnu", "Cat", "Rat", "Dog", "Gnu", "Rat"], 2)
>>> c.getStorageLevel()
StorageLevel(False, False, False, False, 1)
>>> █
```

```
c.cache()
c.getStorageLevel()
```

```
>>> c.cache()
ParallelCollectionRDD[243] at parallelize at PythonRDD.scala:423
>>> c.getStorageLevel()
StorageLevel(False, True, False, False, 1)
>>> █
```

repartition

Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.

```
rdd = sc.parallelize([1, 2, 10, 4, 5, 2, 1, 1, 1], 3)
rdd.getNumPartitions()
rdd2 = rdd.repartition(5)
rdd2.getNumPartitions()
```

```
>>> rdd = sc.parallelize([1, 2, 10, 4, 5, 2, 1, 1, 1], 3)
>>> rdd.getNumPartitions()
3
>>> rdd2 = rdd.repartition(5)
>>> rdd2.getNumPartitions()
5
>>> 
```

Developing with Spark

REPL

Example: Counting the occurrence of lines having a particular word in it

```
textFile = sc.textFile("file:/home/cloudera/datasets/Joyce.txt")
```

```
textFile.count() //Return the number of elements in the dataset
```

```
>>> textFile = sc.textFile("file:/home/cloudera/datasets/Joyce.txt")
>>> textFile.count()
32790
>>> 
```

```
linesWithJoyce = textFile.filter(lambda line : "Joyce" in line)
```

```
linesWithJoyce.count() //How many lines contains "Joyce"
```

```
>>> linesWithJoyce = textFile.filter(lambda line : "Joyce" in line)
>>> linesWithJoyce.count()
6
>>> 
```

Example: Add up the sizes of all the lines

```

lineLength = textFile.map(lambda s : len(s))

totalLength = lineLength.reduce(lambda a, b : a + b) //Aggregate the
elements of the dataset using a function

```

```

>>> totalLength = lineLength.reduce(lambda a, b : a + b)
>>> totalLength
1496373

```

Example: To find out the words count

```

counts=textFile.flatMap(lambda line: line.split(" ")).map(lambda word:
(word, 1)) .reduceByKey(lambda a, b: a + b)

counts.collect()

```

```

>>> counts=textFile.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b).sortByKey()
>>> counts.first()
(u'', 10988)
>>> counts.take(5)
[(u'', 10988), (u'"AS-IS".', 1), (u'"COME', 1), (u'"Defects".', 1), (u'"I', 1)]
>>>

```

Additional Exercise

Analyze Movie Lens Dataset using RDD

Source: <http://grouplens.org/datasets/movielens/>

Objective: What are the occupations of the users who have rated the iconic movie Titanic as 5?

Information about Dataset:

RATINGS FILE DESCRIPTION

All ratings are contained in the file “**ratings.dat**” and are in the following format:

UserID::MovieID::Rating::Timestamp

UserIDs range between 1 and 6040

MovieIDs range between 1 and 3952

Ratings are made on a 5-star scale (whole-star ratings only)

Timestamp is represented in seconds since the epoch as returned by time(2)

Each user has at least 20 ratings

USERS FILE DESCRIPTION

User information is in the file “**users.dat**” and is in the following format:

UserID::Gender::Age::Occupation::Zip-code

All demographic information is provided voluntarily by the users and is not checked for accuracy. Only users who have provided some demographic information are included in this data set.

Gender is denoted by a “M” for male and “F” for female

Age is chosen from the following ranges:

1: “Under 18”

18: “18-24”

25: “25-34”

35: “35-44”

45: “45-49”

50: “50-55”

56: “56+”

Occupation is chosen from the following choices:

0: “other” or not specified

1: “academic/educator”

2: “artist”

3: “clerical/admin”

4: “college/grad student”

5: “customer service”

6: “doctor/health care”

7: “executive/managerial”

8: “farmer”

9: “homemaker”

10: “K-12 student”

11: “lawyer”

12: “programmer”

- 13: “retired”
- 14: “sales/marketing”
- 15: “scientist”
- 16: “self-employed”
- 17: “technician/engineer”
- 18: “tradesman/craftsman”
- 19: “unemployed”
- 20: “writer”

MOVIES FILE DESCRIPTION

Movie information is in the file “**movies.dat**” and is in the following format:

MovieID::Title::Genres

Titles are identical to titles provided by the IMDB (including year of release)

Genres are pipe-separated and are selected from the following genres:

Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western

Steps:

Load the files ‘movies.dat’, ‘users.dat’ and ‘ratings.dat’ into HDFS from local filesystem

1. Create 3 RDD for movies, users and ratings respectively

```
movies = sc.textFile("/home/cloudera/datasets/movies.dat")
users = sc.textFile("/home/cloudera/datasets/users.dat")
ratings = sc.textFile("/home/cloudera/datasets/ratings.dat")
```

2. Print first 10 lines from the movies rdd

```
movies.take(10)
```

3. Find out the id of the movie Titanic

```
movies.filter(lambda x:"Titanic" in x).collect()
```

```
16/08/31 23:42:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
res1: Array[String] = Array(1721::Titanic (1997)::Drama|Romance, 2157::Chambermaid on the Titanic, The (1998)::Romance, 3403::Raise the Titanic (1980)::Drama|Thriller, 3404::Titanic (1953)::Action|Drama)
```

4. Print 10 lines from ratings RDD

```
ratings.take(10)
```

5. Create a filtered rdd based on the users who have rated Titanic as 1

```
titanicRating1 = ratings.map(lambda x: x.split("::")).filter(lambda x: int(x[2]) == 5).collect()
titanicRating1.take(10)
```

```
16/08/31 23:44:48 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
1721    3087    5    974709030
1721    1485    5    974706178
1721    3247    5    974706026
1721    2463    5    974708897
1721    2321    5    974707939
1721    1569    5    974706315
1721    3397    5    974708938
1721    2918    5    974708773
```

6. Convert the previous RDD by assigning user id as key

```
byUser = titanicRating1.keyBy(lambda x :int(x[1]))
byUser.take(3)
```

7. Show some sample values from users rdd

```
users.take(10)
```

```
16/08/31 23:47:02 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all completed, from pool  
1::F::1::10::48067  
2::M::56::16::70072  
3::M::25::15::55117  
4::M::45::7::02460  
5::M::25::20::55455  
6::F::50::9::55117  
7::M::35::1::06810  
8::M::25::12::11413  
9::M::25::17::61614  
10::F::35::1::95370
```

8. Create pair RDD pairRddUsers from users rdd using userid as key

```
pairRddUsers = users.keyBy(lambda x:x.split(":")[0])  
pairRddUsers.take(3)
```

```
>>> users = sc.textFile("file:/home/cloudera/datasets/users.dat")  
>>> pairRddUsers = users.keyBy(lambda x:x.split(":")[0])  
>>> pairRddUsers.take(3)  
[(u'1', u'1::F::1::10::48067'), (u'2', u'2::M::56::16::70072'), (u'3', u'3::M::25::15::55117')]  
>>> █
```

Assignment:

Objective: Using above transformation and action, Analyse Crime Data from San Francisco Police Department.

Steps:

1. Create an RDD with crime incident data.
2. Take first five records of RDD created and print the same.
3. Reference each field with an index map.
4. Show the first value of an RDD.
5. Check number of partition.
6. Capture the line containing header in RDD.
7. Create an RDD with only records eliminating the header.
8. Find the number of incident records?
9. List all the categories from incident data?
10. Find total number of categories?
11. What are the total number of incidents in each category?

12. Find out on which day each of incidents occurred most?

Exercise 6: Dataframe and Spark SQL

Two ways to create Spark Dataframes:

1. Create from an existing RDD
2. Create from other data sources

Creating DataFrames from Existing RDD:

1. Infer schema by Reflection
 - a. Convert RDD containing case classes.
 - b. Use when schema is known.
2. Construct schema programmatically
 - a. Schema is dynamic

1.1 Create from an existing RDD

Spark has a `toDF()` method, that converts an RDD to DataFrame. Note that you need to map the lines into Row objects first:

```
from pyspark.sql import Row
rdd = sc.parallelize([1,2,3,4])
df = rdd.map(lambda l: Row(l)).toDF()
```

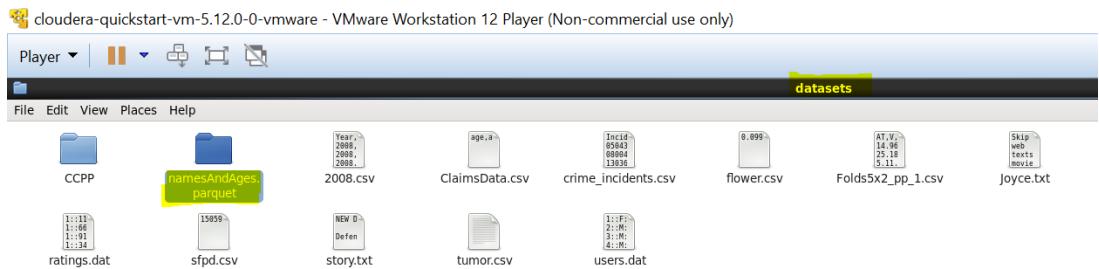
1.2 Creating dataframes from json and saving it to disk

```
df =
sqlContext.read.load("file:/home/cloudera/datasets/people.json",
format="json")

df.select("name", "age").show()

df.select("name",
"age").write.save("file:/home/cloudera/datasets/namesAndAges.parquet",
", format="parquet")
```

Verify the output



2.1 Inferring the Schema Using Reflection

```
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

# Load a text file and convert each line to a Row.
lines =
sc.textFile("file:/home/cloudera/datasets/crime_incidents.csv")
parts = lines.map(lambda l: l.split(","))
crime = parts.map(lambda p: Row(IncidentNum=p[0], Category =p[1],
Descript=p[2],
day=p[3],date=p[4],time=p[5],district=p[6],resolution=p[7],address=p[8],X=p[9],Y=p[10],location=p[11]))
```

Infer the schema, and register the DataFrame as a table.

```
schemaCrime = sqlContext.createDataFrame(crime)
schemaCrime.registerTempTable("incidents")
```

SQL can be run over DataFrames that have been registered as a table.

```
teenagers = sqlContext.sql("SELECT * from incidents")
teenagers.show()
```

Or,

The results of SQL queries are RDDs and support all the normal RDD operations.

```
teenNames = teenagers.map(lambda p: "Name: " + p.name)
for teenName in teenNames.collect():
    print(teenName)
```

2.2 Programmatically Specifying the Schema

A DataFrame can be created programmatically with three steps.

- Create an RDD of tuples or lists from the original RDD;
- Create the schema represented by a StructType matching the structure of tuples or lists in the RDD created in the step 1.
- Apply the schema to the RDD via createDataFrame method provided by SQLContext

Step 1. Load a text file and convert each line to a tuple.

```
lines = sc.textFile("file:/home/cloudera/datasets/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: (p[0], p[1].strip()))
```

Step 2. The schema is encoded in a string.

```
schemaString = "name age"
from pyspark.sql.types import *
fields = [StructField(field_name, StringType(), True) for field_name
in schemaString.split()]
schema = StructType(fields)
```

Step 3. Apply the schema to the RDD and dataframe is created

```
schemaPeople = sqlContext.createDataFrame(people, schema)
```

Print the schema in a tree format

```
schemaPeople.printSchema()
```

Print the rows of the dataframe

```
schemaPeople.show()
```

Let us now Register the DataFrame as a table.

```
schemaPeople.registerTempTable("people")
```

SQL can be run over DataFrames that have been registered as a table.

```
results = sqlContext.sql("SELECT name FROM people")
results.show()
```

Or,

The results of SQL queries are RDDs and support all the normal RDD operations.

```
names = results.map(lambda p: "Name: " + p.name)
for name in names.collect(): print(name)
```

Creating DataFrames with Python

```
# import pyspark class Row from module sql
from pyspark.sql import *
```

Let us Create Example Data - Departments and Employees

Create the Departments

```
department1 = Row(id='123456', name='Computer Science')
```

```
department2 = Row(id='789012', name='Mechanical Engineering')
department3 = Row(id='345678', name='Theater and Drama')
department4 = Row(id='901234', name='Indoor Recreation')
```

Create the Employees

```
Employee = Row("firstName", "lastName", "email", "salary")
employee1 = Employee('michael', 'armbrust', 'no-reply@berkeley.edu',
100000)
employee2 = Employee('xiangrui', 'meng', 'no-reply@stanford.edu',
120000)
employee3 = Employee('matei', None, 'no-reply@waterloo.edu', 140000)
employee4 = Employee(None, 'wendell', 'no-reply@berkeley.edu',
160000)
```

Create the DepartmentWithEmployees instances from Departments and Employees

```
departmentWithEmployees1 = Row(department=department1,
employees=[employee1, employee2])
departmentWithEmployees2 = Row(department=department2,
employees=[employee3, employee4])
departmentWithEmployees3 = Row(department=department3,
employees=[employee1, employee4])
departmentWithEmployees4 = Row(department=department4,
employees=[employee2, employee3])

print department1
print employee2
print departmentWithEmployees1.employees[0].email
```

Create the first DataFrame from a list of the rows.

```
departmentsWithEmployeesSeq1 = [departmentWithEmployees1,  
                                departmentWithEmployees2]  
  
df1 = sqlContext.createDataFrame(departmentsWithEmployeesSeq1)  
  
display(df1)
```

Create a second DataFrame from a list of rows.

```
departmentsWithEmployeesSeq2 = [departmentWithEmployees3,  
                                departmentWithEmployees4]  
  
df2 = sqlContext.createDataFrame(departmentsWithEmployeesSeq2)  
  
display(df2)
```

Working with DataFrames

Union of two DataFrames.

```
unionDF = df1.unionAll(df2)  
display(unionDF)
```

Write the Unioned DataFrame to a Parquet file.

Remove the file if it exists

```
unionDF.write.parquet("file:/home/cloudera/datasets/unionParq")
```

An example using Pandas & Matplotlib Integration

UDF in Spark Dataframe

UDF for Dataframe API

We will write an UDF for the day column and if its Sunday, that will become as weekend else weekdays for the crime_incidents.csv file from which we had created a dataframe in earlier sections.

Parse string day field into weekend/weekday

```
from pyspark.sql.types import StringType  
from pyspark.sql.functions import udf  
  
day_udf = udf(lambda day: "Weekend" if day != 'Sunday' else  
"Weekdays", StringType())  
schemaCrime.withColumn("Calendar", day_udf(schemaCrime.day)).show()
```

Assignment 1

Let us create some data as RDD

```
# Create some data as an RDD:  
  
# Each entry consists of a tuple (weekday, gas_price)  
  
myRDD = sc.parallelize([  
    ("Monday", 1.24),  
    ("Tuesday", 1.42),  
    ("Sunday", 1.33),  
    ("Sunday", 1.21),  
    ("Monday", 1.18)  
])
```

Now, transform the gas price from Euro per litre to USD per gallon, assuming 1.18 USD per Euro and 0.265 gallons per litre. Then calculate the mean of the price and display the result.

Output should be similar to

Note

day	avg(priceUSDperGallon)
Monday	5.387924528301886
Sunday	5.655094339622641
Tuesday	6.323018867924527

Hint: Calculation will be myDF.price * 1.18 / 0.265.

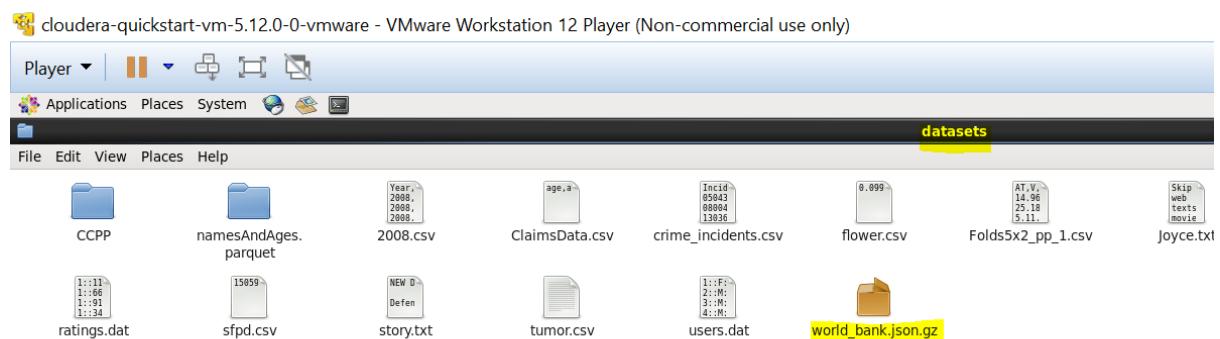
Assignment 2

First of all go to the folder /home/cloudera/datasets and download the data from the link given below

Open Terminal and type below command, which will download the file in the same location

```
wget  
https://raw.githubusercontent.com/bradenrc/sparksql\_pot/master/world\_bank.json.gz
```

The download will happen and show the file as,



1. Let us try to load the file as data frame and print the schema of the file.

```

>>> example1_df = sqlContext.read.json("file:/home/cloudera/datasets/world_bank.json.gz")
>>> print example1_df.printSchema()
root
|-- _id: struct (nullable = true)
|   |-- $oid: string (nullable = true)
|-- approvalfy: string (nullable = true)
|-- board_approval_month: string (nullable = true)
|-- boardapprovaldate: string (nullable = true)
|-- borrower: string (nullable = true)
|-- closingdate: string (nullable = true)
|-- country_namecode: string (nullable = true)
|-- countrycode: string (nullable = true)
|-- countryname: string (nullable = true)
|-- countryshortname: string (nullable = true)
|-- docty: string (nullable = true)
|-- envassesmentcategorycode: string (nullable = true)
|-- grantamt: long (nullable = true)
|-- ibrdcommamt: long (nullable = true)
|-- id: string (nullable = true)
|-- idacommamt: long (nullable = true)
|-- impagency: string (nullable = true)
|-- lendinginstr: string (nullable = true)
|-- lendinginstrtype: string (nullable = true)
|-- lendprojectcost: long (nullable = true)
|-- majorsector_percent: array (nullable = true)
|   |-- element: struct (containsNull = true)
|       |-- Name: string (nullable = true)
|       |-- Percent: long (nullable = true)
|-- mjsector_namecode: array (nullable = true)
|   |-- element: struct (containsNull = true)
|       |-- code: string (nullable = true)
|       |-- name: string (nullable = true)
|-- mjtheme: array (nullable = true)
|   |-- element: string (containsNull = true)
|-- mjtheme_namecode: array (nullable = true)
|   |-- element: struct (containsNull = true)
|       |-- code: string (nullable = true)

```

2. Create a table that's a pointer to the DataFrame as world_bank.
3. Execute the query on the dataframe as, “select * from world_bank”
4. Display the content of the dataframe.

```

>>> temp_df.show(1)
+-----+
| _id|approvalfy|board_approval_month|boardapprovaldate|borrower|closingdate|country_namecode|countrycode|countryname|countryshortname|docty|envassesmentcategorycode|g
rantant|ibrdcommamt|idacommamt|impagency|lendinginstr|lendinginstrtype|lendprojectcost|majorsector_percent|mjsector_namecode|mjtheme|mjtheme_namecode|mjtheme|projline|
xt|productltype|project_abstract|project_name|projectdocs|projectfinancielle|projectstatusdisplay|regionname|regioncode|mjtheme|mjtheme_namecode|mjtheme|projline|
sector|sector_namecode|sectorcode|source|status|supplementprojlg|themel|themenamecode|themecode|totalamt|totalcommnt|url|
+-----+
| [52b213b38594d8a2...| 1990| November|[2013-11-01T00:00:00Z]FEDERAL DEMOCRATIC...|[2018-07-30T00:00:00Z]Federal Democrati...|null|null|null|null|null|null|null|null|null|null|IBRD/I
| 0|0|P129828|138000000|MINISTER OF EDUCATION|Investment Projec...|IN|550000000|[Education,40,...|[Ex,Educati...|Human development,8,1|PE|IBRD/I
DA| L|The development ...|Ethiopia General ...|[28-AUG-2013,PD...| IDA|[Education for all]|null|[Education fo...|65|130000000|[Primary educati...|Primary educati...|[Public administr...|[Tertiary
y educati...|[EP,Primary educ...| ET,B5,E5,EP| IBRD|Active| N|[Education for al...|[65,Education fo...|65|130000000|[http://www.world...|
[52b213b38594d8a2...| 2015| null|0|P144674|0|MINISTRY OF FINANCIAL INVESTMENT...|IN|5700000|[Investment...|[Minist...|[Public adminis...|Economic manag...|1,6|RE|Recipient Execute.
4700000| | |null|TNT| DTF Social Pr...|[29-MAR-2013,PD...| OTHER| Active|Middle East and N...|[Public adminis...|[General public o...|null|
..| | |null|[BS,Public admin...| BZ,B5| IBRD|Active| N|[other economic m...|[24,Other econom...|54,24| | |470000|[http://www.world...|
|[52b213b38594d8a2...| 2014| November|[2013-11-01T00:00:00Z]MINISTRY OF FINAN...|null|null|Tuvalu|Tuvalu|Tuvalu|Tuvalu|Resettlement Plan...|B|
| 0|0|P145310|6660000|MINISTER OF TRANS...|Investment Projec...|IN|6060000|[Transportation,17,Transporta...|[Trade and Inte...|[Trade and int...|5,2,11,6|PE|IBRD/I
DA| L| null|[TI,Rural and In...| TI| IBRD|Active| Y|[Regional integrat...|[47,Regional integrat...|52,81,25,47|6060000|[http://www.world...|
|[52b213b38594d8a2...| 2014| October|[2013-10-31T00:00:00Z]MIN. OF PLANNING ...|null|null|Yemen|Yemen|Republic of Yemen|Procurement Plan...|C|
1500000|0|P144665|0|LABOR INTENSIVE P...|Technical Assista...|IN|1500000|[Health and othe...|[JX,Health and o...|[Social dev/gende...|[7,Social dev/gende...|7,7|RE|Recipient Execute.
..| | |null|[Gov't企划部...|null|null|null|null|[Health and other so...|[Social dev/gende...|[Social dev/gende...|null|null|
..| | |null|[JB,Other social...|JB| IBRD|Active| N|[Participation an...|[57,Participatio...|59,57| | |150000|[http://www.world...|
|[52b213b38594d8a2...| 2014| October|[2013-10-31T00:00:00Z]MINISTRY OF FINANCE|[2019-04-30T00:00:00Z]Kingdom of Lesotho...|LS|Kingdom of Lesotho|Lesotho|Project Informati...|B|
| 0|0|P144933|13100000|MINISTER OF TRADE |Investment Projec...|IN|1500000|[Industry and tr...|[IX,Industry and tr...|[Trade and integrat...|[5,Trade and int...|5,4|PE|IBRD/I
DA| L|The development ...|Second Prim...|[06-SEP-2013,PD...| IDA|[Industry and tr...|41,45|13100000|[African mineral and...|[Other industry,40)| [SME Finance,10]|
|[52b213b38594d8a2...| 2014| Fh,YM,YZ| IBRD|Active| N|[Export developme...|[45,Export developme...|null|null|null|[http://www.world...|
| 0|0|P146161|18000000|null|Investment Projec...|IN|6640000|[Health and othe...|[JX,Health and o...|[Social protectio...|[6,Social protec...|6,6|PE|IBRD/I

```

5. Display query results with a pandas DataFrame.

```

import pandas as pd

```

```
sqlContext.sql("select id, borrower from world_bank limit 2").toPandas()
```

```
>>> import pandas as pd  
>>> sqlContext.sql("select id, borrower from world_bank limit 2").toPandas()  
      id           borrower  
0  P129828  FEDERAL DEMOCRATIC REPUBLIC OF ETHIOPIA  
1  P144674  GOVERNMENT OF TUNISIA  
>>> ■
```

6. Run a groupBy query as below and convert the dataframe to python pandas and observe the output.

```
select  
    regionname ,  
    count(*) as project_count  
from world_bank  
group by regionname  
order by count(*) desc
```

```
>>> sqlContext.sql(query).toPandas()  
      regionname  project_count  
0          Africa            152  
1  East Asia and Pacific        100  
2  Europe and Central Asia         74  
3       South Asia             65  
4  Middle East and North Africa        54  
5  Latin America and Caribbean        53  
6          Other                 2  
>>> ■
```

Exercise 7: Produce and Consume Apache Kafka Messages

Start the Zookeeper server

1. Open a new terminal window and browse to the location “/usr/lib/zookeeper” using cd. Start the zookeeper by command

```
bin/zkServer.sh start
```

```
[cloudera@quickstart ~]$ cd /usr/lib/zookeeper  
[cloudera@quickstart zookeeper]$ bin/zkServer.sh start  
JMX enabled by default  
Using config: /usr/lib/zookeeper/bin/../conf/zoo.cfg  
Starting zookeeper ... STARTED  
[cloudera@quickstart zookeeper]$ █
```

2. You can now validate that Zookeeper is running correctly in standalone mode by connecting to the client port and sending the four letter command “srvr”.

telnet localhost 2181

```
[cloudera@quickstart zookeeper]$ telnet localhost 2181
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
srvr
Zookeeper version: 3.4.5-cdh5.12.0--1, built on 06/29/2017 11:30 GMT
Latency min/avg/max: 0/0/1303
Received: 30491
Sent: 30604
Connections: 4
Outstanding: 0
Zxid: 0xe14
Mode: standalone
Node count: 737
Connection closed by foreign host.
[cloudera@quickstart zookeeper]$ █
```

Starting Kafka and Creating a Kafka Topic

1. Open a new terminal and start Kafka broker, Change the user to cloudera with su – and provide password as cloudera.

```
bin/kafka-server-start.sh config/server.properties
```

Kafka broker is running on port 9042.

2. Open a new terminal window and create a Kafka topic named `weblogs` that will contain messages representing lines in Loudacre's web server logs.

Since your exercise environment is a single-node cluster running on a virtual machine, use a replication factor of 1 and a single partition.

Navigate to the location where Kafka is installed, which is “`/usr/lib/kafka`”.

```
$ bin/kafka-topics.sh --create \ --zookeeper localhost:2181 \ --
replication-factor 1 \ --partitions 1 \
--topic weblogs
```

Topic name

The name of the topic that you wish to create.

Replication Factor

The number of replicas of the topic to maintain within the cluster.

Partitions

The number of partitions to create for the topic.

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Created topic "weblogs".
```

1. Display all Kafka topics to confirm that the new topic you just created is listed:

```
$ kafka-topics --list --zookeeper localhost:2181
```

```
[cloudera@quickstart kafka]$ kafka-topics --list --zookeeper localhost:2181
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
weblogs
[cloudera@quickstart kafka]$
```

2. Describe the Kafka topic created

```
$ bin/kafka-topics.sh --describe --zookeeper localhost:2181
--topic weblogs
```

```
[cloudera@quickstart kafka]$ kafka-topics.sh --describe --zookeeper localhost:2181 --topic weblogs
bash: kafka-topics.sh: command not found
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic:weblogs PartitionCount:1      ReplicationFactor:1      Configs:
      Topic: weblogs Partition: 0    Leader: 0      Replicas: 0      Isr: 0
```

Producing and Consuming Messages

You will now use Kafka command line utilities to start producers and consumers for the topic created earlier.

3. Start a Kafka producer for the weblogs topic:

```
$ kafka-console-producer --broker-list localhost:9092 --topic
weblogs
```

Tip : This exercise involves using multiple terminal windows. To avoid confusion, set a different title for each one by selecting Set Title... on the Terminal menu:

Set the title for this window to “Kafka Producer.”

4. Publish a test message to the weblogs topic by typing the message text and then pressing Enter. For example:

```
test weblog entry 1
```

5. Open a new terminal window and adjust it to fit on the window beneath the producer window. Set the title for this window to “Kafka Consumer.”

6. In the new terminal window, start a Kafka consumer that will read from the beginning of the weblogs topic:

```
$ kafka-console-consumer --zookeeper localhost:2181 --topic  
weblogs --from-beginning
```

You should see the status message you sent using the producer displayed on the consumer’s console, such as:

```
test weblog 1
```

```

cloudera@quickstart:~$ kafka-console-producer --broker-list localhost:9092 --topic weblogs
test weblog entry 1
[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 -topic weblogs --from-beginning
test weblog entry 1

```

7. Press **Ctrl+C** to stop the weblogs consumer, and restart it, but this time omit the `from-beginning` option to this command. You should see that no messages are displayed.

8. Switch back to the producer window and type another test message into the terminal, followed by the **Enter** key:

```
test weblog entry 2
```

The image displays two terminal windows side-by-side. Both windows have a title bar 'cloudera@quickstart:/usr/lib/kafka'. The left window shows the command 'kafka-console-producer --broker-list localhost:9092 --topic weblogs' and its output, which includes multiple SLF4J binding messages and a test message 'test weblog entry 2'. The right window shows the command 'kafka-console-consumer --zookeeper localhost:2181 --topic weblogs' and its output, which includes a deprecation warning about the 'ConsoleConsumer', multiple SLF4J binding messages, and the same test message 'test weblog entry 2'.

```
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092 --topic weblogs
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
test weblog entry 2

[cloudera@quickstart kafka]$ kafka-console-consumer --zookeeper localhost:2181 --topic weblogs
Using the ConsoleConsumer with old consumer is deprecated and will be removed in a future major release. Consider using the new consumer by passing [bootstrap-server] instead of [zookeeper].
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
test weblog entry 2
```

9. Return to the consumer window and verify that it now displays the alert message you published from the producer in the previous step.

Cleaning Up

10. Press Ctrl+C in the consumer terminal window to end its process.
11. Press Ctrl+C in the producer terminal window to end its process.

Exercise 5: Alter Apache Kafka Topics

Let us try altering some parameters for the topic, but instead of using the earlier created topic <weblogs>, we will create another topic "hello-topic" to alter some parameters.

Exercise: Create a topic called hello-topic with replication factor 1 and partition 1.

1. We will add more partitions in the topic created as “hello-topic”. Below command will add 10 more partitions to the hello-topic topic. Note that before, the topic has only 1 partition.

```
bin/kafka-topics.sh --alter --zookeeper localhost:2181 --  
partitions 11 --topic hello-topic
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --alter --zookeeper localhost:2181 --partitions 11 --topic hello-topic  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected  
Adding partitions succeeded!  
[cloudera@quickstart kafka]$
```

Let us verify the number of partitions is being changed with the command

```
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --  
topic hello-topic
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic hello-topic  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
Topic:hello-topic      PartitionCount:11      ReplicationFactor:1      Configs:  
  Topic: hello-topic    Partition: 0    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 1    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 2    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 3    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 4    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 5    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 6    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 7    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 8    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 9    Leader: 0    Replicas: 0    Isr: 0  
  Topic: hello-topic    Partition: 10   Leader: 0    Replicas: 0    Isr: 0  
[cloudera@quickstart kafka]$
```

Tip: REDUCING PARTITION COUNTS

It is not possible to reduce the number of partitions for a topic. The reason this is not supported is because deleting a partition from a topic would cause part of the data in that topic to be deleted as well, which would be inconsistent from a client point of view. In addition, trying to redistribute the data to remaining partitions would be difficult and result in out-of-order messages. Should you need to reduce the number of partitions, you will need to delete the topic and recreate it.

2. Add configurations to the Kafka topic

The general syntax is:

```
bin/kafka-topics.sh --alter --zookeeper localhost:2181 --topic
kafkatopic --config <key>=<value>
```

There are various configuration fields we can set for the topic, here we will set up the max.message.bytes - this is the largest size of the message the broker will allow to be appended to the topic. This size is validated pre-compression. (Defaults to broker's message.max.bytes.)

```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic
hello-topic --config max.message.bytes=128000
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic hello-topic --config max.message.bytes=128000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs.slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs.slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
WARNING: Altering topic configuration from this script has been deprecated and may be removed in future releases.
      Going forward, please use kafka-configs.sh for this functionality
Updated config for topic "hello-topic".
```

3. To remove above overridden configuration, we can use command:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic
hello-topic --delete-config max.message.bytes
```

Exercise 6: Delete a topic

If a topic is no longer needed, it can be deleted in order to free up these resources. In order to perform this action, the brokers in the cluster must have been configured with

the delete.topic.enable option set to true. If this option has been set to false, then the request to delete the topic will be ignored.

We will delete the topic created with the below command

```
bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic hello-topic
```

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic hello-topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Topic hello-topic is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

Lets verify if the topic has been deleted, with the –list command

```
[cloudera@quickstart kafka]$ bin/kafka-topics.sh --zookeeper localhost:2181 --list
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
weblogs
[cloudera@quickstart kafka]$
```

Tip: DATA LOSS AHEAD

Deleting a topic will also delete all its messages. This is not a reversible operation, so make sure it executed carefully.

Exercise 8: Spark Streaming

- Let's start with simple streaming example, to count the number of words in text data received from a data server listening on a TCP socket.
- Create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).
- Create a python project with below inputs:

```

__author__ = 'Kyvor Edubreeze'

# Takes text from input network socket and prints the accumulated
count for each word

from pyspark import SparkContext

from pyspark.streaming import StreamingContext

# create spark and streaming contexts
sc = SparkContext("local[2]", "StreamWordCounter")
ssc = StreamingContext(sc, 10)

# read text from input socket
text = ssc.socketTextStream("localhost", 9999)

# count words
countStream = text.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)

# print the resulting tuples
countStream.pprint()

# start the streaming context
ssc.start()

ssc.awaitTermination()

```

Code Explanation:

First, we import `StreamingContext`, which is the main entry point for all streaming functionality. We create a local `StreamingContext` with two execution threads, and batch interval of 1 second. Using this context, we can create a `DStream` that represents streaming data from a TCP source, specified as hostname (e.g. `localhost`) and port (e.g. `9999`).

This lines `DStream` represents the stream of data that will be received from the data server. Each record in this `DStream` is a line of text. Next, we want to split the lines by space into words. `flatMap` is a one-to-many `DStream` operation that creates a new `DStream` by generating multiple new records from each record in

the source DStream. In this case, each line will be split into multiple words and the stream of words is represented as the words DStream. Next, we want to count these words. The words DStream is further mapped (one-to-one transformation) to a DStream of (word, 1) pairs, which is then reduced to get the frequency of words in each batch of data. Finally, wordCounts.pprint() will print a few of the counts generated every second. Note that when these lines are executed, Spark Streaming only sets up the computation it will perform when it is started, and no real processing has started yet. To start the processing after all the transformations have been setup, we finally call

```
ssc.start()      # Start the computation  
ssc.awaitTermination() # Wait for the computation to terminate
```

- Save the file as WordCount.py in /home/cloudera/SparkStreaming.
- Run the program using spark-submit.
- Submit the python code in the Spark shell with the following script in the command line

```
spark-submit /home/cloudera/SparkStreaming/WordCount.py localhost  
9999
```

Run Netcat command and open another terminal to submit the job

```
nc -nlk 9999
```

- Output:-

```
--  
Time: 2018-08-17 04:45:20  
--  
(u'Hello', 1)  
(u'helo', 2)  
(u'world', 1)
```

Spark Streaming Stateful Transformations

```
# Spark Stateful Streaming with Python  
  
# Takes text from input network socket and prints the accumulated  
# count for each word  
  
from pyspark import SparkContext
```

```
from pyspark.streaming import StreamingContext

# define the update function
def updateTotalCount(currentCount, countState):
    if countState is None:
        countState = 0
    return sum(currentCount, countState)

# create spark and streaming contexts
sc = SparkContext("local[*]", "StreamWordCounter")
ssc = StreamingContext(sc, 10)

# defining the checkpoint directory
ssc.checkpoint("/tmp")

# read text from input socket
text = ssc.socketTextStream("localhost", 9999)

# count words
countStream = text.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)

# update total count for each key
totalCounts = countStream.updateStateByKey(updateTotalCount)

# print the resulting tuples
totalCount.print()

# start the streaming context
ssc.start()
ssc.awaitTermination()
```

Code Explanation:

UpdateStateByKey Operation

The updateStateByKey operation allows you to maintain arbitrary state while continuously updating it with new information. To use this, you will have to do two steps.

Define the state - The state can be an arbitrary data type.

Define the state update function - Specify with a function how to update the state using the previous state and the new values from an input stream.

- Save the file as Streaming.py in /home/cloudera/SparkStreaming
- Submit the python code in the Spark shell with the following script in the command line

```
spark-submit /home/cloudera/SparkStreaming/Streaming.py localhost  
9999
```

- Run the program using spark-submit

```
cloudera@quickstart:~  
[cloudera@quickstart ~]$ spark-submit SparkStreaming/Streaming.py localhost 9999  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/lib/avro/avro-tools-1.7.6-cdh5.12.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory].  
18/04/10 21:45:15 INFO spark.SparkContext: Running Spark version 1.6.0  
18/04/10 21:45:15 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
18/04/10 21:45:15 WARN util.Utils: Your hostname, quickstart.cloudera resolves to a loopback address: 127.0.0.1; using 192.168.152.154 instead (on interface eth3)  
18/04/10 21:45:15 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address  
18/04/10 21:45:15 INFO spark.SecurityManager: Changing view acls to: cloudera  
18/04/10 21:45:15 INFO spark.SecurityManager: Changing modify acls to: cloudera
```

- Run Netcat command and open another terminal to submit the job

```
nc -nlk 9999
```

```
[cloudera@quickstart workspace]$ nc -nlk 9999  
hello world  
hello hadoop
```

- Any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following

```

16/09/13 08:19:03 INFO executor.Executor: Finished task 0.0 in stage 4.0 (TID 3)
. 1312 bytes result sent to driver
16/09/13 08:19:03 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 4.0
(TID 3) in 10 ms on localhost (1/1)
16/09/13 08:19:03 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 4.0, whose t
asks have all completed, from pool
16/09/13 08:19:03 INFO scheduler.DAGScheduler: ResultStage 4 (print at SparkStre
amReceiver.scala:24) finished in 0.012 s
16/09/13 08:19:03 INFO scheduler.DAGScheduler: Job 2 finished: print at SparkStr
eamReceiver.scala:24, took 0.033085 s
-----
Time: 1473779943000 ms
-----
(hello,2)
(world,1)
(hadoop,1)

```

Spark Streaming with Kafka

Before we start, make sure that zookeeper is running on 2181 and Kafka broker is running on 9042 port.

Let us now create a kafka topic named as “new_topic”.

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --
replication-factor 1 --partitions 1 --topic new_topic
```

Let us start Kafka Producer

```
kafka-console-producer --broker-list localhost:9092 --topic
new_topic
```

Create a python project with below inputs in home/cloudera/SparkStreaming folder and save as “KafkaWithSparkStreaming.py” :

```
import sys

from pyspark import SparkContext

from pyspark.streaming import StreamingContext

from pyspark.streaming.kafka import KafkaUtils

if __name__ == "__main__":
    sc = SparkContext(appName="PythonStreamingDirectKafkaWordCount")
    ssc = StreamingContext(sc, 2)
    brokers, topic = sys.argv[1:]
```

```

kvs = KafkaUtils.createDirectStream(ssc,
[topic], {"metadata.broker.list": brokers})

lines = kvs.map(lambda x: x[1])

counts = lines.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)

counts.pprint()

ssc.start()

ssc.awaitTermination()

```

Run the program using spark-submit

```
spark-submit /SparkStreaming/KafkaWithSparkStreaming.py
localhost:9092 new_topic
```

```
[cloudera@quickstart ~]$ spark-submit SparkStreaming/KafkaWithSparkStreaming.py localhost:9092 new_topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/avro-tools-1.7.6-cdh5.12.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
18/04/11 02:05:14 INFO spark.SparkContext: Running Spark version 1.6.0
18/04/11 02:05:15 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/04/11 02:05:16 WARN util.Utils: Your hostname, quickstart.cloudera resolves to a loopback address: 127.0.0.1; using 192.168.152.155 instead (on interface eth3)
18/04/11 02:05:16 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
18/04/11 02:05:16 INFO spark.SecurityManager: Changing view acls to: cloudera
18/04/11 02:05:16 INFO spark.SecurityManager: Changing modify acls to: cloudera
18/04/11 02:05:16 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(cloudera); users with modify permissions: Set(cloudera)
18/04/11 02:05:16 INFO util.Utils: Successfully started service 'sparkDriver' on port 53286.
```

Now we will start sending messages from kafka producer

```
[cloudera@quickstart kafka]$ kafka-console-producer --broker-list localhost:9092
--topic new_topic
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
echo
hi
im dre
```

The messages will be streaming in the spark-submit screen.

To configure PySpark in Jupyter notebook

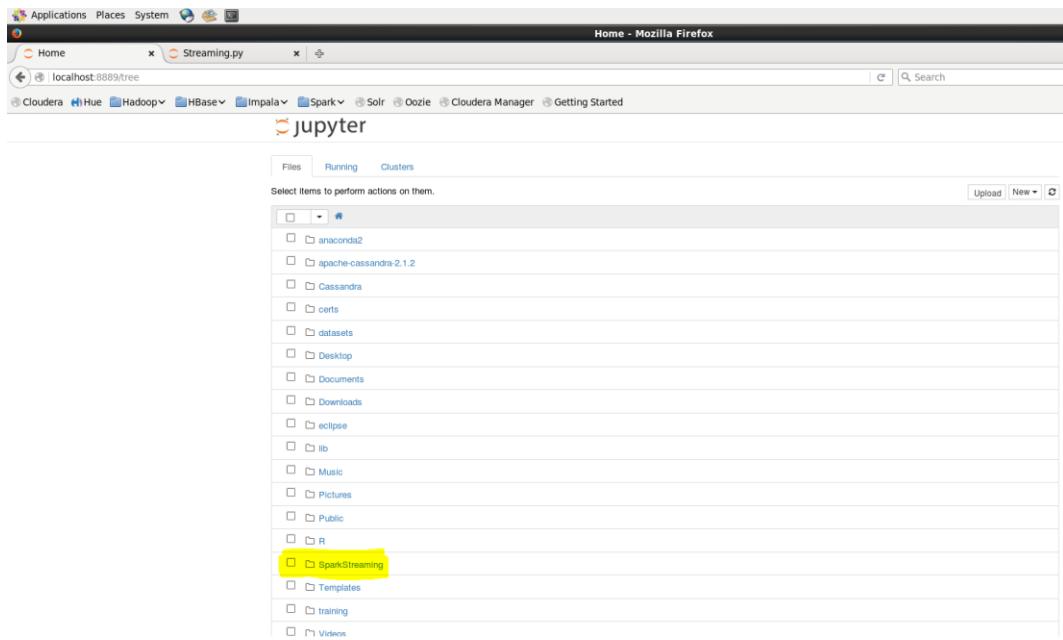
Open the new command terminal and export the below commands.

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

Now type in the command terminal

```
pyspark
```

This will open pyspark in Jupyter notebook.



Click to go in the SparkStreaming folder already created.

Exercise 9: SparkML

Predicting power grid demand using Spark ML

Data Source: <https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant>

Features consist of hourly average ambient variables

Temperature (T) in the range 1.81°C and 37.11°C,

Ambient Pressure (AP) in the range 992.89-1033.30 milibar,

Relative Humidity (RH) in the range 25.56% to 100.16%

Exhaust Vacuum (V) in the range 25.36-81.56 cm Hg

Net hourly electrical energy output (EP) 420.26-495.76 MW

The averages are taken from various sensors located around the plant that record the ambient variables every second. The variables are given without normalization.

Dataset Information:

The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load. Features consist of hourly average ambient variables Temperature (T), Ambient Pressure (AP),

Relative Humidity (RH) and Exhaust Vacuum (V) to predict the net hourly electrical energy output (EP) of the plant.

A combined cycle power plant (CCPP) is composed of gas turbines (GT), steam turbines (ST) and heat recovery steam generators. In a CCPP, the electricity is generated by gas and steam turbines, which are combined in one cycle, and is transferred from one turbine to another. While the Vacuum is collected from and has effect on the Steam Turbine, the other three of the ambient variables effect the GT performance.

Steps:

1. Load the data

```
val df = sqlContext.read.format("csv")
    .option("header","true")
    .option("inferSchema","true")
    .load("/data/Combined_Cycle_Power_Plant/")
df.printSchema
```

```
df: org.apache.spark.sql.DataFrame = [AT: double, V: double, AP: double, RH: double, PE: double]
root
| -- AT: double (nullable = true)
| -- V: double (nullable = true)
| -- AP: double (nullable = true)
| -- RH: double (nullable = true)
| -- PE: double (nullable = true)
```

2. Cache the dataframe, as it will be used many time.

```
df.cache
df.count
df.show(5)
```

```
res59: df.type = [AT: double, V: double, AP: double, RH: double, PE: double]
res60: Long = 47840
+---+---+---+---+
|   AT|     V|     AP|     RH|     PE|
+---+---+---+---+
|14.96|41.76|1024.07|73.17|463.26|
|25.18|62.96|1020.04|59.08|444.37|
| 5.11| 39.4|1012.16|92.14|488.56|
|20.86|57.32|1010.24|76.64|446.48|
|10.82| 37.5|1009.23|96.62| 473.9|
+---+---+---+---+
only showing top 5 rows
```

3. Register the dataframe as table

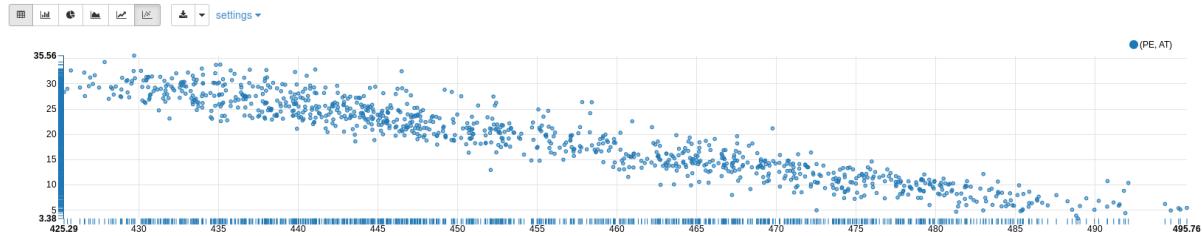
```
df.registerTempTable("pp")
```

4. Query the table using SQL interpreter from zeppelin.

```
%sql  
SELECT PE, AT from pp
```

PE	AT
463.26	14.96
444.37	25.18
488.56	5.11
446.48	20.86
473.9	10.82
443.67	26.27
467.35	15.89
478.42	9.48

To view the output graphically

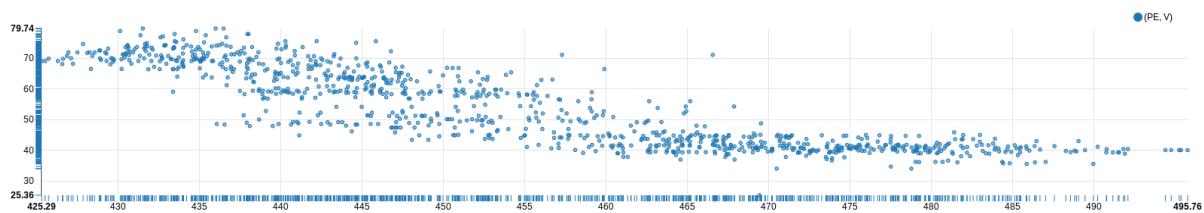


5. Query the table for analysis between PE vs V

```
%sql  
SELECT PE, V FROM pp
```

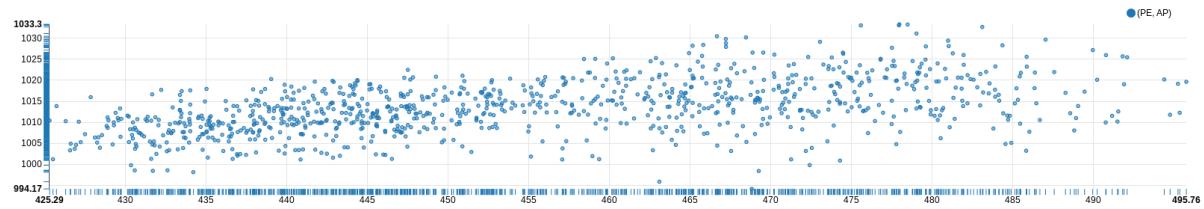
PE	V
463.26	41.76
444.37	62.96
488.56	39.4
446.48	57.32
473.9	37.5
443.67	59.44
467.35	43.96
478.42	44.71
475.98	45

Results are limited by 1000.



6. Analysis between PE and AP

```
%sql
SELECT PE, AP FROM pp
```



7. Analysis between PE and RH

```
%sql
SELECT PE, RH FROM pp
```

PE	RH
463.26	73.17
444.37	59.08
488.56	92.14
446.48	76.64
473.9	96.62
443.67	58.77
467.35	75.24

8. Data preparation

- VectorAssembler is a transformer that combines a given list of columns into a single vector column.
- Convert the `power_plant` SQL table into a DataFrame named `dataset`
- Set the vectorizer's input columns to a list of the four columns of the input DataFrame: `["AT", "V", "AP", "RH"]`
- Set the vectorizer's output column name to `"features"`

```
import org.apache.spark.ml.feature.VectorAssembler
val vectorizer = new VectorAssembler()
vectorizer.setInputCols(Array("AT", "V", "AP", "RH"))
vectorizer.setOutputCol("features")
```

```
vectorizer: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_c81487e45a35
res66: vectorizer.type = vecAssembler_c81487e45a35
res67: vectorizer.type = vecAssembler_c81487e45a35
```

9. Data Modeling

- Split the data into training and test datasets, i.e. 20% for testing and 80% for training.

Objective:

We need a way of evaluating how well our linear regression model predicts power output as a function of input parameters. We can do this by splitting up our initial data set into a _Training Set_ used to train our model and a _Test Set_ used to evaluate the model's performance in giving predictions.

```
var Array(testingSet, trainingSet) = df.randomSplit(Array(0.20,
0.80), 0L)
```

```
testingSet.count  
trainingSet.count
```

```
testingSet: org.apache.spark.sql.DataFrame = [AT: double, V: double, AP: double, RH: double, PE: double]  
trainingSet: org.apache.spark.sql.DataFrame = [AT: double, V: double, AP: double, RH: double, PE: double]  
res69: Long = 9547  
res70: Long = 38293
```

Linear Regression Model

We'll create a Linear Regression Model and use the built in help to identify how to train it.

- Initialize our linear regression learner.
- Explain params to dump the parameters we can use.

```
import org.apache.spark.ml.regression.LinearRegression  
  
import org.apache.spark.ml.regression.LinearRegressionModel  
  
import org.apache.spark.ml.Pipeline  
  
val lr = new LinearRegression()  
  
lr.explainParams()
```

```
import org.apache.spark.ml.regression.LinearRegression  
import org.apache.spark.ml.regression.LinearRegressionModel  
import org.apache.spark.ml.Pipeline  
lr: org.apache.spark.ml.regression.LinearRegression = linReg_301e5ead4215  
res72: String =  
  elasticNetParam: the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty (default: 0.0)  
  featuresCol: features column name (default: features)  
  fitIntercept: whether to fit an intercept term (default: true)  
  labelCol: label column name (default: label)  
  maxIter: maximum number of iterations (>= 0) (default: 100)  
  predictionCol: prediction column name (default: prediction)  
  regParam: regularization parameter (>= 0) (default: 0.0)  
  solver: the solver algorithm for optimization. If this is not set or empty, default value is 'auto'. (default: auto)  
  standardization: whether to standardize the training features before fitting the model (default: true)  
  tol: the convergence tolerance for iterative algorithms (de...
```

10. Set the parameters for the method:

- Set the name of the prediction column to "Predicted_PE"
- Set the name of the label column to "PE"
- Set the maximum number of iterations to 100
- Set the regularization parameter to 0.1

```
lr.setPredictionCol("PE_Predict")  
.setLabelCol("PE")
```

```
.setMaxIter(100)  
.setRegParam(0.1)
```

```
res74: org.apache.spark.ml.regression.LinearRegression = linReg_301e5ead4215
```

11. Create a model by training on ‘trainingSet’

- Use the new spark.ml pipeline API, Similar to python scikit-learn
- First train on the entire dataset to see what we get

```
val lrPipeline = new Pipeline()  
lrPipeline.setStages(Array(vectorizer, lr))  
val lrModel = lrPipeline.fit(trainingSet)
```

```
lrPipeline: org.apache.spark.ml.Pipeline = pipeline_5a953dde602d  
res76: lrPipeline.type = pipeline_5a953dde602d  
lrModel: org.apache.spark.ml.PipelineModel = pipeline_5a953dde602d
```

12. Linear Regression Model

Linear Regression is simply a Line of best fit over the data that minimizes the square of the error, given multiple input dimensions we can express each predictor as a line function of the form:

[$y = a + b x_1 + b x_2 + b x_i \dots \backslash\backslash$]

where $\backslash\backslash(a\backslash\backslash)$ is the intercept and the $\backslash\backslash(b\backslash\backslash)$ are the coefficients.

```
val model = lrModel.stages(1).asInstanceOf[LinearRegressionModel]
```

```
model: org.apache.spark.ml.regression.LinearRegressionModel = linReg_301e5ead4215
```

13. To express the coefficients of that line we can retrieve the Estimator stage from the PipelineModel and express the weights and the intercept for the function.

- Intercepts

```
val intercept = model.intercept
```

```
intercept: Double = 437.3755581407379
```

- Weights

```
val weights = model.weights.toArray
```

```
warning: there were 1 deprecation warning(s); re-run with -deprecation for details
weights: Array[Double] = Array(-1.9184240575231875, -0.2543479937050499, 0.07825010764537191, -0.1471027706567747)
```

14. List of the column names (without PE)

```
val featuresNoLabel = df.columns.filter(col => col != "PE")
```

```
featuresNoLabel: Array[String] = Array(AT, V, AP, RH)
```

15. Merge the weights and labels

```
val coefficents =
sc.parallelize(weights).zip(sc.parallelize(featuresNoLabel))
coefficents.collect.foreach(println)
```

```
coefficents: org.apache.spark.rdd.RDD[(Double, String)] = ZippedPartitionsRDD2[116] at zip at <console>:87
(-1.9184240575231875,AT)
(-0.2543479937050499,V)
(0.07825010764537191,AP)
(-0.1471027706567747,RH)
```

16. Assigning value of intercept to equation

```
var equation = s"y = $intercept "
```

```
equation: String = "y = 437.3755581407379 "
```

17. Sort the coefficients from greatest absolute weight most to the least absolute weight

```
var variables = Array  
coefficents.sortByKey().collect().foreach(x =>  
{  
    val weight = Math.abs(x._1)  
    val name = x._2  
    val symbol = if (x._1 > 0) "+" else "-"  
    equation += (s" $symbol ${weight} * ${name})")  
}  
)
```

```
variables: Array.type = scala.Array@55b81d1f
```

18. Linear Regression Equation

```
println("Linear Regression Equation: " + equation)
```

```
Linear Regression Equation: y = 437.3755581407379 - (1.9184240575231875 * AT) - (0.2543479937050499 * V) - (0.1471027706567747 * RH) + (0.07825010764537191 * AP)
```

19. Apply our LR model to the test data and predict power output

```
val predictionsAndLabels = lrModel.transform(testingSet)  
predictionsAndLabels.show(5)
```

Scroll through the resulting table and notice how the values in the Power Output (PE) column compare to the corresponding values in the predicted Power Output (Predicted_PE) column.

```

predictionsAndLabels: org.apache.spark.sql.DataFrame = [AT: double, V: double, AP: double, RH: double, PE: double, features: vector, PE_Predict: double]
+---+-----+-----+-----+-----+-----+
| AT|     V|    AP|    RH|      PE|      features|      PE_Predict|
+---+-----+-----+-----+-----+-----+
|3.38|41.31| 998.79|97.76|489.11|[3.38,41.31,998.7...| 484.1588273620686|
| 3.6|35.19|1018.73| 99.1|488.98|[3.6,35.19,1018.7...|486.65657322465705|
|3.73|39.42| 1024.4|82.42|488.58|[3.73,39.42,1024....|488.22863840871094|
|3.74|35.19|1018.58|98.84| 490.5|[3.74,35.19,1018....|486.41450306082777|
|3.95|38.44|1016.75| 79.65|492.46|[3.95,38.44,1016....|487.86470750111897|
+---+-----+-----+-----+-----+-----+
only showing top 5 rows

```

20. Compute an evaluation metric for our test dataset

- Create an RMSE evaluator using the label and predicted columns
- Run the evaluator on the DataFrame
- Calculating R squared value.

Note: Another useful statistical evaluation metric is the coefficient of determination, denoted R^2 or r^2 and pronounced "R squared".

It is a number that indicates the proportion of the variance in the dependent variable that is predictable from the independent variable and it provides a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model. The coefficient of determination ranges from 0 to 1 (closer to 1), and the higher the value, the better our model.

```

import org.apache.spark.mllib.evaluation.RegressionMetrics

val metrics = new
RegressionMetrics(predictionsAndLabels.select("PE_Predict",
"PE").rdd.map(r => (r(0).asInstanceOf[Double],
r(1).asInstanceOf[Double])))

val rmse = metrics.rootMeanSquaredError

val explainedVariance = metrics.explainedVariance

val r2 = metrics.r2

println (f"Root Mean Squared Error: $rmse")

println (f"Explained Variance: $explainedVariance")

println (f"R2: $r2")

```

```

import org.apache.spark.mllib.evaluation.RegressionMetrics
metrics: org.apache.spark.mllib.evaluation.RegressionMetrics = org.apache.spark.mllib.evaluation.RegressionMetrics@f53cfef
rmse: Double = 4.596932972254193
explainedVariance: Double = 267.5580020580147
r2: Double = 0.9275409236725611
Root Mean Squared Error: 4.596932972254193
Explained Variance: 267.5580020580147
R2: 0.9275409236725611

```

Note: Generally, a good model will have 68% of predictions within 1 RMSE and 95% within 2 RMSE of the actual value. Let's calculate and see if a RMSE of 4.51 meets this criterion.

Tuning and Evaluation

Now that we have a model with all of the data let's try to make a better model by tuning over several parameters to see if we can get better results.

```
import org.apache.spark.ml.tuning.{ParamGridBuilder, CrossValidator}  
import org.apache.spark.ml.evaluation._  
  
import org.apache.spark.ml.tuning.{ParamGridBuilder, CrossValidator}  
import org.apache.spark.ml.evaluation._
```

21. Use a cross validator to split the data into training and validation subsets

Let's set up our evaluator class to judge the model based on the best root mean squared error

```
val regEval = new RegressionEvaluator()  
regEval.setLabelCol("PE")  
    .setPredictionCol("PE_Predict")  
    .setMetricName("rmse")
```

```
regEval: org.apache.spark.ml.evaluation.RegressionEvaluator = regEval_6eb0d0675120  
res98: regEval.type = regEval_6eb0d0675120
```

22. Create our crossvalidator with 3 fold cross validation

```
val crossval = new CrossValidator()  
crossval.setEstimator(lrPipeline)  
crossval.setNumFolds(5)  
crossval.setEvaluator(regEval)
```

```
crossval: org.apache.spark.ml.tuning.CrossValidator = cv_a61d2e94671b
res101: crossval.type = cv_a61d2e94671b
res102: crossval.type = cv_a61d2e94671b
res103: crossval.type = cv_a61d2e94671b
```

23. Tune over our regularization parameter from 0.01 to 0.10

```
val regParam = ((1 to 10) toArray).map(x => (x /100.0))
```

```
warning: there were 1 feature warning(s); re-run with -feature for details
regParam: Array[Double] = Array(0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1)
```

24. Create a parameter grid using the ParamGridBuilder, and add the grid to the CrossValidator.

```
val paramGrid = new ParamGridBuilder()
  .addGrid(lr.regParam, regParam)
  .build()
crossval.setEstimatorParamMaps(paramGrid)
```

```
paramGrid: Array[org.apache.spark.ml.param.ParamMap] =
Array({
  linReg_301e5ead4215-regParam: 0.01
}, {
  linReg_301e5ead4215-regParam: 0.02
}, {
  linReg_301e5ead4215-regParam: 0.03
}, {
  linReg_301e5ead4215-regParam: 0.04
}, {
  linReg_301e5ead4215-regParam: 0.05
}, {
  linReg_301e5ead4215-regParam: 0.06
}, {
  linReg_301e5ead4215-regParam: 0.07
}, {
  linReg_301e5ead4215-regParam: 0.08
}, {
  linReg_301e5ead4215-regParam: 0.09
}, {
  linReg_301e5ead4215-regParam: 0.1
})
```

25. Let's find and return the best model

```
val cvModel = crossval.fit(trainingSet)
```

```
cvModel: org.apache.spark.ml.tuning.CrossValidatorModel = cv_a61d2e94671b
```

26. We have tuned let's see what we got for tuning parameters and what our RMSE was versus our initial model.

- Use cvModel to compute an evaluation metric for our test dataset: testingSet

```
val predictionsAndLabels = cvModel.transform(testingSet)

val metrics = new
RegressionMetrics(predictionsAndLabels.select("PE_Predict",
"PE").rdd.map(r => (r(0).asInstanceOf[Double],
r(1).asInstanceOf[Double])))
```

```
predictionsAndLabels: org.apache.spark.sql.DataFrame = [AT: double, V: double, AP: double, RH: double, PE: double, features: vector, PE_Predict: double]
metrics: org.apache.spark.mllib.evaluation.RegressionMetrics = org.apache.spark.mllib.evaluation.RegressionMetrics@1a3b890e
```

27. Run the previously created RMSE evaluator, on the predictionsAndLabels DataFrame

- compute the r2 evaluation metric for our test dataset

```
val rmse = metrics.rootMeanSquaredError
val explainedVariance = metrics.explainedVariance
val r2 = metrics.r2

println (f"Root Mean Squared Error: $rmse")
println (f"Explained Variance: $explainedVariance")
println (f"R2: $r2")
```

```
rmse: Double = 4.593657981825083
explainedVariance: Double = 269.9212285609958
r2: Double = 0.9276441308553334
Root Mean Squared Error: 4.593657981825083
Explained Variance: 269.9212285609958
R2: 0.9276441308553334
```

So our initial untuned and tuned linear regression models are statistically identical.

Regression Tree

Given that the only linearly correlated variable is Temperature, it makes sense try another machine learning method such a Decision Tree to handle non-linear data and see if we can improve our model

A Decision Tree creates a model based on splitting variables using a tree structure. We will first start with a single decision tree model.

1. Create a DecisionTreeRegressor

```
import org.apache.spark.ml.regression.DecisionTreeRegressor

val dt = new DecisionTreeRegressor()
dt.setLabelCol("PE")
dt.setPredictionCol("PE_Predict")
dt.setFeaturesCol("features")
dt.setMaxBins(100)
```

```
import org.apache.spark.ml.regression.DecisionTreeRegressor
dt: org.apache.spark.ml.regression.DecisionTreeRegressor = dtr_8585361dd982
res117: org.apache.spark.ml.regression.DecisionTreeRegressor = dtr_8585361dd982
res118: org.apache.spark.ml.regression.DecisionTreeRegressor = dtr_8585361dd982
res119: org.apache.spark.ml.regression.DecisionTreeRegressor = dtr_8585361dd982
res120: dt.type = dtr_8585361dd982
```

2. Create a Pipeline and set the stages of the pipeline

```
val dtPipeline = new Pipeline()
dtPipeline.setStages(Array(vectorizer, dt))
```

3. Let's reuse our CrossValidator

- Use [ParamGridBuilder] to build a parameter grid with the parameter `dt.maxDepth` and a list of the values 2 and 3, and add the grid to the [CrossValidator].

- Run the [CrossValidator] to find the parameters that yield the best model (i.e. lowest RMSE) and return the best model.

```

crossval.setEstimator(dtPipeline)

val paramGrid = new ParamGridBuilder()
  .addGrid(dt.maxDepth, Array(2, 3))
  .build()

crossval.setEstimatorParamMaps(paramGrid)

val dtModel = crossval.fit(trainingSet)

```

```

dtPipeline: org.apache.spark.ml.Pipeline = pipeline_78ed1072fa9d
res122: dtPipeline.type = pipeline_78ed1072fa9d
res123: crossval.type = cv_a61d2e94671b
paramGrid: Array[org.apache.spark.ml.param.ParamMap] =
  Array({
    dtr_8585361dd982-maxDepth: 2
  }, {
    dtr_8585361dd982-maxDepth: 3
})
res124: crossval.type = cv_a61d2e94671b
dtModel: org.apache.spark.ml.tuning.CrossValidatorModel = cv_a61d2e94671b

```

Let's see how our tuned DecisionTreeRegressor model's RMSE and $\sqrt{r^2}$ values compare to our tuned LinearRegression model.

```

import org.apache.spark.ml.regression.DecisionTreeRegressionModel
import org.apache.spark.ml.PipelineModel

val predictionsAndLabels = dtModel.bestModel.transform(testingSet)

val metrics = new
  RegressionMetrics(predictionsAndLabels.select("PE_Predict",
    "PE").map(r => (r(0).asInstanceOf[Double],
    r(1).asInstanceOf[Double])))

val rmse = metrics.rootMeanSquaredError

```

```

val explainedVariance = metrics.explainedVariance
val r2 = metrics.r2

println (f"Root Mean Squared Error: $rmse")
println (f"Explained Variance: $explainedVariance")
println (f"R2: $r2")

```

```

import org.apache.spark.ml.regression.DecisionTreeRegressionModel
import org.apache.spark.ml.PipelineModel
predictionsAndLabels: org.apache.spark.sql.DataFrame = [AT: double, V: double, AP: double, RH: double, PE: double, features: vector, PE_Predict: double]
metrics: org.apache.spark.mllib.evaluation.RegressionMetrics = org.apache.spark.mllib.evaluation.RegressionMetrics@54e2ea3
rmse: Double = 5.169951561569703
explainedVariance: Double = 264.7744220845704
r2: Double = 0.9083506479156687
Root Mean Squared Error: 5.169951561569703
Explained Variance: 264.7744220845704
R2: 0.9083506479156687

```

This line will pull the Decision Tree model from the Pipeline as display it as an if-then-else string

```

dtModel.bestModel.asInstanceOf[PipelineModel].stages.last.asInstanceOf[DecisionTreeRegressionModel].toDebugString

```

```

res132: String =
"DecisionTreeRegressionModel (uid=dtr_8585361dd982) of depth 3 with 15 nodes
  If (feature 0 <= 17.85)
    If (feature 0 <= 11.67)
      If (feature 0 <= 8.63)
        Predict: 483.79243774574053
      Else (feature 0 > 8.63)
        Predict: 476.3108737113401
    Else (feature 0 > 11.67)
      If (feature 0 <= 14.35)
        Predict: 469.0458953626635
      Else (feature 0 > 14.35)
        Predict: 462.1050880962613
    Else (feature 0 > 17.85)
      If (feature 0 <= 22.99)
        If (feature 1 <= 45.87)
          Predict: 458.12494773519165

```

So our DecisionTree has slightly worse RMSE than our LinearRegression model (LR: 4.59 vs DT: 5.19).

Exercise10: Machine Learning

To configure PySpark in Jupyter notebook

Open the new command terminal and export the below commands.

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

Now type in the command terminal

```
pyspark --packages com.databricks:spark-csv_2.10:1.0.3
```

This will open pyspark in Jupyter notebook.

Start exploring as per the instructions.

To perform below exercise, install R-studio from ‘software’ folder or open R console and run below commands from command line.

Alternatively, while dealing with huge datasets we will use ‘Zeppelin’ with R interpreter pre-installed on the machine.

Statistical analysis in R is performed by using many in-built functions.

Functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.

1. Mean

From R console, use function mean() to calculate mean.

```
# Create a vector.  
  
x <- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5)  
  
# Find Mean.  
  
result.mean <- mean(x)  
  
print(result.mean)
```

```
> x <- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5)  
> result.mean <- mean(x)  
> print(result.mean)  
[1] 8.22
```

2. Median

Similarly, to calculate median

```
# Create the vector.  
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)  
# Find the median.  
median.result <- median(x)  
print(median.result)
```

```
> median.result <- median(x)  
> print(median.result)  
[1] 5.6  
> |
```

Linear Regression

Exercise9.1- Using R-studio/ R-console

Example: Predicting the weight of a person when his height is known.

Steps:

1. Gather the sample of observed values of height and corresponding weight.

Input:

```
# Values of height  
151, 174, 138, 186, 128, 136, 179, 163, 152, 131  
# Values of weight.  
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

2. Create relationship model using **lm()** function.

Find the coefficients from the model and create the equation.

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)  
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)  
# Apply the lm() function.  
relation <- lm(y~x)  
print(relation)
```

```

> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
> relation <- lm(y~x)
> print(relation)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)          x
-38.4551        0.6746

```

3. Get summary of the model to know average error i.e. **residuals**.

```

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
print(summary(relation))

```

```

> relation <- lm(y~x)
> print(summary(relation))

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-6.3002 -1.6629  0.0412  1.8944  3.9775 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -38.45509   8.04901 -4.778  0.00139 ***
x            0.67461   0.05191 12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548, Adjusted R-squared:  0.9491 
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06

```

4. To predict the weight of new person use **predict()** function.

```

# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
# The response vector.

```

```

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

# Find weight of a person with height 170.

a <- data.frame(x = 170)

result <- predict(relation,a)

print(result)

```

```

> relation <- lm(y~x)
> a <- data.frame(x = 170)
> result <- predict(relation,a) .
> print(result)
    1
76.22869
> |

```

5. Visualize the regression graphically

```

# Create the predictor and response variable.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

relation <- lm(y~x)

# Give the chart file a name.

png(file = "linearregression.png")

# Plot the chart.

plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab =
"Height in cm")

# Save the file.

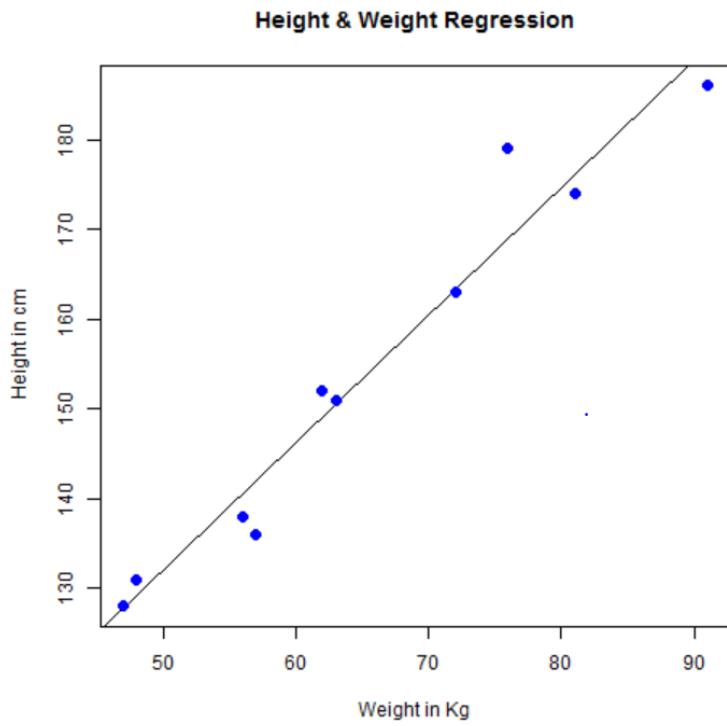
dev.off()

```

```

> relation <- lm(y~x)
> png(file = "linearregression.png")
> plot(y,x,col = "blue",main = "Height & Weight Regression",abline(lm(x~y)),cex = 1.3,pch = 16,xlab
= "Weight in Kg",ylab = "Height in cm")
> dev.off()
null device
      1
> |

```



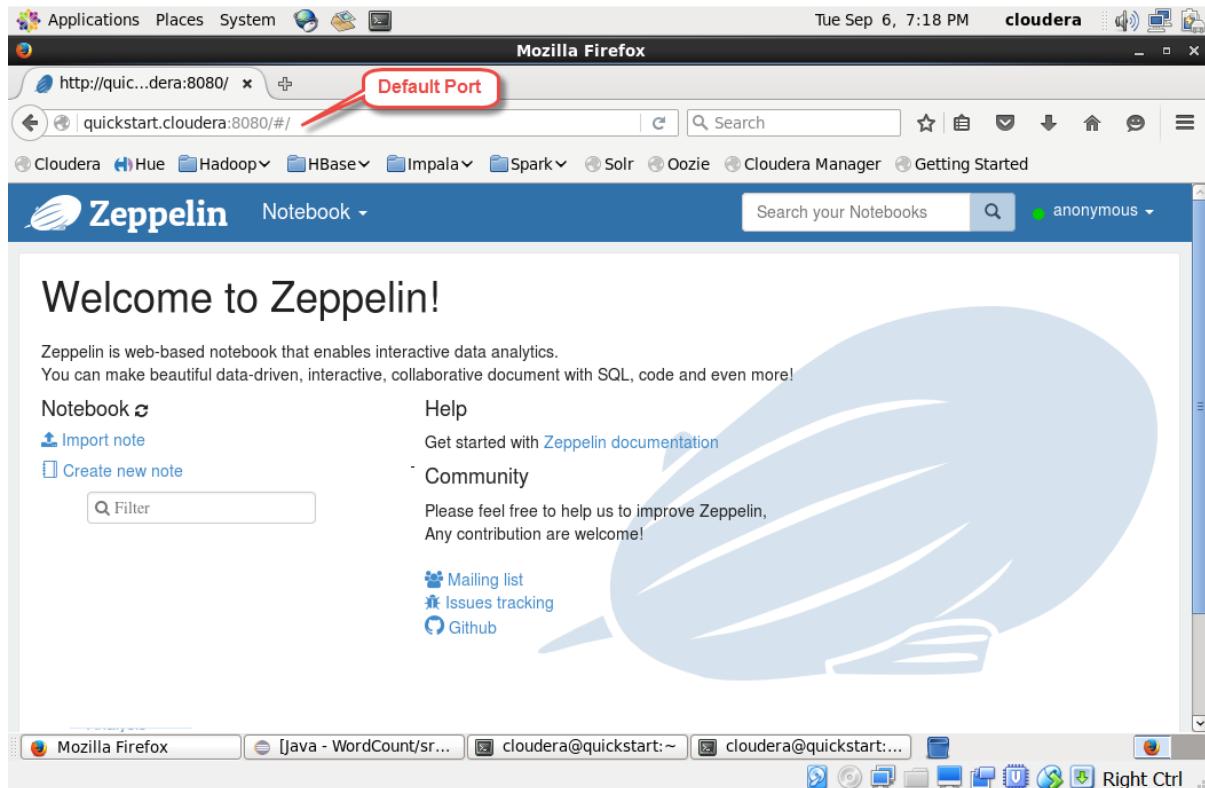
[Exercise9.2](#) Perform below exercise using zeppelin notebook

```
To start Zeppelin service
cd /usr/lib/zeppelin-0.6.0-bin-all
sudo bin/zeppelin-daemon.sh start
```

```
[cloudera@quickstart ~]$ cd /usr/lib/zeppelin-0.6.0-bin-all/
[cloudera@quickstart zeppelin-0.6.0-bin-all]$ ls bin
common.cmd      functions.sh          interpreter.sh      zeppelin.sh
common.sh       install-interpreter.sh  zeppelin.cmd
functions.cmd   interpreter.cmd       zeppelin-daemon.sh
[cloudera@quickstart zeppelin-0.6.0-bin-all]$ bin/zeppelin-daemon.sh start
Zeppelin start                                [ OK ]
[cloudera@quickstart zeppelin-0.6.0-bin-all]$ █
```

Accessing Zeppelin UI

Open browser and use port 8080 to access Zeppelin notebook <quickstart.cloudera:8080>



Goal:

1. Start with importing the required package for R

```
%r require(ggplot2)
```

2. Load the data and list the features

```
%r
power_plant = read.table("/data/CCPP/Folds5x2_pp_1.csv", header =
TRUE, sep = ",")
str(power_plant)

# Features consist of hourly average ambient variables
# Temperature (T) in the range 1.81°C and 37.11°C,
# Ambient Pressure (AP) in the range 992.89-1033.30 milibar,
# Relative Humidity (RH) in the range 25.56% to 100.16%
```

```
# Exhaust Vacuum (V) in teh range 25.36-81.56 cm Hg  
# Net hourly electrical energy output (EP) 420.26-495.76 MW
```

```
'data.frame': 9568 obs. of 5 variables:  
 $ AT: num 14.96 25.18 5.11 20.86 10.82 ...  
 $ V : num 41.8 63 39.4 57.3 37.5 ...  
 $ AP: num 1024 1020 1012 1010 1009 ...  
 $ RH: num 73.2 59.1 92.1 76.6 96.6 ...  
 $ PE: num 463 444 489 446 474 ...
```

3. To check first few entries

```
%r head(power_plant)
```

```
1 14.96 41.76 1024.07 73.17 463.26  
2 25.18 62.96 1020.04 59.08 444.37  
3 5.11 39.40 1012.16 92.14 488.56  
4 20.86 57.32 1010.24 76.64 446.48  
5 10.82 37.50 1009.23 96.62 473.90  
6 26.27 59.44 1012.23 58.77 443.67
```

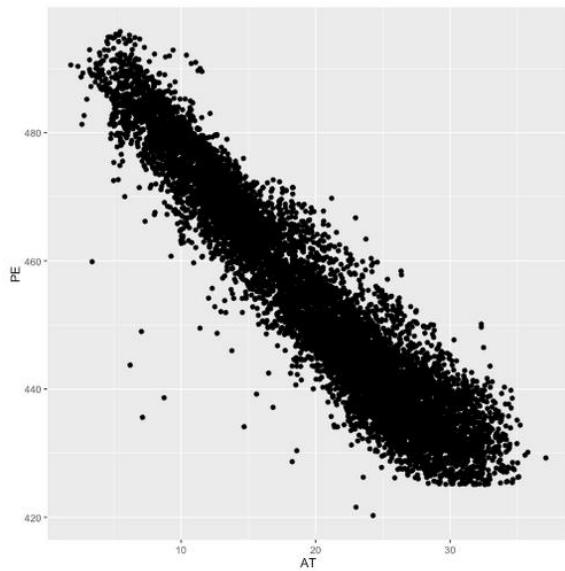
4. To check the summary of the dataset

```
%r summary(power_plant)
```

```
Min.   : 1.81   Min.   :25.36   Min.   : 992.9   Min.   : 25.56<br />  
1st Qu.:13.51   1st Qu.:41.74   1st Qu.:1009.1   1st Qu.: 63.33<br />  
Median :20.34   Median :52.08   Median :1012.9   Median : 74.97<br />  
Mean   :19.65   Mean   :54.31   Mean   :1013.3   Mean   : 73.31<br />  
3rd Qu.:25.72   3rd Qu.:66.54   3rd Qu.:1017.3   3rd Qu.: 84.83<br />  
Max.   :37.11   Max.   :81.56   Max.   :1033.3   Max.   :100.16<br />  
PE<br />  
Min.   :420.3<br />  
1st Qu.:439.8<br />  
Median :451.6<br />  
Mean   :454.4<br />  
3rd Qu.:468.4<br />  
Max.   :495.8
```

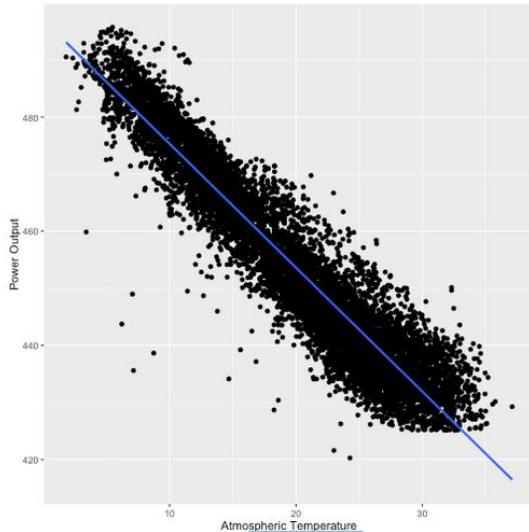
5. Plot between AT and PE

```
%r  
ggplot(power_plant, aes(x = AT, y = PE)) +  
  geom_point()
```



6. Another way of representation

```
%r  
ggplot(power_plant, aes(x = AT, y = PE)) +  
  geom_point() +  
  geom_smooth(method = "lm") +  
  labs(x = "Atmospheric Temperature", y = "Power Output")
```



Linear Regression Model

```
%r
lm.fit = lm(PE ~ AT, data = power_plant)
#Statistical Significance of the Model
capture.output(summary(lm.fit))
```

```
[2] "Call:"<br />
[3] "lm(formula = PE ~ AT, data = power_plant)"<br />
[4] "<br />
[5] "Residuals:"<br />
[6] "    Min      1Q  Median      3Q     Max "<br />
[7] "-45.951 -3.644   0.101   3.696  23.251 "<br />
[8] "<br />
[9] "Coefficients:"<br />
[10] "              Estimate Std. Error t value Pr(>|t|)    "<br />
[11] "(Intercept) 497.034120   0.156434 3177.3 <2e-16 ***"<br />
[12] "AT          -2.171320   0.007443  -291.7 <2e-16 ***"<br />
[13] "-<br />
[14] "Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1"
[15] "<br />
[16] "Residual standard error: 5.426 on 9566 degrees of freedom"<br />
[17] "Multiple R-squared:  0.8989, Adjusted R-squared:  0.8989 "<br />
[18] "F-statistic: 8.51e+04 on 1 and 9566 DF,  p-value: < 2.2e-16"<br />
[...]
```

Interpretation

Residuals:

The residuals are the difference between the actual values of the variable you're predicting and predicted values from your regression— $y - \hat{y}$. For most regressions

you want your residuals to look like a normal distribution when plotted. If our residuals are normally distributed, this indicates the mean of the difference between our predictions and the actual values is close to 0 (good) and that when we miss, we're missing both short and long of the actual value, and the likelihood of a miss being far from the actual value gets smaller as the distance from the actual value gets larger.

Significance Stars

The stars are shorthand for significance levels, with the number of asterisks displayed according to the p-value computed. for high significance and * for low significance.

Estimated Coefficient

The estimated coefficient is the value of slope calculated by the regression. It might seem a little confusing that the Intercept also has a value, but just think of it as a slope that is always multiplied by 1.

Standard Error of the Coefficient Estimate

Measure of the variability in the estimate for the coefficient. Lower means better but this number is relative to the value of the coefficient. As a rule of thumb, you'd like this value to be at least an order of magnitude less than the coefficient estimate.

t-value or t-statistics

Score that measures whether or not the coefficient for this variable is meaningful for the model. You probably won't use this value itself, but know that it is used to calculate the p-value and the significance levels.

Significance Legend

The more punctuation there is next to your variables, the better.

Blank=bad,

Dots=pretty good

Stars=good

More Stars=very good

Residual Std Error / Degrees of Freedom

The Residual Std Error is just the standard deviation of your residuals. You'd like this number to be proportional to the quantiles of the residuals in #1. For a normal distribution, the 1st and 3rd quantiles should be $1.5 \pm$ the std error.

The Degrees of Freedom is the difference between the number of observations included in your training sample and the number of variables used in your model (intercept counts as a variable).

Variable p-value

Probability the variable is NOT relevant. You want this number to be as small as possible. If the number is really small, R will display it in scientific notation.

R-squared

Metric for evaluating the goodness of fit of your model. Higher is better with 1 being the best. An R² value close to 1 indicates that the model explains a large portion of the variance in the response variable.

WARNING: While a high R-squared indicates good correlation, correlation does not always imply causation.

R-square adjusted

R-square adjusted is penalized for having a large number of parameters in the model

$$R^2_{\text{adjusted}} = 1 - \frac{(1 - R^2)(n - 1)}{(n - p)}$$

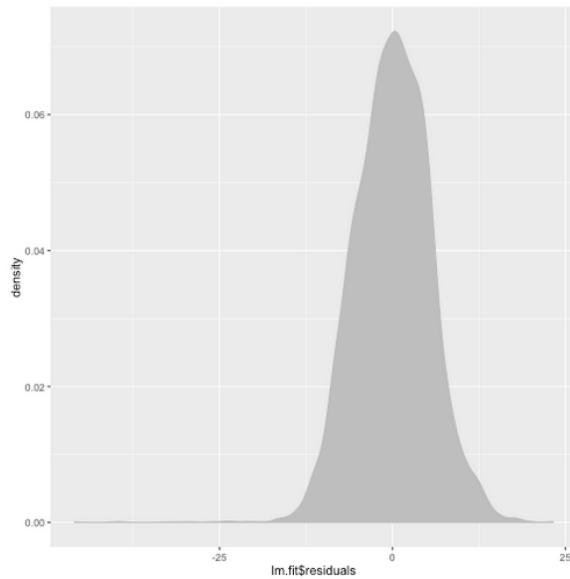
F-statistic & resulting p-value

$$F = (\text{explained variance}) / (\text{unexplained variance})$$

Performs an F-test on the model. This takes the parameters of our model (in our case we only have 1) and compares it to a model that has fewer parameters. In theory the model with more parameters should fit better. If the model with more parameters (your model) doesn't perform better than the model with fewer parameters, the F-test will have a high p-value (probability NOT significant boost). If the model with more parameters is better than the model with fewer parameters, you will have a lower p-value.

When there is no relationship between the response and predictors, one would expect the F-statistic to take on a value close to 1. Large F-statistic suggests that at least some predictors are related to the response variable.

```
%r ggplot() + geom_density(aes(x=lm.fit$residuals), fill = "grey", color = "grey")
```



```
%r #Residual standard error:  
sd(lm.fit$residuals)
```

```
[1] 5.425383
```

```
%r #R2:
```

```
R2 = sum((lm.fit$fitted.values - mean(power_plant$PE)) ^ 2) /  
sum((power_plant$PE - mean(power_plant$PE)) ^ 2)  
R2
```

[1] 0.8989476

```
%r #F-statistics  
  
# F = ((TSS - RSS) / p) / (RSS / (n - p - 1))  
# TSS = Total sum of square = sum((y - mean(y)) ^ 2)  
# RSS = Residual Sum of Square = sum((y_predicted - y) ^ 2)  
# p = no of predictors  
# n = number of observations  
  
p = 1  
TSS = sum((power_plant$PE - mean(power_plant$PE)) ^ 2)  
RSS = sum((lm.fit$fitted.values - power_plant$PE) ^ 2)  
  
n = 9568  
F = ((TSS - RSS) / p) / (RSS / (n - p - 1))  
F
```

[1] 85097.76

```
%r #R-squared Adjusted  
  
R2.adjusted = 1 - (1 - R2)*(n - 1) / (n - p)  
R2.adjusted
```

[1] 0.8989476

```
%r names(lm.fit)
```

```
[1] "coefficients"  "residuals"      "effects"        "rank"
[5] "fitted.values" "assign"         "qr"             "df.residual"
[9] "xlevels"       "call"          "terms"          "model"
```

```
%r coef(lm.fit)
```

```
(Intercept)      AT
497.03412    -2.17132
```

```
%r confint(lm.fit)
```

```
(Intercept) 496.72748 497.34076
AT           -2.18591  -2.15673
```

Logistic Regression

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1.

Example:

In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

```
# Select some columns form mtcars.

input <- mtcars[,c("am","cyl","hp","wt")]

print(head(input))
```

```

> input <- mtcars[,c("am","cyl","hp","wt")]
> print(head(input))
      am cyl  hp   wt
Mazda RX4     1   6 110 2.620
Mazda RX4 Wag 1   6 110 2.875
Datsun 710    1   4  93 2.320
Hornet 4 Drive 0   6 110 3.215
Hornet Sportabout 0   8 175 3.440
Valiant       0   6 105 3.460

```

Create Regression Model:

```

input <- mtcars[,c("am","cyl","hp","wt")]

am.data = glm(formula = am ~ cyl + hp + wt, data = input, family =
binomial)

print(summary(am.data))

```

```

> input <- mtcars[,c("am","cyl","hp","wt")]
> am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)
> print(summary(am.data))

Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

Deviance Residuals:
    Min      1Q      Median      3Q      Max 
-2.17272 -0.14907 -0.01464  0.14116  1.27641 

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) 19.70288   8.11637   2.428   0.0152 *  
cyl         0.48760   1.07162   0.455   0.6491    
hp          0.03259   0.01886   1.728   0.0840 .  
wt        -9.14947   4.15332  -2.203   0.0276 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 43.2297  on 31  degrees of freedom
Residual deviance: 9.8415  on 28  degrees of freedom
AIC: 17.841

Number of Fisher Scoring iterations: 8

```

Conclusion:

As the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

Decision Tree

Decision tree is a graph to represent choices and their results in form of a tree. The nodes in the graph represent an event or choice and the edges of the graph represent the decision rules or conditions.

The R package "**party**" is used to create decision trees.

```
install.packages("party")
```

Input Data

We will use the R in-built data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age", "shoe size", "score" and whether the person is a native speaker or not.

```
# Load the party package. It will automatically load other dependent
packages.

library(party)

# Print some records from data set readingSkills.

print(head(readingSkills))
```

```

> library(party)
Loading required package: grid
Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4
Loading required package: strucchange
Loading required package: zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

  as.Date, as.Date.numeric

Loading required package: sandwich
> print(head(readingsSkills))
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3           no   11 30.42170 49.60593
4          yes   7 28.66450 40.28456
5          yes   11 31.88207 55.46085
6          yes   10 30.07843 52.83124
> |

```

We will use the `ctree()` function to create the decision tree and see its graph.

```

# Load the party package. It will automatically load other dependent
packages.

library(party)

# Create the input data frame.

input.dat <- readingsSkills[c(1:105),]

# Give the chart file a name.

png(file = "decision_tree.png")

# Create the tree.

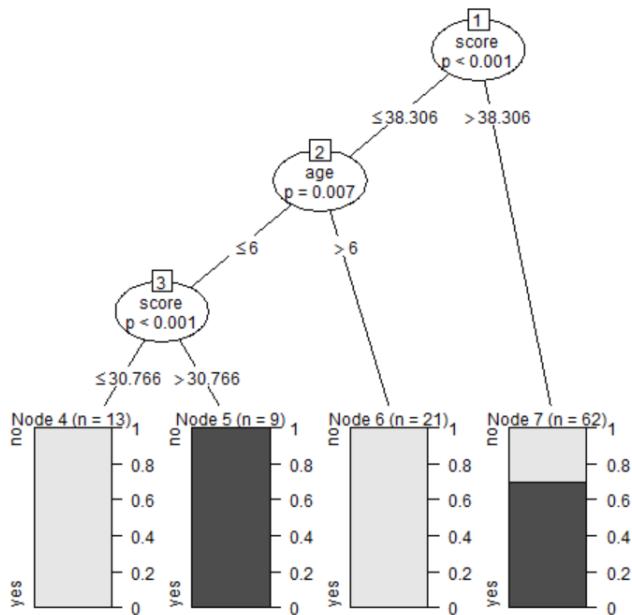
output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data = input.dat)

# Plot the tree.

plot(output.tree)

```

```
# Save the file.  
dev.off()
```



Conclusion

From the decision tree shown above we can conclude that anyone whose readingSkills score is less than 38.3 and age is more than 6 is not a native Speaker.

Random Forest

In the random forest approach, a large number of decision trees are created. Every observation is fed into every decision tree. The most common outcome for each observation is used as the final output. A new observation is fed into all the trees and taking a majority vote for each classification model.

An error estimate is made for the cases which were not used while building the tree. That is called an **OOB (Out-of-bag)** error estimate which is mentioned as a percentage.

The R package "**randomForest**" is used to create random forests.

```
install.packages("randomForest")
```

```
> install.packages("randomForest")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/randomForest_4.6-12.zip'
Content type 'application/zip' length 177129 bytes (172 KB)
downloaded 172 KB

package 'randomForest' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\technocrafty\AppData\Local\Temp\RtmpcJlGBQ\downloaded_packages
```

Input Data

We will use the R in-built data set named readingSkills to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age", "shoesize", "score" and whether the person is a native speaker.

```
# Load the party package. It will automatically load other required
packages.

library(party)

# Print some records from data set readingSkills.

print(head(readingSkills))
```

We will use the **randomForest()** function to create the decision tree and see it's graph.

```
# Load the party package. It will automatically load other required
packages.

library(party)

library(randomForest)

# Create the forest.

output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score,
                                data = readingSkills)

# View the forest results.
```

```

print(output.forest)

# Importance of each predictor.

print(importance(fit,type = 2))

> print(output.forest)

Call:
randomForest(formula = nativeSpeaker ~ age + shoeSize + score,      data = readingSkills)
              Type of random forest: classification
                      Number of trees: 500
No. of variables tried at each split: 1

          OOB estimate of  error rate: 1%
Confusion matrix:
      no yes class.error
no 99   1     0.01
yes 1  99     0.01
> |

```

Conclusion

From the random forest shown above we can conclude that the shoesize and score are the important factors deciding if someone is a native speaker or not. Also the model has only 1% error which means we can predict with 99% accuracy.

Classification

About Dataset: D2hawkEye

D2Hawkeye, Inc. develops data service solutions for the healthcare industry.

D2ReportManager that helps to design, distribute, and archive financial and clinical reports; D2BenefitAdvisor and D2Analyzer, on-line analytical processing tools that enable users to build data sets for data mining and ad hoc query analysis; D2Consulting, a medical and clinical advisory service; and D2Connect that provides e-business services, custom software development, and technology outsourcing. It also provides medical advisory, medical analytics, benchmarking, and other business services on an outsourced basis. The company serves health plans, third party administrators, case management and disease management industry, benefit consultants, physician groups, employers, and pharmaceutical industry, as well as re-insurers, MGUs, and stop-loss groups.

Mission

D2Hawkeye tries to improve healthcare case management

- Identify high-risk patients
- Work with patients to manage treatment and associated costs
- Arrange specialist care
- Medical costs often relate to severity of health problems, and are an issue for both patient and provider

Goal: Improve the quality of cost predictions

1. Load the data from local filesystem.

```
%r Claims = read.csv("data/ClaimsData.csv")
```

2. List the variables

```
%r str(Claims)
```

```
'data.frame': 458005 obs. of 16 variables:  
 $ age      : int 85 59 67 52 67 68 75 70 67 67 ...  
 $ alzheimers: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ arthritis: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ cancer   : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ copd     : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ depression: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ diabetes  : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ heart.failure: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ ihd      : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ kidney   : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ osteoporosis: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ stroke   : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ reimbursement2008: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ bucket2008 : int 1 1 1 1 1 1 1 1 1 1 ...  
 $ reimbursement2009: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ bucket2009 : int 1 1 1 1 1 1 1 1 1 1 ...
```

3. List the output of \$bucket2009

```
%r capture.output(base:::table(Claims$bucket2009))
```

```
[1] """<br />  
[2] "    1    2    3    4    5 "  
[3] "307444  87099  40976  19843  2643 "
```

Took a few seconds. Last updated by anonymous at September 06 2016, 10:32:33 PM.

4. View the summary of claims dataset

```
%r summary(Claims)
```

```

Min.   : 26.00  Min.   :0.0000  Min.   :0.0000  Min.   :0.0000<br />
1st Qu.: 67.00  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:0.0000<br />
Median : 73.00  Median :0.0000  Median :0.0000  Median :0.0000<br />
Mean   : 72.63  Mean   :0.1922  Mean   :0.1543  Mean   :0.06411<br />
3rd Qu.: 81.00  3rd Qu.:0.0000  3rd Qu.:0.0000  3rd Qu.:0.0000<br />
Max.   :100.00  Max.   :1.0000  Max.   :1.0000  Max.   :1.0000<br />
      copd      depression      diabetes      heart.failure<br />
Min.   :0.0000  Min.   :0.0000  Min.   :0.0000  Min.   :0.0000<br />
1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:0.0000<br />
Median :0.0000  Median :0.0000  Median :0.0000  Median :0.0000<br />
Mean   :0.1361  Mean   :0.2131  Mean   :0.3805  Mean   :0.2847<br />
3rd Qu.:0.0000  3rd Qu.:0.0000  3rd Qu.:1.0000  3rd Qu.:1.0000<br />
Max.   :1.0000  Max.   :1.0000  Max.   :1.0000  Max.   :1.0000<br />
      ind      kidney      osteoporosis      stroke<br />
Min.   :0.0000  Min.   :0.0000  Min.   :0.0000  Min.   :0.0000<br />
1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:0.0000<br />
Median :0.0000  Median :0.0000  Median :0.0000  Median :0.0000<br />

```

5. Percentage of patients in each cost bucket

```
%r base:::table(Claims$bucket2009)/nrow(Claims)
```

```
0.671267781 0.190170413 0.089466272 0.043324855 0.005770679
```

6. Model for healthcare cost prediction

Goal: To predict the cost bucket the patient fell into in 2009 using the CART model.

```
%r require(caTools) #For splitting data into training and test sets
```

7. Split the entire data into training and test sets, with 70/30 split.

```
%r
set.seed(88)
split = sample.split(Claims$bucket2009, SplitRatio = 0.7)
ClaimsTrain = subset(Claims, split == TRUE)
ClaimsTest = subset(Claims, split == FALSE)
Summary(ClaimsTrain)
```

What is the average age of patients in the training set, ClaimsTrain?

```
mean(ClaimsTrain$age)
```

Baseline Method and Penalty Matrix

The baseline method would predict that the cost bucket for a patient in 2009 will be the same as it was in 2008.

So let's create a classification matrix to compute the accuracy for the baseline method on the test set.

The accuracy is the sum of the diagonal, the observations that were classified correctly, divided by the total number of observations in our test set.

```
%r
```

```
confMatrix = base:::table(ClaimsTest$bucket2009, ClaimsTest$bucket2008)  
confMatrix
```

```
1 110138  7787  3427  1452   174  
2 16000   10721  4629  2931   559  
3 7006    4629  2774  1621   360  
4 2688    1943  1415  1539   352  
5 293     191   160   309   104
```

```
%r diag(confMatrix)
```

```
110138 10721 2774 1539 104
```

8. Baseline accuracy on test set

```
%r
```

```
sum(diag(confMatrix)) / nrow(ClaimsTest)
```

```
[1] 0.6838135
```

9. Define a “penaltymatrix” as the cost of being wrong

```
%r
```

```
PenaltyMatrix =  
matrix(c(0,1,2,3,4,2,0,1,2,3,4,2,0,1,2,6,4,2,0,1,8,6,4,2,0), byrow = TRUE,  
nrow = 5)
```

```
PenaltyMatrix
```

```
# X axis corresponds to the predictions and Y axis to the actuals
```

```
[1,]  0  1  2  3  4  
[2,]  2  0  1  2  3  
[3,]  4  2  0  1  2  
[4,]  6  4  2  0  1  
[5,]  8  6  4  2  0
```

10. Penalty error of Baseline Method

```
%r
penaltyErrors = as.matrix(base::table(ClaimsTest$bucket2009,
ClaimsTest$bucket2008)) * PenaltyMatrix
penaltyErrors
```

```
1     0 7787 6854 4356  696
2 32000     0 4629 5862 1677
3 28024 9258     0 1621  720
4 16128 7772 2830     0  352
5 2344 1146  640   618     0
```

```
%r #Penalty Error Rate based on baseline data
sum(penaltyErrors) / nrow(ClaimsTest)
```

```
[1] 0.7386055
```

Our Goals:

Now our goal is to create a CART model that will give better accuracy higher than 68.38% and a penalty error lower than 0.739

```
%r
# Load necessary libraries
library(rpart) #For building cart model
library(rpart.plot) #For plotting cart model
```

Complexity Parameter using Cross Validation

The complexity parameter (cp) is used to control the size of the decision tree and to select the optimal tree size.

```
%r
library(e1071)
library(caret)
```

```
%r
numFolds = trainControl( method = "cv", number = 10 )
```

```
cpGrid = expand.grid( .cp = seq(0.0,0.03,0.001))
```

11. Perform the cross validation

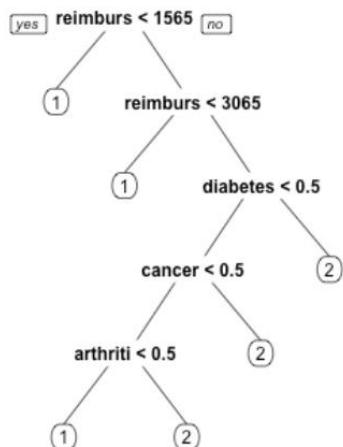
```
cv = train(bucket2009 ~ age + alzheimers + arthritis + cancer + copd +  
depression + diabetes + heart.failure + ihd + kidney + osteoporosis +  
stroke + bucket2008 + reimbursement2008, data = ClaimsTrain, method =  
"rpart", trControl = numFolds, tuneGrid = cpGrid )
```

```
%r  
capture.output(cv)
```

```
[1] "CART "<br />  
[2] ""<br />  
[3] "274803 samples"<br />  
[4] " 15 predictor"<br />  
[5] ""<br />  
[6] "No pre-processing"<br />  
[7] "Resampling: Cross-Validated (10 fold) "<br />  
[8] "Summary of sample sizes: 247322, 247322, 247323, 247322, 247322, 247323, ... "  
[9] "Resampling results across tuning parameters:"<br />  
[10] ""<br />  
[11] " cp RMSE Rsquared "<br />  
[12] " 0.000 0.7938988 0.2168627"<br />  
[13] " 0.001 0.7276304 0.3012112"<br />  
[14] " 0.002 0.7295325 0.2975572"<br />  
[15] " 0.003 0.7347164 0.2875466"<br />  
[16] " 0.004 0.7364503 0.2841780"<br />  
[17] " 0.005 0.7388593 0.2794822"<br />
```

```
%r  
ClaimsTree = rpart(bucket2009 ~ age + alzheimers + arthritis + cancer +  
copd + depression + diabetes + heart.failure + ihd + kidney + osteoporosis  
+ stroke + bucket2008 + reimbursement2008, data=ClaimsTrain,  
method="class", cp=0.001)
```

```
%r {"imageWidth": "700px"}  
prp(ClaimsTree)  
# The size of the tree depends on the volume of training data and number of  
classifications
```



12. Make predictions

```
%r
PredictTest = predict(ClaimsTree, newdata = ClaimsTest, type =
"class")
```

13. Confusion Matrix

```
%r
confusionMatrixTest = base::table(ClaimsTest$bucket2009,
PredictTest)
```

14. Prediction Accuracy

```
%r
sum(diag(confusionMatrixTest)) / nrow(ClaimsTest)
#Remember baseline accuracy was 0.6838135
```

[1] 0.7105545

The accuracy of CART model is 71.05%

15. Penalty Error

```
%r
as.matrix(base::table(ClaimsTest$bucket2009, PredictTest)) *
PenaltyMatrix
```

```

PredictTest
    1   2   3   4   5
1   0 9274 0 0 0
2 36738 0 0 0 0
3 31648 16956 0 0 0
4 18456 19444 0 0 0
5 2752 4278 0 0 0

```

16. Penalty Error Rate on Test Data based on the prediction result

```

%r
sum(as.matrix(base::table(ClaimsTest$bucket2009, PredictTest)) *
PenaltyMatrix) / nrow(ClaimsTest)

# Remember penalty error rate based on baseline data was 0.7386055

```

[1] 0.7617057

The penalty error of the baseline method is 0.758

While we increased the accuracy, the penalty error also went up. Why?

By default, rpart will try to maximize the overall accuracy, and every type of error is seen as having a penalty of one.

Our CART model predicts 3, 4, and 5 so rarely because there are very few observations in these classes. Therefore, we do not really expect this model to do better on the penalty error than the baseline method.

To fix this, rpart function allows us to specify a parameter called loss.

17. New CART model with loss matrix

```

%r
ClaimsTree = rpart(bucket2009 ~ age + alzheimers + arthritis +
cancer + copd + depression + diabetes + heart.failure + ihd + kidney +
osteoporosis + stroke + bucket2008 + reimbursement2008,
data=ClaimsTrain, method="class", cp=0.00005, parms = list(loss =
PenaltyMatrix))

```

18. Redo predictions and penalty error

```

%r
PredictTest = predict(ClaimsTree, newdata = ClaimsTest, type =
"class")

base::table(ClaimsTest$bucket2009, PredictTest)

```

```

PredictTest
 1   2   3   4   5
1 94310 25295 3087 286 0
2 7176 18942 8079 643 0
3 3590 7706 4692 401 1
4 1304 3193 2803 636 1
5 135 356 408 156 2

```

19. Baseline Accuracy for each bucket

```

%r
base:::table(ClaimsTest$bucket2009) / nrow(ClaimsTest)

```

0.671269964 0.190172596 0.089464089 0.043323763 0.005769588

20. Prediction Accuracy for each bucket

```

%r
buckets = base:::table(ClaimsTest$bucket2009)
accuracy_buckets = diag(base:::table(ClaimsTest$bucket2009,
PredictTest)) / buckets
accuracy_buckets

```

0.766885134 0.543685419 0.286272117 0.080131032 0.001892148

21. Penalty Error Rate

```

%r
sum(as.matrix(base:::table(ClaimsTest$bucket2009, PredictTest)) *
PenaltyMatrix) / nrow(ClaimsTest)

#Remember baseline accuracy was 0.6838135 and accuracy based on
previous model was 0.7578902

```

[1] 0.6418161

The accuracy of this CART model is the 64.73%.

22. Penalty Error Rate based on model with penalty condition

```
%r  
  
sum(as.matrix(base::table(ClaimsTest$bucket2009, PredictTest)) *  
PenaltyMatrix) / nrow(ClaimsTest)  
  
# Remember penalty error rate based on baseline data was 0.7386055  
  
# And model without penalty condition was 0.7578902
```

[1] 0.6418161

The penalty error of the CART model with loss is 0.642.

Our accuracy is now lower than the baseline method, but our penalty error is also much lower.

Observations:

Model accuracy of each bucket improved substantially

Penalty Error Rate has gone down

Sacrificed model accuracy to reduce penalty error rate

Clustering

It is an unsupervised algorithm help segment data into groups

Two popular clustering algorithms –

1. Hierarchical and
 2. k-means (works best with large dataset)

Similarity is calculated as distance between two points based on feature values

```
%r #Euclidean distance between 2 movies
```

```
God_Father = c(0,1,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0)
```

```

Titanic = c(0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0)

#Distance between these two data points in feature space
sqrt(sum((God_Father - Titanic) ^ 2))

#God_Father - Titanic

```

[1] 1.414214

Hierarchical Clustering

- Start each point with its own cluster.
- Find two nearest clusters based on the Euclidean distance between two centroids and merge into a single cluster.
- Eventually all points will belong to a single cluster.
- The clustering data is visualized using a dendrogram.
- Height presents the distance before the 2 clusters were combined.
- Dendrogram can help us decide how many clustering do we want.

Example1:

1. Load the movie_lens.txt file and list the variables.

```

%r
movies = read.table("/data/movie_lens.txt", header=FALSE,
sep="|", quote="\"")
str(movies)

```

```

'data.frame': 1682 obs. of 24 variables:
 $ V1 : int 1 2 3 4 5 6 7 8 9 10 ...
 $ V2 : Factor w/ 1664 levels "'Til There Was You (1997)",...: 1525 618 555 594 344 1318 1545 111 391 1240 ...
 $ V3 : Factor w/ 241 levels "", "01-Aug-1997",...: 71 71 71 71 71 71 71 71 71 182 ...
 $ V4 : logi NA NA NA NA NA NA ...
 $ V5 : Factor w/ 1661 levels "", "<a href="http://us.imdb.com/M/title-exact?%22Langoliers,%20The%22%20(1995)%20(mini)%22,...:>http://us.
imdb.com/M/title-exact?%22Langoliers,%20The%22%20(1995)%20(mini)"...:</a> 1430 564 504 542 309 1661 1452 102 356 1182 ...
 $ V6 : int 0 0 0 0 0 0 0 0 0 ...
 $ V7 : int 0 1 0 1 0 0 0 0 0 ...
 $ V8 : int 0 1 0 0 0 0 0 0 0 ...
 $ V9 : int 1 0 0 0 0 0 0 0 0 ...
 $ V10: int 1 0 0 0 0 0 0 1 0 0 ...
 $ V11: int 1 0 0 1 0 0 0 1 0 0 ...
 $ V12: int 0 0 0 0 1 0 0 0 0 0 ...
 $ V13: int 0 0 0 0 0 0 0 0 0 0 ...
 $ V14: int 0 0 0 1 1 1 1 1 1 1 ...
 $ V15: int 0 0 0 0 0 0 0 0 0 0 ...

```

2. Add names to the column

```
%r  
colnames(movies) = c("ID", "Title", "ReleaseDate",  
"VideoReleaseDate", "IMDB", "Unknown", "Action", "Adventure",  
"Animation", "Childrens", "Comedy", "Crime", "Documentary", "Drama",  
"Fantasy", "FilmNoir", "Horror", "Musical", "Mystery", "Romance",  
"SciFi", "Thriller", "War", "Western")
```

3. List the variables again

```
%r str(movies)
```

```
'data.frame': 1682 obs. of 24 variables:  
 $ ID      : int 1 2 3 4 5 6 7 8 9 10 ...  
 $ Title   : Factor w/ 1664 levels "'Til There Was You (1997)",...: 1525 618 555 594 344 1318 1545 111 391 1240 ...  
 $ ReleaseDate : Factor w/ 241 levels "", "01-Aug-1997",...: 71 71 71 71 71 71 71 71 182 ...  
 $ VideoReleaseDate: logi NA NA NA NA NA ...  
 $ IMDB    : Factor w/ 1661 levels "", "<a href="http://us.imdb.com/M/title-exact?%22Langoliers,%20The%22%20(1995)%20(mini)%22...  
 :>"http://us.imdb.com/M/title-exact?%22Langoliers,%20The%22%20(1995)%20(mini)",...:</a> 1430 564 504 542 309 1661 1452 102 356 1182 ...  
 $ Unknown  : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ Action   : int 0 1 0 1 0 0 0 0 0 0 ...  
 $ Adventure: int 0 1 0 0 0 0 0 0 0 0 ...  
 $ Animation: int 1 0 0 0 0 0 0 0 0 0 ...  
 $ Childrens: int 1 0 0 0 0 0 0 1 0 0 ...  
 $ Comedy   : int 1 0 0 1 0 0 0 1 0 0 ...  
 $ Crime   : int 0 0 0 0 1 0 0 0 0 0 ...  
 $ Documentary: int 0 0 0 0 0 0 0 0 0 0 ...  
 $ Drama   : int 0 0 0 1 1 1 1 1 1 1 ...  
 $ Fantasy : int 0 0 0 0 0 0 0 0 0 0 ...  
 $ ...       : int 0 0 0 0 0 0 0 0 0 0 ...
```

4. Remove unnecessary variables

```
%r  
movies$ID = NULL  
movies$ReleaseDate = NULL  
movies$VideoReleaseDate = NULL  
movies$IMDB = NULL  
  
str(movies)
```

```
'data.frame': 1682 obs. of 20 variables:  
$ Title : Factor w/ 1664 levels "'Til There Was You (1997)",...: 1525 618 555 594 344 1318 1545 111 391 1240 ...  
$ Unknown : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Action : int 0 1 0 1 0 0 0 0 0 0 ...  
$ Adventure : int 0 1 0 0 0 0 0 0 0 0 ...  
$ Animation : int 1 0 0 0 0 0 0 0 0 0 ...  
$ Childrens : int 1 0 0 0 0 0 0 1 0 0 ...  
$ Comedy : int 1 0 0 1 0 0 0 1 0 0 ...  
$ Crime : int 0 0 0 0 1 0 0 0 0 0 ...  
$ Documentary: int 0 0 0 0 0 0 0 0 0 0 ...  
$ Drama : int 0 0 0 1 1 1 1 1 1 1 ...  
$ Fantasy : int 0 0 0 0 0 0 0 0 0 0 ...  
$ FilmNoir : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Horror : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Musical : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Mystery : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Romance : int 0 0 0 0 0 0 0 0 0 0 ...  
$ SciFi : int 0 0 0 0 0 0 0 0 0 0 ...
```

5. Remove duplicate entries from the dataset

```
%r # Remove duplicates  
  
movies = unique(movies)  
  
str(movies)
```

```
'data.frame': 1664 obs. of 20 variables:  
$ Title : Factor w/ 1664 levels "'Til There Was You (1997)",...: 1525 618 555 594 344 1318 1545 111 391 1240 ...  
$ Unknown : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Action : int 0 1 0 1 0 0 0 0 0 0 ...  
$ Adventure : int 0 1 0 0 0 0 0 0 0 0 ...  
$ Animation : int 1 0 0 0 0 0 0 0 0 0 ...  
$ Childrens : int 1 0 0 0 0 0 0 1 0 0 ...  
$ Comedy : int 1 0 0 1 0 0 0 1 0 0 ...  
$ Crime : int 0 0 0 0 1 0 0 0 0 0 ...  
$ Documentary: int 0 0 0 0 0 0 0 0 0 0 ...  
$ Drama : int 0 0 0 1 1 1 1 1 1 1 ...  
$ Fantasy : int 0 0 0 0 0 0 0 0 0 0 ...  
$ FilmNoir : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Horror : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Musical : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Mystery : int 0 0 0 0 0 0 0 0 0 0 ...  
$ Romance : int 0 0 0 0 0 0 0 0 0 0 ...  
$ SciFi : int 0 0 0 0 0 0 0 0 0 0 ...
```

6. Compute the euclidean distance

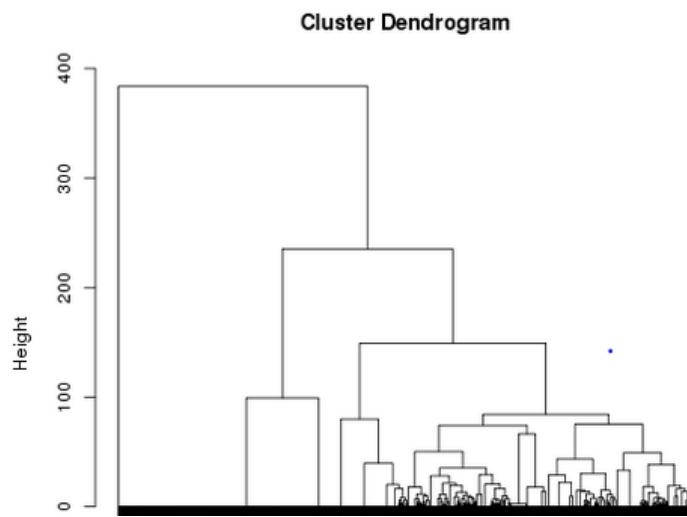
```
%r  
  
distances = dist(movies[2:20], method = "euclidean")  
  
#Clustering is based on column 2 through 20.
```

7. Calculates distance using centroid and variance within a cluster.

```
%r  
  
# Hierarchical clustering  
  
clusterMovies = hclust(distances, method = "ward.D")  
  
#ward.D
```

8. Plot the Dendrogram

```
%r  
plot(clusterMovies)
```



9. Assign points to cluster

```
%r  
clusterGroups = cutree(clusterMovies, k = 10)
```

```
%r clusterGroups
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	1	3	4	1	1	4	1	3	3	5	6	4
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
7	3	4	4	6	1	1	3	3	5	5	2	2	3	4
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
1	8	6	9	1	6	4	2	3	5	5	5	3	3	9
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
4	9	8	7	1	1	4	2	2	6	3	4	4	4	4
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
4	2	1	4	9	7	5	6	1	7	1	1	1	1	8
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
3	3	1	2	1	7	2	7	2	5	4	4	7	3	6
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
1	6	9	1	1	2	1	3	1	3	2	1	1	5	5
106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
3	4	5	2	1	7	1	4	1	8	5	2	2	8	1
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135

10. Let's use the tapply function to compute the percentage of movies in each genre and cluster

```
%r  
# Now let's figure out what the clusters are like.  
tapply(movies$Action, clusterGroups, mean) #This will calculate mean  
of Action variable for each of 10 clusters
```

```
0.1784512 0.7839196 0.1238532 0.0000000 0.0000000 0.1015625 0.0000000  
8 9 10  
0.0000000 0.0000000 0.0000000
```

```
%r tapply(movies$Romance, clusterGroups, mean)
```

```
0.10437710 0.04522613 0.03669725 0.00000000 0.00000000 1.00000000  
7 8 9 10  
1.0000000 0.0000000 0.0000000 0.0000000
```

```
%r  
clusters = data.frame()  
for(i in 1:10){  
  clusters = rbind(clusters, colMeans(subset(movies[2:20],  
clusterGroups == i)))  
}  
clusters = t(clusters)  
rownames(clusters) = colnames(movies[2:20])  
clusters
```

```
Unknown 0.006734007 0.000000000 0.000000000 0 0 0.0000000 0 0.0000000  
Action 0.178451178 0.783919598 0.123853211 0 0 0.1015625 0 0.0000000  
Adventure 0.185185185 0.351758794 0.036697248 0 0 0.0000000 0 0.0000000  
Animation 0.134680135 0.010050251 0.000000000 0 0 0.0000000 0 0.0000000  
Childrens 0.393939394 0.005025126 0.009174312 0 0 0.0000000 0 0.0000000  
Comedy 0.363636364 0.065326633 0.064220183 0 1 0.1093750 1 0.0212766  
Crime 0.033670034 0.005025126 0.412844037 0 0 0.0468750 0 0.0000000  
Documentary 0.010101010 0.000000000 0.000000000 0 0 0.0000000 0 1.0000000  
Drama 0.306397306 0.110552764 0.380733945 1 0 0.6640625 0 0.0000000  
Fantasy 0.070707071 0.000000000 0.004587156 0 0 0.0000000 0 0.0000000  
FilmNoir 0.000000000 0.000000000 0.105504587 0 0 0.0078125 0 0.0000000  
Horror 0.016835017 0.080402010 0.018348624 0 0 0.0156250 0 0.0000000  
Musical 0.188552189 0.000000000 0.000000000 0 0 0.0000000 0 0.0000000  
Mystery 0.000000000 0.000000000 0.275229358 0 0 0.0000000 0 0.0000000  
Romance 0.104377104 0.045226131 0.036697248 0 0 1.0000000 1 0.0000000  
SciFi 0.074074074 0.346733668 0.041284404 0 0 0.0000000 0 0.0000000  
Thriller 0.040404040 0.376884422 0.610091743 0 0 0.1406250 0 0.0000000
```

11. Find which cluster 'Men in Black' is in.

```
%r  
subset(movies, Title=="Men in Black (1997)")
```

```
257 Men in Black (1997)      0      1      1      0      0  
Comedy Crime Documentary Drama Fantasy FilmNoir Horror Musical Mystery  
257      1      0      0      0      0      0      0      0      0  
Romance SciFi Thriller War Western  
257      0      1      0      0      0
```

```
%r  
clusterGroups[which(movies>Title=="Men in Black (1997)")]
```

```
257  
2
```

12. Create a new data set with just the movies from cluster 2

```
%r  
cluster2 = subset(movies, clusterGroups==2)
```

13. Look at the first 10 titles in this cluster

```
%r  
cluster2$title[1:10]
```

```
[1] GoldenEye (1995)<br />  
[2] Bad Boys (1995)<br />  
[3] Apollo 13 (1995)<br />  
[4] Net, The (1995)<br />  
[5] Natural Born Killers (1994)<br />  
[6] Outbreak (1995)<br />  
[7] Stargate (1994)<br />  
[8] Fugitive, The (1993)<br />  
[9] Jurassic Park (1993)<br />  
[10] Robert A. Heinlein's The Puppet Masters (1994)  
1664 Levels: 'Til There Was You (1997) ...
```

Example2:

1. Load the flower.csv file and list the variables

```
%r  
flower = read.csv("/data/flower.csv", header=FALSE)  
str(flower)
```

```
'data.frame': 50 obs. of 50 variables:  
 $ V1 : num 0.0991 0.0991 0.1034 0.1034 0.1034 ...  
 $ V2 : num 0.112 0.108 0.112 0.116 0.108 ...  
 $ V3 : num 0.134 0.116 0.121 0.116 0.112 ...  
 $ V4 : num 0.138 0.138 0.121 0.121 0.112 ...  
 $ V5 : num 0.138 0.134 0.125 0.116 0.112 ...  
 $ V6 : num 0.138 0.129 0.121 0.108 0.112 ...  
 $ V7 : num 0.129 0.116 0.103 0.108 0.112 ...  
 $ V8 : num 0.116 0.103 0.103 0.103 0.116 ...  
 $ V9 : num 0.1121 0.0991 0.1078 0.1121 0.1164 ...  
 $ V10: num 0.121 0.108 0.112 0.116 0.125 ...  
 $ V11: num 0.134 0.125 0.129 0.134 0.129 ...  
 $ V12: num 0.147 0.134 0.138 0.129 0.138 ...  
 $ V13: num 0.000862 0.146552 0.142241 0.142241 0.133621 ...  
 $ V14: num 0.000862 0.000862 0.142241 0.133621 0.12931 ...  
 $ V15: num 0.142 0.142 0.134 0.121 0.116 ...  
 $ V16: num 0.125 0.125 0.116 0.108 0.108 ...
```

2. Change the data type to matrix

```
%r  
  
flowerMatrix = as.matrix(flower)  
  
str(flowerMatrix)
```

```
num [1:50, 1:50] 0.0991 0.0991 0.1034 0.1034 0.1034 ...
```

3. Turn matrix into a vector

```
%r  
  
flowerVector = as.vector(flowerMatrix)  
  
str(flowerVector)
```

```
num [1:2500] 0.0991 0.0991 0.1034 0.1034 0.1034 ...
```

4. Compute the euclidean distance

```
%r  
  
distance = dist(flowerVector, method = "euclidean")
```

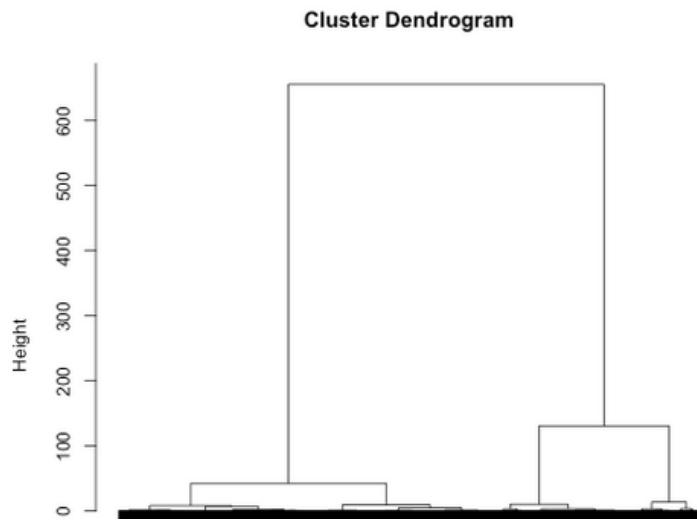
5. Prepare the data for clustering

```
%r  
  
# Hierarchical clustering  
  
clusterIntensity = hclust(distance, method="ward.D")
```

6. Plot the dendrogram

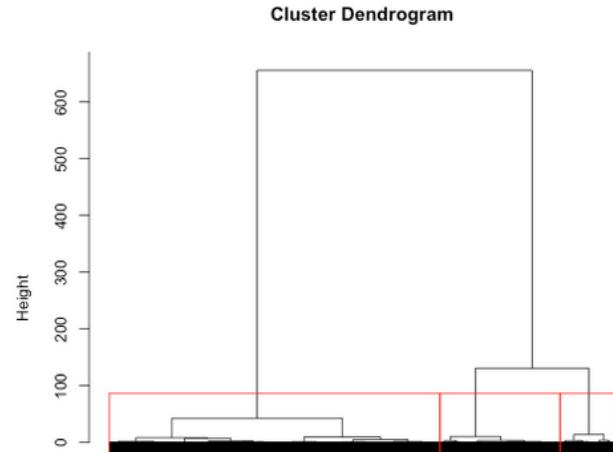
```
%r
```

```
plot(clusterIntensity)
```



7. Select 3 cluster and plot the dendrogram

```
%r  
plot(clusterIntensity)  
rect.hclust(clusterIntensity, k = 3, border = "red")
```



```
%r  
flowerClusters = cutree(clusterIntensity, k = 3)  
flowerClusters
```

8. Find the mean intensity values using tapply function

```
%r  
  
tapply(flowerVector, flowerClusters, mean)
```

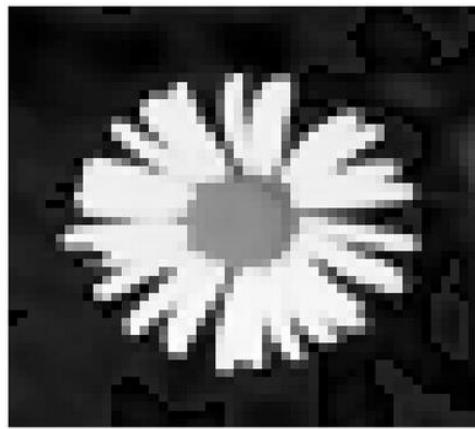
0.08574315 0.50826255 0.93147713

9. Plot the image and the clusters

```
%r  
dim(flowerClusters) = c(50,50)  
image(flowerClusters, axes = FALSE)
```



```
%r  
# Original image  
image(flowerMatrix, axes=FALSE, col=grey(seq(0,1,length=256)))
```



k-means Clustering

Algorithm in nutshell

Step 1: Specify a desired number of clusters k based on previous knowledge or experimenting

Step 2: Randomly assign each data to a cluster

Step 3: Compute cluster centroids

Step 4: Reassign each point to the closest cluster centroid

Step 5: Repeat 4 and 5 until no improvement is made

MRI Image Clustering using K-means

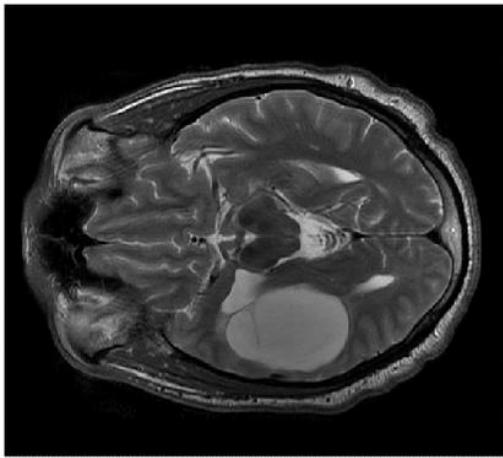
1. Load the tumor.csv file and list the variables

```
%r  
healthy = read.csv("/data/tumor.csv", header=FALSE)  
healthyMatrix = as.matrix(healthy)  
str(healthyMatrix)
```

num [1:571, 1:512] 0 0 0 0 0 0 0 0 0 ...

2. Plot the image

```
%r  
image(healthyMatrix, axes=FALSE, col=grey(seq(0,1,length=256)))
```



3. Run k-means algorithm with number of clusters as 5

```
%r  
  
# Run k-means  
  
set.seed(1)  
  
healthyVector = as.vector(healthyMatrix)  
  
KMC = kmeans(healthyVector, centers = k, iter.max = 1000)  
  
str(KMC)
```

```
List of 9  
$ cluster : int [1:292352] 5 5 5 5 5 5 5 5 5 ...  
$ centers : num [1:5, 1] 0.43396 0.61961 0.28868 0.18492 0.00708  
..- attr(*, "dimnames")=List of 2  
.. ..$ : chr [1:5] "1" "2" "3" "4" ...  
.. ..$ : NULL  
$ totss : num 8600  
$ withinss : num [1:5] 54.3 50.6 69 76.8 43.8  
$ tot.withinss: num 294  
$ betweenss : num 8305  
$ size : int [1:5] 29811 8366 60806 53943 139426  
$ iter : int 5  
$ ifault : int 0
```

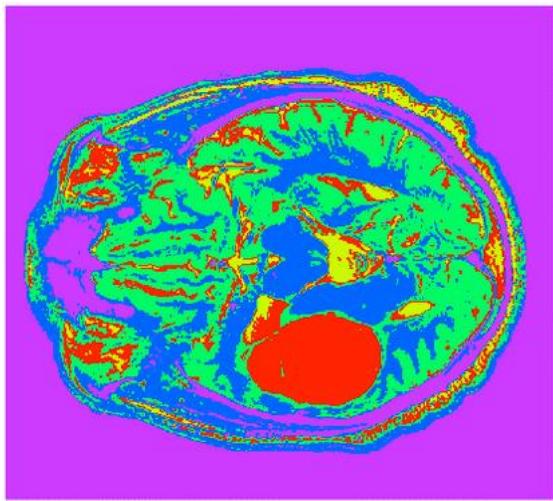
4. Extract clusters

```
%r  
  
healthyClusters = KMC$cluster  
KMC$centers[2]
```

```
[1] 0.6196116
```

5. Plot the image of clusters

```
%r  
dim(healthyClusters) = c(nrow(healthyMatrix), ncol(healthyMatrix))  
image(healthyClusters, axes = FALSE, col=rainbow(k))
```



PCA (Principle Component Analysis)

```
%r  
library(ISLR)  
nci.labs=NCI60$labs  
nci.data=NCI60$data  
dim(nci.data)
```

[1] 64 6830

```
%r pr.out=prcomp(nci.data, scale=TRUE) #Performs a principal components analysis on the given data matrix and returns the results as an object of class prcomp.
```

```
pr.out
```

```

Standard deviations:
[1] 2.785347e+01 2.148136e+01 1.982046e+01 1.703256e+01 1.597181e+01
[6] 1.572108e+01 1.447145e+01 1.354427e+01 1.314400e+01 1.273860e+01
[11] 1.268672e+01 1.215769e+01 1.183019e+01 1.162554e+01 1.143779e+01
[16] 1.100051e+01 1.065666e+01 1.048880e+01 1.043518e+01 1.032194e+01
[21] 1.014608e+01 1.005439e+01 9.902655e+00 9.647656e+00 9.507638e+00
[26] 9.332529e+00 9.273200e+00 9.090046e+00 8.981173e+00 8.750031e+00
[31] 8.599622e+00 8.447375e+00 8.373048e+00 8.215787e+00 8.157313e+00
[36] 7.974655e+00 7.904462e+00 7.821271e+00 7.721562e+00 7.586035e+00
[41] 7.456193e+00 7.344380e+00 7.104489e+00 7.013055e+00 6.958385e+00
[46] 6.866265e+00 6.807439e+00 6.647630e+00 6.616068e+00 6.407926e+00
[51] 6.219838e+00 6.203258e+00 6.067065e+00 5.918049e+00 5.912333e+00
[56] 5.735386e+00 5.472610e+00 5.292148e+00 5.021174e+00 4.683979e+00
[61] 4.175673e+00 4.082121e+00 4.041243e+00 2.148247e-14

```

```
%r str(pr.out$x)
```

```
num [1:64, 1:64] -19.7 -22.9 -27.2 -42.5 -55 ...
```

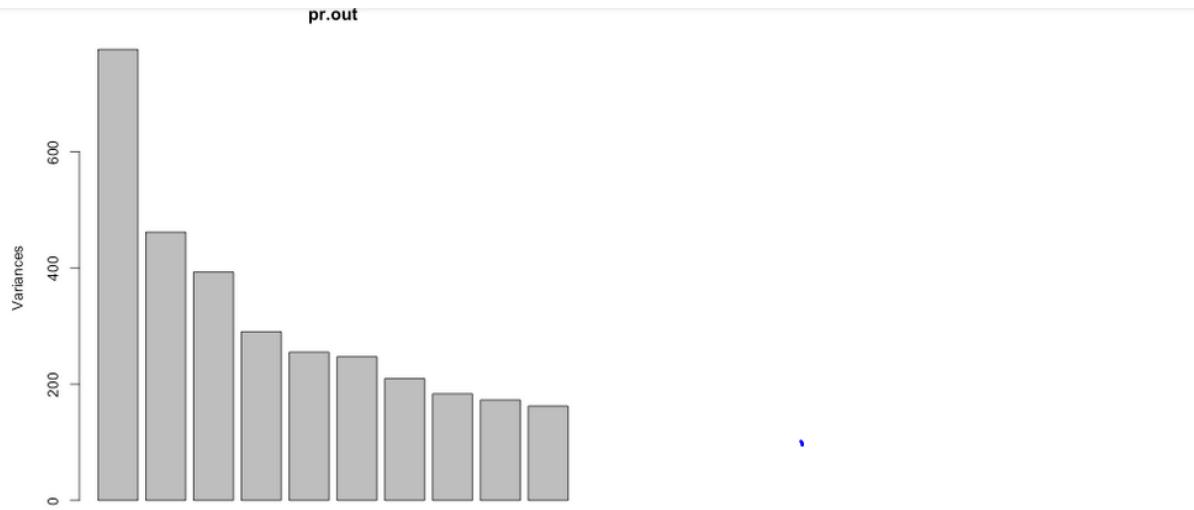
```
%r capture.output(summary(pr.out))
```

```

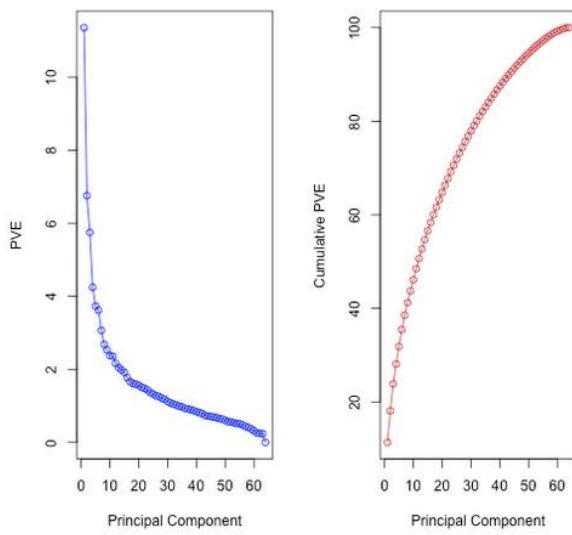
[1] "Importance of components:<br />
[2] " PC1      PC2      PC3      PC4      PC5"<br />
[3] "Standard deviation 27.8535 21.48136 19.82046 17.03256 15.97181"<br />
[4] "Proportion of Variance 0.1136 0.06756 0.05752 0.04248 0.03735"<br />
[5] "Cumulative Proportion 0.1136 0.18115 0.23867 0.28115 0.31850"<br />
[6] " PC6      PC7      PC8      PC9      PC10"<br />
[7] "Standard deviation 15.72108 14.47145 13.54427 13.14400 12.73860"<br />
[8] "Proportion of Variance 0.03619 0.03066 0.02686 0.02529 0.02376"<br />
[9] "Cumulative Proportion 0.35468 0.38534 0.41220 0.43750 0.46126"<br />
[10] " PC11     PC12     PC13     PC14     PC15"<br />
[11] "Standard deviation 12.68672 12.15769 11.83019 11.62554 11.43779"<br />
[12] "Proportion of Variance 0.02357 0.02164 0.02049 0.01979 0.01915"<br />
[13] "Cumulative Proportion 0.48482 0.50646 0.52695 0.54674 0.56590"<br />
[14] " PC16     PC17     PC18     PC19     PC20"<br />
[15] "Standard deviation 11.00051 10.65666 10.48880 10.43518 10.3219"<br />
[16] "Proportion of Variance 0.01772 0.01663 0.01611 0.01594 0.0156"<br />
[17] "Cumulative Proportion 0.58361 0.60024 0.61635 0.63229 0.6479"<br />
... "

```

```
%r plot(pr.out)
```



```
%r
pve=100*pr.out$sdev^2/sum(pr.out$sdev^2)
par(mfrow=c(1,2))
plot(pve, type="o", ylab="PVE", xlab="Principal Component",
col="blue")
plot(cumsum(pve), type="o", ylab="Cumulative PVE", xlab="Principal
Component", col="brown3")
```



```
%r base::table(nci.labs)
```

```

nci.labs
BREAST      CNS      COLON    K562A-repro  K562B-repro  LEUKEMIA
7           5         7        1           1           6
MCF7A-repro MCF7D-repro MELANOMA   NSCLC       OVARIAN     PROSTATE
1           1         8        9          6          2
RENAL      UNKNOWN
9           1

```

```

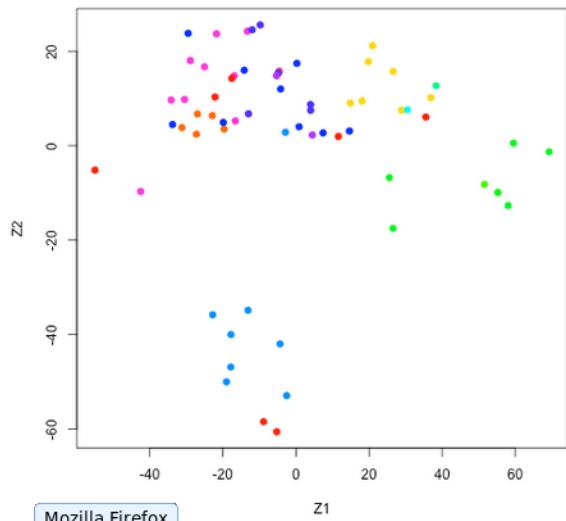
%r
Cols=function(vec) {
  cols=rainbow(length(unique(vec))) #rainbow: Create a vector of n
contiguous colors.
  return(cols[as.numeric(as.factor(vec))])
}
par(mfrow=c(1,2))

```

```

%r plot(pr.out$x[,1:2], col=Cols(nci.labs),
pch=19,xlab="Z1",ylab="Z2")

```



```

%r plot(pr.out$x[,c(1,3)], col=Cols(nci.labs),
pch=19,xlab="Z1",ylab="Z3")

```

