# Task

The goal of this project is to infer qualitative data regarding the use of the CitiBike Service in New York City.

Each row in the data set contains the following data (separated by a ",", and sometimes each value was among two ""):

- tripduration
- starttime
- stoptime
- start station id
- end station id
- start station name
- start station latitude
- start station longitude
- end station id
- end station name
- end station latitude
- end station longitude
- bikeid
- usertype (customer = 24-hour or 7-day pass; subscriber = annual member)
- birth year
- sex of the biker

Using Hadoop in a fully distributed cluster (use your own physical/virtual machines) provide the following information:

- average duration of trips per week in 2015
- number of customers (NOT subscribers) using the bikes per week in 2015
- number of trips and average duration of trips per biker age range (16-19, 20-29, 30-39, 40-49, 50-59, 60-69)
- for each day between the 1st of June and the 31st of August provide the id and name of the station that saw the most amount of traffic (number of incoming bikes + number of outgoing bikes)

The data sets were referred to January-November 2015 (so we have 11 files) in .Csv format.

# Hadoop & Cluster Configuration

In order to develop this project, we have first set up Hadoop on a single machine and, after having written the code, we have tested it on that single node. Afterwards we have launch the program on a fully distributed cluster (4 machine, 1 namenode and 3 datanodes).
For the configuration of the fully distributed cluster, we have used 3 machines with Ubuntu 14.04, from Amazon Web Services (with features that allows us to use them freely), with basic free features and, in particular:
- 8 GB of storage
- All ports open

For a complete explanation of how to set up these machines, we have followed this guide:
http://insightdataengineering.com/blog/hadoopdevops/

(For an explanation of how to setup hadoop on a single node:
http://zhongyaonan.com/hadoop-tutorial/setting-up-hadoop-2-6-on-mac-osx-yosemite.html )

In order to setup a ssh connection to these machines you need to generate (on the Amazon AWS console) a new key pair, download the private key file (.pem), add it to your ~/.ssh/ directory. Then, retrieve from Amazon Aws the public dns and the Ip of the machines, and it to ~/.ssh/config file as following:

```
Host namenode
  HostName namenode_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename

Host datanode1
  HostName datanode1_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename

Host datanode2
  HostName datanode2_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename

Host datanode3
  HostName datanode3_public_dns
  User ubuntu
  IdentityFile ~/.ssh/pem_key_filename
```

At this time, it is also possible to set a passwordless connection to the four machines.
Now we need to install Hadoop and Java on the machine.
Afterwards we define some variables to add to ~/.profile file.

```
export JAVA_HOME=/usr
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

Now we proceed to configure hadoop, modifying the following files in the following way in all nodes:

$HADOOP_CONF_DIR/hadoop-env.sh:

```
# The java implementation to use.
export JAVA_HOME=/usr
```

$HADOOP_CONF_DIR/core-site.xml:

```
<configuration>
 <property>
  <name>fs.defaultFS</name>
  <value>hdfs://namenode_public_dns:9000</value>
 </property>
</configuration>
```

$HADOOP_CONF_DIR/yarn-site.xml:

```
<configuration>

<!-- Site specific YARN configuration properties -->

 <property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
 </property>
 <property>
  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
 </property>
 <property>
  <name>yarn.resourcemanager.hostname</name>
  <value>namenode_public_dns</value>
 </property>
</configuration>
```

$HADOOP_CONF_DIR/mapred-site.xml:

```
<configuration>
 <property>
  <name>mapreduce.jobtracker.address</name>
  <value>namenode_public_dns:54311</value>
 </property>
 <property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
 </property>
</configuration>
```

And, only on the namenode

/etc/hosts:

```
127.0.0.1 localhost
namenode_public_dns namenode_hostname
datanode1_public_dns datanode1_hostname
datanode2_public_dns datanode2_hostname
datanode3_public_dns datanode3_hostname

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

$HADOOP_CONF_DIR/hdfs-site.xml:

```
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>3</value>
 </property>
 <property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///usr/local/hadoop/hadoop_data/hdfs/namenode</value>
 </property>
</configuration>
```

$HADOOP_CONF_DIR/masters:

```
namenode_hostname
```

$HADOOP_CONF_DIR/slaves

```
datanode1_hostname
datanode2_hostname
datanode3_hostname
```

And, only on datanodes:

$HADOOP_CONF_DIR/hdfs-site.xml:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///usr/local/hadoop/hadoop_data/hdfs/datanode</value>
  </property>
</configuration>
```

# Starting Hadoop & run the programme

Now we can start the hadoop cluster on the namenode:

```
hdfs namenode -format
```
(format the distributed file system)

```
$HADOOP_HOME/sbin/start-dfs.sh
```

```
$HADOOP_HOME/sbin/start-yarn.sh
```

Now, if we type “jps” on terminal we would see:

```
namenode$ jps
21817 JobHistoryServer
21853 Jps
21376 SecondaryNameNode
21540 ResourceManager
21157 NameNode

datanodes$ jps
20936 NodeManager
20792 DataNode
21036 Jps
```

Now we can create a directory where we can put our input files (after having copied it from our local machines to namenode) for our program.

```
hdfs dfs –mkdir /user
hdfs dfs –mkdir /user/inputs
hdfs dfs –put your_directory_on_namenode_where_data_are_saved /user/inputs
```

Now we can launch the program with the command:

hadoop jar jar_directory/CityBikeComp.jar input_files_directory output_directory_on_hdfs_1 output_directory_on_hdfs_2 output_directory_on_hdfs_3 hadoop_folder_directory

We will see in terminal the logger, and we can see the progress of job in browser at https://address_of_machine:8088, and we can see the distributed file system information and directory at https://address_of_machine:50070 .

Once job are finished, we can browse the hdfs to see the output files.

If we want to re-run the jar, we can stop the hdfs, delete the temporary files of previous jobs and then repeat the same procedure, starting with formatting the hdfs.

For example:
$HADOOP_HOME/sbin/stop-dfs.sh
$HADOOP_HOME/sbin/stop-yarn.sh
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs/datanode/current
sudo rm -rf /usr/local/hadoop/hadoop_data/hdfs/namenode/current

Now we can download the file on our local machine and we can run the jar program called "CityBikeCharts.jar" seeing the graph produced referring to the output files.
The arguments needed for running this program are the local directory of the first and the second output files (in this order).

# Main program (CityBikeComp)

Now we will see how we have structured the main program (CityBikeComp.jar)
There are four mapper and four reducer classes. The mapper and relative reducers can be identified because of the number in the suffix (1,2,3,4).
According to the task, we have 4 enquiries:

1. average duration of trips per week in 2015
2. number of customers (NOT subscribers) using the bikes per week in 2015
3. number of trips and average duration of trips per biker age range (16-19, 20-29, 30-39, 40-49, 50-59, 60-69)
4. for each day between the 1st of June and the 31st of August provide the id and name of the station that saw the most amount of traffic (number of incoming bikes + number of outgoing bikes)

We have resolved the first two enquiries with Job1 (Mapper1 – Reducer1), the 3rd with Job2, and the 4th with Job3 and Job4 (thus with a chaining, in which Job4 depends on Job3, and so Job4 will start at the end of Job3).
In the Main Class (containing the run method necessarily for let hadoop to start map reduce jobs) we have set the configuration for the 4 jobs, then created a Job for each configuration, assigned them to a JobControl, and, after having set the dependency of Job4 from Job3, we run the Job and wait in a while-loop until all the Job are finished.

**Job configuration:**

After having assigned to each job configuration the respective mapper and reducer classes, we set the output class of keys and values, and the format of input and output files.
We choose Text for input and output format (it is a csv file), and also Text for all the output values (we have combined more number as values, separating them with a character such as "-", so we cannot choose IntWritable in this case), and for the key output, except for the first, where it is IntWritable, because we choose to refer to the week number.

*Input paths:* the 11 files for the first 3 jobs, whereas the input for the last job is equal to the output path of the 3rd Job (in our case: args[3]/temporanea. The output of the 4rd Job, solving the last request, will be args[3]/finita).
*Outputh paths:* args[1] for the first job, args[2] for the second, args[3]/temporanea for the third.

Every time we need to read from a file, it is necessary to take the key or the value and convert them into String (from Text) or Integer (from IntWritable).
Since we read line per line, we need to perform a lot of *split* to get the value of our interest.

In each Mapper (except the last one, that read from a different input) class we need to read from the .csv file provided, we remove all "" (if there are), and split the values (separeted by ","). We skip the first line (the header), saying to skip it if the first value in the splitted array is "trip duration".

**Solution to question 1 and 2:**
**Mapper 1**: Calculates the **week number** and uses it as the **key**. Then **puts** on the output file the **trip duration** and the number 1 or 0 (if **customer or subscriber**) separated by a "-" sign as a value.

**Reducer 1**: Splits the average and the number earlier assigned (1 or 0), then sums all the duration and the number of customers. Finally saves the **week number** as the **key** and the **average duration** (as sum of all duration divided per total number of user) and the **number of custo**mers (these values are separated by a "-" sign) as a **value**.

**Solution to question 3:**
**Mapper 2**: Using the function to convert the birth year of the customers into the age range puts the **age** range as a **key**, and the **trip duration** as a **value**.

**Reducer 2**: Makes the **average** of **trip duration** for each **age range** and then saves on the **output files** it and the **amount of traffic** (which is basically the count used to compute the average).

**Solution to question 4:**
**Mapper 3**: Puts on the output files some value just if the date of the ride read from the .csv file is between June and August. The **key** on the output files is the **day, the station name and the station id** concatenated. Moreover it performs **two puts** per line one for the start station, the other for end station. The value is just a **flag** necessary for the count. (Uses the word "uno")

**Reducer 3**: Splits the key, and sets as the new **key** for the final output file the **date of the ride**. As a **value** uses the **station id, the station name and the sum of the ride** for that day and for that station. This last value is calculated simply adding 1 to a counter each time there is an equal key (A certain station in a certain data).
The computation of the station with the maximum amount of traffic is left to the 4th job.

**Mapper 4**: Replicates the output file of Reducer 3 in our temporary output file. (The **key** is the **date** and the **value** is **station id, the station name and the sum of the ride**)

**Reducer 4**: Computes the station that has shown the most amount of traffic each day. Splits the values (separated by "-"), and takes the last one, which is the amount of traffic (computed in Job 3), and looks for the maximum. After having found it, it puts on the output the relative station id and station name (one puts per day).

# Graph program (CityBikeCharts)

To develop the code regarding the graph parts, we have used the library JFreeCharts.
The classes “Chart1” and “Chart2” are referred to the first job and show the trend of the average duration and the numbers of trips in 2015.
The classes “Chart3” and “Chart4” are referred to the second job and are bar charts showing the number of trips and the average duration of trips per biker age range.

The job of each class is to read each single line of the output files, taking the part of interest (for example the week and average, without caring about the amount of trips), and add them to a “dataset” created.