



# Report: Spark Practical Work

## Big Data

Academic year: 2022/23

### Group 2

Víctor Morcuende Castell

Guillermo Nájera Lavid

Javier Rocamora García

## TABLE OF CONTENTS

1.	Dataset analysis .....	III
2.	Variables Selected .....	V
3.	Variables Excluded .....	VI
4.	Variable Transformations .....	VII
5.	Variables Created .....	VIII
6.	Machine Learning Techniques .....	IX
7.	Validation Process.....	XI
8.	Model Evaluation .....	XII
9.	Compilation Instructions.....	XIV
10.	Data Placement.....	XV
11.	Conclusions.....	XVIII

# 1. Dataset analysis

As it is said in the project, the dataset we will be working on consists of flight arrival and departure details for all commercial flights within the USA on a set of years. This data in particular comes from The Data Expo 2009, which is a special bi-annual Poster Session that gathers Joint Statistical Meetings and which is sponsored by the Graphics Section and the Computing Section.

The original dataset consists of a large dataset, where there are nearly 120 million records in total, divided into several independent files, and takes up 1.6 gigabytes, but for our case we won't need to use the entire dataset, instead we will be using a small piece of it (starting with 2008.csv) for the purpose of being able to process it in our development environment.

To begin with and having already a quick view of the first dataset we used (2008.csv) after downloading it, we will proceed with a little analysis of the variables to know what the data is telling us and their possible values. In this analysis we will already discard the forbidden variables declared in the project statement (those being "ArrTime", "ActualElapsedTime", "AirTime", "TaxiIn", "Diverted", "CarrierDelay", "WeatherDelay", "NASDelay", "SecurityDelay" and "LateAircraftDelay"). So, in the end we will end up with these variables/columns:

- **Year:** year of the flight (1987-2008)
- **Month:** month of the flight (1-12)
- **DayofMonth:** day of the month (1-31)
- **DayOfWeek:** 1 (Monday) - 7 (Sunday)
- **DepTime:** actual departure time (local, hhmm)
- **CRSDepTime:** scheduled departure time (local, hhmm)
- **CRSArrTime:** scheduled arrival time (local, hhmm)
- **UniqueCarrier:** unique carrier ID code
- **FlightNum:** flight number
- **TailNum:** plane tail number
- **CRSElapsedTime:** actual elapsed time of the flight (in minutes)
- **ArrDelay:** arrival delay (in minutes)
- **DepDelay:** departure delay (in minutes)
- **Origin:** origin IATA airport code
- **Dest:** destination IATA airport code
- **Distance:** flight distance (in miles)
- **TaxiOut:** taxi time from departure from the gate to wheels up (in minutes)
- **Cancelled:** was the flight cancelled? 0 (NO) - 1 (YES)
- **CancellationCode:** reason for cancellation (A = carrier, B = weather, C = NAS, D = security)

The second dataset used (plane-data.csv) will give us the next information:

- **TailNum:** plane tail number (as in the other dataset)
- **Type:** type of the plane (Corporation...)
- **Manufacturer:** manufacturer of the plane
- **Issue\_date:** date when the plane is ready to be used (dd/mm/yyyy)
- **Model:** model of the plane
- **Status:** current state of the plane regarding whether the plane is currently in service, undergoing maintenance or retired from service (Valid...)
- **Aircraft\_type:** type of aircraft of the plane
- **Engine\_type:** type of engine of the plane
- **Year:** year when the plane was constructed

## 2. Variables Selected

Once having a previous clear idea of the dataset we will be working on, now we proceed with the **selection of variables** from the dataset which in the end will be used for this project. This selection will be done **considering the objective of this project, predicting the arrival delay of commercial flights** (therefore, “ArrDelay” is also included).

From the first set of **datasets**, the **variables selected** are:

- **Year**: we thought of excluding this variable, since all the column has the same values, but when we tried it, we saw that the performance of the model decayed
- **Month**: we think that it was useful to have the exact date of the flights (so we could consider possible holidays season)
- **DayofMonth**: same reason as with “Month”
- **DayOfWeek**: same reason as with “Month” and “DayofMonth”
- **DepTime**: it is clear that we need this variable for the model to perform better (we then checked that it has a high correlation with the target variable)
- **CRSArrTime**: same reason as with “DepTime”
- **DepDelay**: same reason as with “DepTime” and “CRSArrTime”
- **Origin**: we believe that it is important to know from where the flight took off
- **Dest**: therefore, the same with the landing place
- **Distance**: we think that it is an important variable since the target variable may vary depending on the distance of the flight
- **TaxiOut**: another important variable to take into account for the prediction of the target variable

From the second dataset (plane-data.csv), we use:

- **Type**
- **Manufacturer**
- **Model**
- **Aircraft\_type**
- **Engine\_type**

Regarding these variables, we believe that they are relevant for our analysis and prediction of the model, since they tell information related to the planes, which play a significant role in the goal of predicting the arrival delay.

### 3. Variables Excluded

Considering that we discarded from the beginning the forbidden variables declared in the project statement, the rest of the variables excluded will be:

From the first set of datasets:

- **CRSDepTime**: we delete this variable since we saw that it was highly correlated with “DepTime”. Moreover, we chose to remove this variable and not “DepTime” since the latter was more correlated to the target variable than the former
- **UniqueCarrier**: we believe that this identifier is irrelevant for our purpose
- **FlightNum**: same reason as with “UniqueCarrier”
- **TailNum**: same reason as with “UniqueCarrier” and “TailNum”
- **CRSElapsedTime**: we delete this variable since we saw that it was highly correlated with “Distance”. Moreover, we chose to remove this variable and not “Distance” since the latter was more correlated to the target variable than the former
- **Cancelled**: it does not make sense to have the flights that are cancelled for the goal of predicting the arrival delay of the flights
- **CancellationCode**: same reason as with “Cancelled”

From the second dataset (plane-data.csv), we deleted:

- **TailNum**: we believe that this identifier is irrelevant for our purpose
- **Status**: extra information that is also irrelevant
- **Year**: we are not going to consider the year of the plane
- **Issue\_date**: we deleted this variable since we transformed it to “PlaneAge” (which will be explained later)

## 4. Variable Transformations

After acquiring the datasets that we are going to use, we performed several transformations.

First, the null values of the target variable, “**ArrDelay**”, since we believe that it **does not make sense to keep this information**, so we deleted every row from the dataset that contained a null value in ArrDelay.

After that, once the set of **datasets** or dataset **and** the **plane-data** dataset **were merged**, we **filtered** the “**Issue\_date**” variable in order **to erase the null values**, as it does not make sense to keep them and later we would use this column to **create a variable**, therefore it could not contain nulls. Moreover, we performed this exact filter to the variables of **plane-data** for the same reasons.

Regarding the set of datasets or dataset (except for the target variable), we used a **User Defined Function (UDF) to replace the NA values for nulls**, as we then would use the “mean imputer” and the “most frequent imputer” for the purpose of **filling our dataset** and these methods need this requirement to work properly.

Before applying the imputers mentioned, we **deleted the columns/variables that only contained null values**, as it does not make sense to have them. After that, we had to make sure all the Integer values in the dataset had the **Integer datatype**, making transformations (casting) to Integer those who were not.

Once having all the dataset values correctly for the imputer, we decided to apply the “**most frequent**” imputer for the “**Year**”, “**Month**”, “**DayOfMonth**” and “**DayOfWeek**” columns, thus transforming the **null values** of those columns following the **most frequent value**. Then, we also applied the “**mean**” imputer for the **rest of numerical columns**, those being “**DepTime**”, “**CRSArrTime**”, “**DepDelay**”, “**Distance**” and “**TaxiOut**”.

After that, we chose to transform the “**Issue\_date**” variable into a new one called “**PlaneAge**”, which will be described in the next section.

Moreover, another transformation that we performed thanks to another UDF was changing the values of “**DepTime**” and “**CRSArrTime**” **into strings** containing day parts such as “**early night**” (time between 9 PM and 23:59 PM), “**late evening**” (time between 7 PM and 9 PM), since we thought that it would be **more useful for our prediction goal** to exhibit these values in this way.

With respect to the **categorical variables**, which are “**Origin**” and “**Dest**” for the set of datasets or dataset and “**Type**”, “**Engine\_type**”, “**Aircraft\_type**”, “**Model**”, “**Issue\_date**” and “**Manufacturer**” for the plane-data dataset, we were going to apply the **One Hot Encoder (OHE)** method. To apply it, we firstly used another UDF which checked for **null values** in these categorical variables and swap them into the value “**unknown**”, to assure a correct understanding of the future one hot encoded variables. Then, we applied the OHE, which consisted of converting the **categorical variables into integers** (for the OHE to work) and then apply the proper OHE.

## 5. Variables Created

One variable created was the “**PlaneAge**” variable, which is a **subtraction from** the columns/variables “**Year**” and “**Issue\_date**”, describing the age of the plane since it was inserted in the market. This variable explains in a useful manner the **lifetime of the plane**, which we believe gives much more **relevant information** than the “Issue\_date” column/variable, that only explains the year in which the plane was **ready to be used** (do not confuse it with the previously excluded variable “**year**” from the plane-data dataset, which tells the **year in which the plane was designed** but still not in the market to use).



## 6. Machine Learning Techniques

One machine learning technique used was the **vector assembler**, in which we **extracted the features** that will be **fed to the prediction models** from the columns/variables of the datasets. After that, we **normalized** the features obtained from the vector assembler to properly work with them. Furthermore, we used a **pipeline** for the purpose of creating a **sequence of run stages** in which we could **fit and transform** the above methods all together in one sequence (indexer, one hot encoder, assembler and normalizer).

Another machine learning technique used was “**feature stepwise selection**” (FSS). Stepwise feature selection is a method of selecting a subset of features for building a machine learning model. It involves adding or removing features one at a time based on their contribution to the model's performance, with the goal of selecting a subset of features that leads to the best performing model. Regarding the selection chosen, we used **2 selectors**, first the one with mode “**False Discovery Rate**” (FDR) and then the one with “**Family-wise Error Rate**” (FWE). Both are statistical measures used to control the rate of false positives in multiple hypothesis testing. We decided to use FDR because it **decreases the number of false positives** below the **p-value** (in our case 0.05 for both) thanks to the Benjamini-Hochberg procedure. On the other hand, we used **FWE** because it performs even better, thanks to its unique way of correctly rejecting the null hypothesis when it should (in 5 out of 100 tests the null hypothesis is rejected when it should not).

In relation to the models, we used **3 different regression models**: Linear Regression, Decision Tree Regressor and Random Forest Regressor. Now, we are going to explain each of them:

**Linear Regression (LR)** is a statistical method used to model the **relationship between a dependent variable and** one or more **independent** variables. To construct it, we used the `ParamGridBuilder()` function from Spark to add the following parameters:

- **RegParam**: used to specify the value of the regularization. We used 0.01, which is the middle value of the 3 possible ones (0.1, 0.01, 0.001) as it corresponds to a strong but flexible regularization, in order to help prevent overfitting (strong part) but without constraining it too much, keeping it as accurate as possible (flexible part)
- **ElasticNetParam**: used to specify the value of L1 and L2 regularization. We chose to go with a 0.25 value, since we believe that it would provide a balance between L1 and L2, providing the model with the smoothness of the L2 regularization and the sparsity of the L1 regularization
- **MaxIter**: we decided to set a maximum number of iterations (10) to let the algorithm perform efficiently but without spending excessive time on it

Moreover, we also created **2 regression evaluators** for the purpose of **measuring how well the model performs**. The first evaluator would measure the **RMSE** metric (which measures the difference between the **predicted values** and the **actual ones**) and the second one the **R<sup>2</sup> metric** (which measures the **variability** of the **target variable** that is explained by the explanatory variables).

The second model that we utilized was **Decision Tree Regressor** (DTR), which is a famous machine learning regression algorithm that uses a **decision tree to make predictions**. In this case, we didn't provide any specific parameter as we did with LR because we decided to use the default ones. However, we also used the **2 regression evaluators** (RMSE and  $R^2$ ) that we used in LR.

Finally, the last model created was the **Random Forest Regressor** (RFR), which basically is made up of **multiple decision trees** that work together to make predictions. As we did with DTR, we chose to use the default parameters to build the model and also added the 2 regression evaluators used in LR and DTR.

## 7. Validation Process

Firstly, to **split the datasets into training and test**, we chose to split it **randomly** in an arrangement of **70/30** for **both techniques** (the FDR and the FWE). After that, we decided to use **K-fold cross validation** for the purpose of **evaluating the performance of the 3 models**. Being more precise, we selected a **5-fold cross validation**, meaning that the datasets would be partitioned into 5 equal-sized subsets, being then fed into the model to be trained and evaluated. We chose the number 5 as we believed that with the huge amount of data the dataset presented, the models would perform better with a 5-fold cross validation.

After that, we **trained** and then **tested** with the test datasets each model for both FSS (the one with FDR and the one with FWE). Then, we **evaluated** each model by **measuring the 2 metrics** that we explained before, that is, RMSE and  $R^2$ .

## 8. Model Evaluation

In this section of the report, we will discuss the results given by the **3 models** regarding the **2000 dataset**: Linear Regression, Decision Tree Regressor and Random Forest Regressor, which are the 3 models we decided to use, as already discussed before, and before speaking of the individual results of each model, we thought it would be a good idea to say beforehand that among all the **feature selection method** we implemented, **neither made a significant impact** in either performance nor accuracy for either model. This means that from now on, when we discuss the results of each training method, no mention will be made of any feature selection method whatsoever, since it would be messy, superfluous, and counterproductive.

The **linear regression model** was the first one to be tested, since it was supposed to be the simplest and fastest of the three, what we thought would be beneficial at the time of tuning the dataset processing process. However, it resulted to be not only **the fastest**, but also the one to **best predict** the target variable giving promising results, with an **RMSE** of around **12.75** and an **R2** value of around **0.89**, which resulted in being the best results we would obtain in the assignment development duration. After seeing these results, we felt confident enough to conclude the assignment, but we thought to be of good practice and convenient to test other methods, such as the ones mentioned above, to see which results they would give and study whether our predictions could improve.

The **decision tree regression model** threw quite worse results, as its' **RMSE** and **R2** values were **16.81** and **0.79** respectively, significantly far from the **12.75** and **0.89** given by the linear regression.

The **random forest regression** algorithm was, quite unexpectedly, the **worse of the three**, giving an **RMSE** of **19.17** and an **R2** of **0.726**, which we still believe to be a decent result, but still too far from the promising results thrown by decision tree and especially linear regression.

At first glance, we believed that **decision tree and random forest regressions** would offer **much better results than linear regression**. However, after thorough inspection and deep investigations, we finally understood that maybe, for **our specific use case**, a **linear regression model** could be of better use given the **strong linearity** of our data and especially due to the fact that our **data** has very **few** or even none **outliers** nor any or few noise whatsoever present in the datasets, which also could very well explain why the feature selection algorithms we tried did not have any effect in the performance of either model.

This results perfectly **met our expectations**, with a **high 0.89 R2** value in the 2000 dataset, which as already mentioned before is a decent and acceptable result. For furthermore details run the application with the desired dataset and see the output summary, as shown below with the 2000 dataset summary

```

----- SUMMARY OF THE MODELS' PERFORMANCE -----
----- LINEAR REGRESSION with FDR -----
12.759887712216546
0.8789395949246434
----- LINEAR REGRESSION with FWE -----
12.759887712216548
0.8789395949246434
----- DECISION TREE REGRESSOR with FDR -----
16.812239375533462
0.789835408754207
----- DECISION TREE REGRESSOR with FWE -----
16.812239375533462
0.789835408754207
----- RANDOM FOREST REGRESSOR with FDR -----
19.173490294383562
0.7266552270389369
----- RANDOM FOREST REGRESSOR with FWE -----
19.173490294383562
0.7266552270389369

```

## 9. Compilation Instructions

As we used a **sbt project**, it is very easy to compile and execute the assignment. After **opening the project folder**, open the **command prompt** and **execute sbt**.

```
PS C:\Users\guill\IdeaProjects\BigData-Spark-Assignment> sbt
[info] welcome to sbt 1.8.0 (Oracle Corporation Java 1.8.0_351)
[info] loading global plugins from C:\Users\guill\.sbt\1.0\plugins
[info] loading project definition from C:\Users\guill\IdeaProjects\BigData-Spark-Assignment\project
[info] loading settings for project root from build.sbt ...
[info] set current project to BigData-Spark-Assignment (in build file:/C:/Users/guill/IdeaProjects/BigData-Spark-Assignment/)
[info] sbt server started at local:sbt-server-b21fe9c9db5597a0b1e1
[info] started sbt server
```

Afterwards you should be able to **compile the project** in sbt with the next command:

```
sbt:BigData-Spark-Assignment> compile
[success] Total time: 2 s, completed 23-dic-2022 17:30:03
sbt:BigData-Spark-Assignment>
```

Finally, you will be able to run the project specifying the **Full Path** of the datasets you want to process. You will be able to specify one or multiple arguments, in which case, the program will merge them automatically and preprocess them as one whole dataset alongside the already included `plane_data.csv` file in the “resources” directory. Remember to specify the **Full Absolute Path** of your dataset as shown in the picture below, otherwise, the execution will fail, since uploading the 11 GB worth of datasets to the “resources” directory would result unproductive and difficult to manage:

**\$ run DATASET\_NAME OPTIONAL\_DATASET\_NAME\_2 ...**

```
sbt:BigData-Spark-Assignment> run 2000
```

The **script** required for **more than one dataset** lies below:

```
sbt:BigData-Spark-Assignment> run 2000 2001 2002 2006 2007 2008
```

## 10. Data Placement

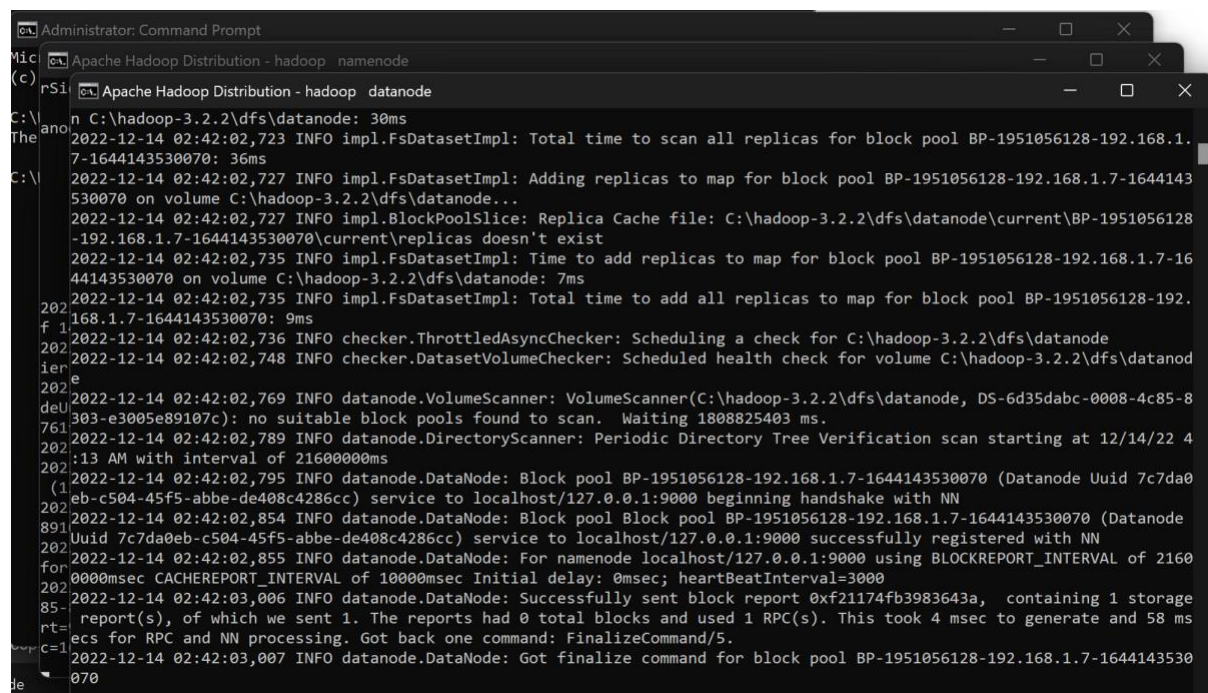
The input data which has been used in this project has been **placed in a Hadoop (HDFS) folder** that we created and called “**datasets**”.

To begin with, we will suppose Hadoop is installed and working. The **version of Hadoop** that we used was the version **3.2.2**, since we considered it was the most stable and also it was the latest one we found the “winutils.exe” implemented for.

Once having **Hadoop** installed properly, we proceed to its **execution**, executing the next commands in the command prompt:

```
$ start-dfs
```

If the command is working properly, it should show something like this:



```
Administrator: Command Prompt
Apache Hadoop Distribution - hadoop namenode
C:\>
2022-12-14 02:42:02,723 INFO impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-1951056128-192.168.1.7-1644143530070: 36ms
2022-12-14 02:42:02,727 INFO impl.FsDatasetImpl: Adding replicas to map for block pool BP-1951056128-192.168.1.7-1644143530070 on volume C:\hadoop-3.2.2\dfs\datanode...
2022-12-14 02:42:02,727 INFO impl.BlockPoolSlice: Replica Cache file: C:\hadoop-3.2.2\dfs\datanode\current\BP-1951056128-192.168.1.7-1644143530070\current\replicas doesn't exist
2022-12-14 02:42:02,735 INFO impl.FsDatasetImpl: Time to add replicas to map for block pool BP-1951056128-192.168.1.7-1644143530070 on volume C:\hadoop-3.2.2\dfs\datanode: 7ms
2022-12-14 02:42:02,735 INFO impl.FsDatasetImpl: Total time to add all replicas to map for block pool BP-1951056128-192.168.1.7-1644143530070: 9ms
2022-12-14 02:42:02,736 INFO checker.ThrottledAsyncChecker: Scheduling a check for C:\hadoop-3.2.2\dfs\datanode
2022-12-14 02:42:02,748 INFO checker.DatasetVolumeChecker: Scheduled health check for volume C:\hadoop-3.2.2\dfs\datanode
2022-12-14 02:42:02,769 INFO datanode.VolumeScanner: VolumeScanner(C:\hadoop-3.2.2\dfs\datanode, DS-6d35dabc-0008-4c85-8303-e3005e89107c): no suitable block pools found to scan. Waiting 1808825403 ms.
2022-12-14 02:42:02,789 INFO datanode.DirectoryScanner: Periodic Directory Tree Verification scan starting at 12/14/22 4:13 AM with interval of 2160000ms
2022-12-14 02:42:02,795 INFO datanode.DataNode: Block pool BP-1951056128-192.168.1.7-1644143530070 (Datanode Uuid 7c7da0eb-c504-45f5-abbe-de408c4286cc) service to localhost/127.0.0.1:9000 beginning handshake with NN
2022-12-14 02:42:02,854 INFO datanode.DataNode: Block pool BP-1951056128-192.168.1.7-1644143530070 (Datanode Uuid 7c7da0eb-c504-45f5-abbe-de408c4286cc) service to localhost/127.0.0.1:9000 successfully registered with NN
2022-12-14 02:42:02,855 INFO datanode.DataNode: For namenode localhost/127.0.0.1:9000 using BLOCKREPORT_INTERVAL of 2160000msec CACHEREPORT_INTERVAL of 10000msec Initial delay: 0msec; heartBeatInterval=3000
2022-12-14 02:42:03,006 INFO datanode.DataNode: Successfully sent block report 0xf21174fb3983643a, containing 1 storage report(s), of which we sent 1. The reports had 0 total blocks and used 1 RPC(s). This took 4 msec to generate and 58 ms for RPC and NN processing. Got back one command: FinalizeCommand/5.
2022-12-14 02:42:03,007 INFO datanode.DataNode: Got finalize command for block pool BP-1951056128-192.168.1.7-1644143530070
```

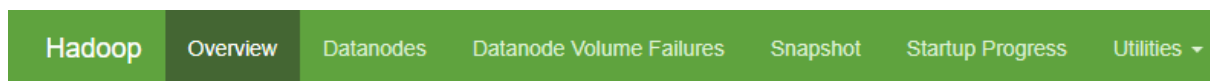
The next command we needed to execute in another command prompt was:

```
$ start-yarn
```

If the command is working properly, it should show something similar to these windows:

```
Administrator: Command Prompt
Apache Hadoop Distribution - yarn resourcemanager
2022-12-14 02:42:52,293 INFO org.apache.hadoop.yarn.server.nodemanager.webapp.JAXBContextResolver
2022-12-14 02:42:52,293 INFO com.sun.jersey.server.impl.application.WebApplicationImpl _initiate
2022-12-14 02:42:52,293 INFO com.sun.jersey.guice.spi.container.GuiceComponentProviderFactory getComponentProvider
2022-12-14 02:42:52,293 INFO org.apache.hadoop.yarn.server.nodemanager.webapp.JAXBContextResolver to GuiceManagedComponentProvider with
2022-12-14 02:42:52,293 INFO the scope "Singleton"
2022-12-14 02:42:52,293 INFO com.sun.jersey.guice.spi.container.GuiceComponentProviderFactory getComponentProvider
2022-12-14 02:42:52,293 INFO org.apache.hadoop.yarn.webapp.GenericExceptionHandler to GuiceManagedComponentProvider with the scope "Sin
2022-12-14 02:42:52,293 INFO gleton"
2022-12-14 02:42:52,293 INFO com.sun.jersey.guice.spi.container.GuiceComponentProviderFactory getComponentProvider
2022-12-14 02:42:52,293 INFO org.apache.hadoop.yarn.server.nodemanager.webapp.NMWebServices to GuiceManagedComponentProvider with the s
2022-12-14 02:42:52,293 INFO cope "Singleton"
2022-12-14 02:42:52,293 INFO handler.ContextHandler: Started o.e.j.w.WebAppContext@750ff7d3{node,/,file:///C:/Users/javie/AppData/Local/Temp/jetty-0_0_0-8042-_-any-9113242696808407249.dir/webapp/,AVAILABLE}{jar:file:/C:/hadoop-3.2.2/share
2022-12-14 02:42:52,293 INFO /hadoop/yarn/hadoop-yarn-common-3.2.2.jar!/webapps/node}
2022-12-14 02:42:52,293 INFO server.AbstractConnector: Started ServerConnector@35fe2125{HTTP/1.1,[http/1.1]}{0.0.0.0:8042}
2022-12-14 02:42:52,293 INFO server.Server: Started @5452ms
2022-12-14 02:42:52,293 INFO webapp.WebApps: Web app node started at 8042
2022-12-14 02:42:52,293 INFO nodemanager.NodeStatusUpdaterImpl: Node ID assigned is : 192.168.1.36:63187
2022-12-14 02:42:52,293 INFO util.JvmPauseMonitor: Starting JVM pause monitor
2022-12-14 02:42:52,293 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0:8031
2022-12-14 02:42:52,293 INFO nodemanager.NodeStatusUpdaterImpl: Registering with RM using containers :[]
2022-12-14 02:42:52,293 INFO security.NMContainerTokenSecretManager: Rolling master-key for container-tokens, got key wi
2022-12-14 02:42:52,293 INFO th id 1021187797
2022-12-14 02:42:52,293 INFO security.NMTokenSecretManagerInNM: Rolling master-key for container-tokens, got key with id
2022-12-14 02:42:52,293 INFO 664141746
2022-12-14 02:42:52,293 INFO nodemanager.NodeStatusUpdaterImpl: Registered with ResourceManager as 192.168.1.36:63187 wi
2022-12-14 02:42:52,293 INFO th total resource of <memory:8192, vCores:8>
```

Now we can **test if Hadoop is working properly** by acceding the next webpage:



## Overview 'localhost:9000' (active)

Started:	Thu Dec 22 14:11:22 +0100 2022
Version:	3.2.2, r7a3bc90b05f257c8ace2f76d74264906f0f7a932
Compiled:	Sun Jan 03 10:26:00 +0100 2021 by hexiaoqiao from branch-3.2.2
Cluster ID:	CID-e761f835-5479-48e3-a70d-1b47acc13d6f
Block Pool ID:	BP-1951056128-192.168.1.7-1644143530070

## Summary

Security is off.

Safe mode is ON. The reported blocks 3 has reached the threshold 0,9990 of total blocks 3. The minimum number of live d: will be turned off automatically in 2 seconds.

6 files and directories, 3 blocks (3 replicated blocks, 0 erasure coded block groups) = 9 total filesystem object(s).



Then we proceeded to **create** a **folder “datasets”** in the HDFS and upload the datasets we want to process with the application. It is possible to do that both with the **HDFS interface** or the **command line**:

```
C:\Users\javie>hdfs dfs -mkdir hdfs://localhost:9000/datasets
```

```
C:\Users\javie>hdfs dfs -put 2008.csv hdfs://localhost:9000/datasets
```

Hadoop

Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse the file system  
Logs  
Log Level  
Metrics  
Configuration  
Process Thread Dump

## Browse Directory

/datasets

Go!

Show 25 entries

Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
	-rw-r--r--	root	hadoop	223.21 MB	Dec 16 02:41	1	128 MB	<u>2008.csv</u>

Showing 1 to 1 of 1 entries

Previous 1 Next

Hadoop, 2021.

## 11. Conclusions

This assignment has allowed us to put into practice the **concepts** learnt during the **theory lessons**, and to get **proficiency** in the use of **Spark** and other related Big Data technologies, as well as Hadoop File System.

Also, we found very interesting making a **Spark application** capable of **loading and processing an input data**, for later creating and validating a machine learning prediction model for a real-world problem, using **real-world data**. Working with real data is in the end very valuable in order to know how does this data look like and how difficult is to work with it and the time needed to analyze it.

The construction of a **prediction model** is, in the end, a typical Data Science problem, and all the knowledge applied to this project cover most of the aspects that are needed to successfully develop it. Moreover, being already familiar with all the processes required to create the model and indeed performing it allowed us to focus on a high-quality predictor which could make the most out of our accuracy results.

To end up, we really think that this project has taught us **interesting insights** not only about how to work with Spark, but also on **managing** to use **huge quantities of data**, **manipulate**, **transform**, and **evaluate them**. Finally, one last lesson that this assignment has force us to learn is the ability to manage our **time** wisely and to **work together** more productively.