

Tutorial 06 – Análise com *Cloudera Hadoop* e *RStudio* - *Big Data*

Este tutorial demonstra como usar o [sparklyr](#) com um **Cloudera Hadoop & Spark**. Os dados são baixados da *web* e armazenados nas tabelas de **Hive** no **HDFS** que podem estar em vários nós de trabalho. O servidor **RStudio** é instalado no nó mestre e orquestra a análise no **Spark**.

Instalação e Configuração do *RStudio*

Primeiramente é necessário instalar todas as bibliotecas e aplicações que são utilizadas pelo **RStudio** na conexão com o **Cloudera Hadoop** e suas ferramentas.

Dirija-se ao terminal do sistema operacional **CentOS** e execute os seguintes comandos de instalação:

```
sudo yum install R wget xml2 curl httr libxml2 libxml2-devel libcurl libcurl-devel libssl-dev -y
```

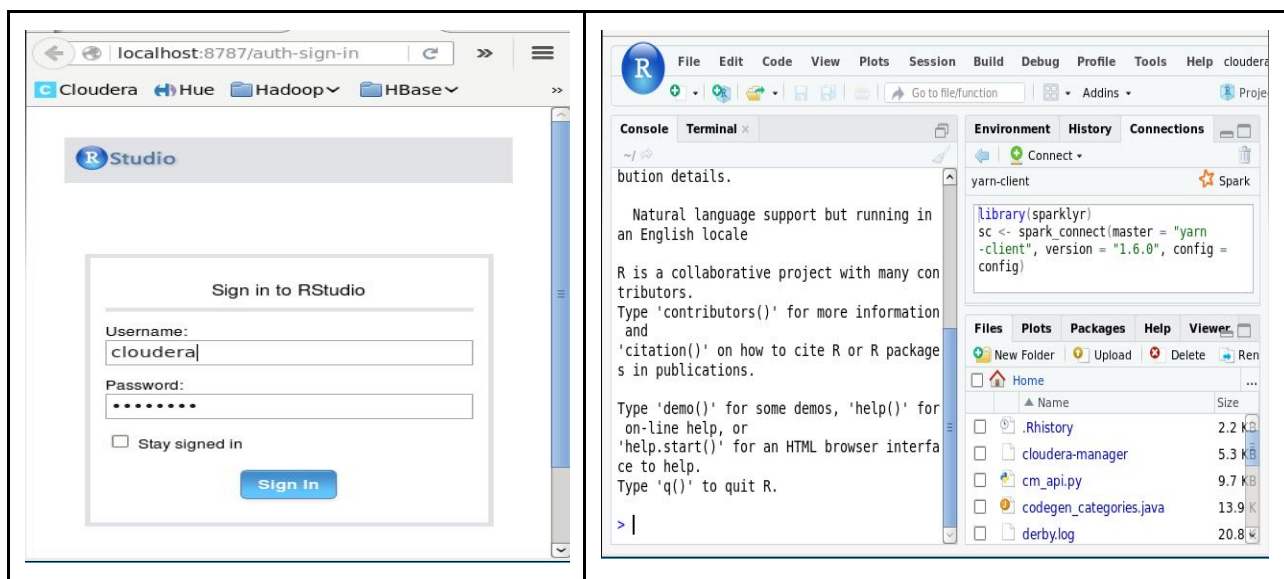
Também precisamos instalar o **RStudio Server** com os seguintes comandos no terminal:

```
sudo wget https://download2.rstudio.org/rstudio-server-rhel-1.1.383-x86_64.rpm
sudo yum install --nogpgcheck rstudio-server-rhel-1.1.383-x86_64.rpm -y
```

Como este tutorial foi desenvolvido para ser executado em uma máquina virtual local e não será utilizado um cloud precisamos **informar ao Spark** onde se encontram os **metadados do Hive**, por isto é necessário digitar o seguinte comando de cópia no terminal:

```
sudo cp /etc/hive/conf/hive-site.xml /usr/lib/spark/conf/.
```

Agora é possível acessar o **RStudio** no navegador no caminho <http://localhost:8787> e o usuário e senha de com direito de acesso ao **Hadoop**. Para este tutorial vamos utilizar o usuário e senha padrão do Cloudera que são **cloudera** para usuário e **cloudera** para senha.



Instale os pacotes do **sparklyr** (conector **spark**) e o **ggplot2** (gráficos) no **console** do **RStudio** e reinicie o servidor com o comando "**sudo reboot**" no terminal para assumir as novas configurações.

```
install.packages("ggplot2")
install.packages("sparklyr")
```

Dados do Hive

Depois de instalado e configurado o **RStudio Server** é necessário carregar alguns tabelas no **Hive**. Esta demonstração irá carregar e utilizar 3 tabelas no **Hive**. Os nomes das tabelas são: **flights**, **airlines** e **airports**. Usando **Hue**, poderemos ver as tabelas carregadas. As tabelas que foram carregadas em outro momento para o **Hive** também estarão disponíveis para o **RStudio**.

Os comandos abaixo devem ser executados no terminal. Eles tem por objetivo baixar alguns **datasets** da **web** e armazená-los num diretório temporário para depois ser carregado no **HDFS**.

```
# Cria diretório temporário para download
mkdir /tmp/flights

# Faz download dos dados de vôo (flight) por ano
for i in {2006..2008}
do
    echo "$(date) $i Download"
    fnam=$i.csv.bz2
    wget -O /tmp/flights/$fnam http://stat-computing.org/dataexpo/2009/$fnam
    echo "$(date) $i Unzip"
    bunzip2 /tmp/flights/$fnam
done

# Download dos dados da linhas aéreas (airlines)
wget -O /tmp/airlines.csv
http://www.transtats.bts.gov/Download_Lookup.asp?Lookup=L_UNIQUE_CARRIERS

# Download dos dados dos aeroportos
wget -O /tmp/airports.csv
https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat
```

Agora que baixamos os datasets vamos armazená-los no sistema de arquivos **HDFS** do **Hadoop** executando os comandos abaixo no terminal.

```
hadoop fs -mkdir -p /user/hive/warehouse/flights
hadoop fs -put /tmp/flights/2006.csv /user/hive/warehouse/flights/2006.csv
hadoop fs -put /tmp/flights/2007.csv /user/hive/warehouse/flights/2007.csv
hadoop fs -put /tmp/flights/2008.csv /user/hive/warehouse/flights/2008.csv

hadoop fs -put /tmp/airlines.csv /user/hive/warehouse/airlines.csv
hadoop fs -put /tmp/airports.csv /user/hive/warehouse/airports.csv
```

Após inserido os arquivos do *dataset* no **HDFS** precisamos carregá-los para tabelas dos metadados do **Hive**. Entre no editor de **Query** do **Hive** disponível no **Hue** e execute os seguintes comandos para criar a tabela (**CREATE TABLE**) de voos (**flights**) e também carregar os dados (**LOAD DATA**) do **HDFS** para os metadados do **Hive**.

```
CREATE EXTERNAL TABLE IF NOT EXISTS flights
```

```
( year int,  
  month int,  
  dayofmonth int,  
  dayofweek int,  
  deptime int,  
  crsdeptime int,  
  arrtime int,  
  crsarrrtime int,  
  uniquecarrier string,  
  flightnum int,  
  tailnum string,  
  actualelapsedtime int,  
  crselapsedtime int,  
  airtime string,  
  arrdelay int,  
  depdelay int,  
  origin string,  
  dest string,  
  distance int,  
  taxiin string,  
  taxiout string,  
  cancelled int,  
  cancellationcode string,  
  diverted int,  
  carrierdelay string,  
  weatherdelay string,  
  nasdelay string,  
  securitydelay string,  
  lateaircraftdelay string
```

```
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
TBLPROPERTIES("skip.header.line.count"="1");
```

```
LOAD DATA INPATH '/user/hive/warehouse/flights/2006.csv' INTO TABLE flights;  
LOAD DATA INPATH '/user/hive/warehouse/flights/2007.csv' INTO TABLE flights;  
LOAD DATA INPATH '/user/hive/warehouse/flights/2008.csv' INTO TABLE flights;
```

Também é necessário carregar os dados das linhas aéreas (*airlines*) no **Hive**.

```
CREATE EXTERNAL TABLE IF NOT EXISTS airlines
(
  Code string,
  Description string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
  "separatorChar" = '\,',
  "quoteChar" = '\"'
)
STORED AS TEXTFILE
tblproperties("skip.header.line.count"="1");

LOAD DATA INPATH '/user/hive/warehouse/airlines.csv' INTO TABLE airlines;
```

Necessário carregar os aeroportos (*airports*) também.

```
CREATE EXTERNAL TABLE IF NOT EXISTS airports
(
  id string,
  name string,
  city string,
  country string,
  faa string,
  icao string,
  lat double,
  lon double,
  alt int,
  tz_offset double,
  dst string,
  tz_name string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
  "separatorChar" = '\,',
  "quoteChar" = '\"'
)
STORED AS TEXTFILE;

LOAD DATA INPATH '/user/hive/warehouse/airports.csv' INTO TABLE airports;
```

Agora que os dados foram carregados, basta se conectar no **RStudio** e **executar** os comandos abaixo no **console**:

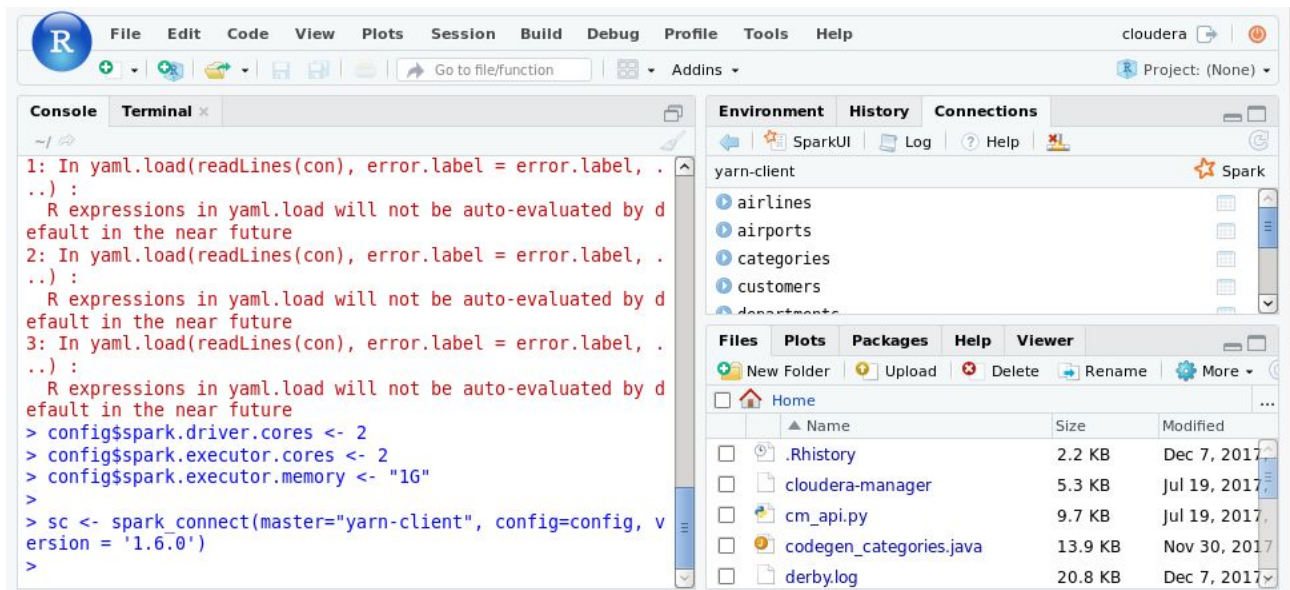
```
library(sparklyr)
library(dplyr)
library(ggplot2)

Sys.setenv(SPARK_HOME = "/usr/lib/spark")
Sys.setenv(HADOOP_CONF_DIR = '/etc/hadoop/conf')
Sys.setenv(YARN_CONF_DIR = '/etc/hadoop/conf')

config <- spark_config()
config$spark.driver.cores <- 2
config$spark.executor.cores <- 2
config$spark.executor.memory <- "1G"

sc <- spark_connect(master="yarn-client", config=config, version = '1.6.0')
```

Uma vez que estiver conectado, você verá o painel *Spark* aparecer junto com suas tabelas do **Hive**. Você pode inspecionar suas tabelas clicando no ícone de dados.



The screenshot shows the RStudio IDE interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The top right shows the user 'cloudera' and the project 'Project: (None)'. The left pane is split into 'Console' and 'Terminal'. The 'Console' tab is active, showing the execution of R commands to connect to Spark and configure it. The right pane is split into 'Environment', 'History', and 'Connections'. The 'Environment' tab is active, showing the 'yarn-client' connection and a list of tables: airlines, airports, categories, customers, and departments. Below the Environment pane is a 'Files' pane showing a file explorer view of the home directory, listing files like .Rhistory, cloudera-manager, cm_api.py, codegen_categories.java, and derby.log.

```
1: In yaml.load(readLines(con), error.label = error.label, .
..) :
  R expressions in yaml.load will not be auto-evaluated by d
efault in the near future
2: In yaml.load(readLines(con), error.label = error.label, .
..) :
  R expressions in yaml.load will not be auto-evaluated by d
efault in the near future
3: In yaml.load(readLines(con), error.label = error.label, .
..) :
  R expressions in yaml.load will not be auto-evaluated by d
efault in the near future
> config$spark.driver.cores <- 2
> config$spark.executor.cores <- 2
> config$spark.executor.memory <- "1G"
>
> sc <- spark_connect(master="yarn-client", config=config, v
ersion = '1.6.0')
>
```

Name	Size	Modified
.Rhistory	2.2 KB	Dec 7, 2017
cloudera-manager	5.3 KB	Jul 19, 2017
cm_api.py	9.7 KB	Jul 19, 2017
codegen_categories.java	13.9 KB	Nov 30, 2017
derby.log	20.8 KB	Dec 7, 2017

Existe alguma evidência que sugere que algumas companhias aéreas comprem tempo em voo?

Esta análise prevê o tempo adquirido em voo pela companhia aérea.

Vamos criar um modelo de dados para trabalhar. Execute os comando abaixo no console do *RStudio*. Este comando esta filtrando os dados para armazenar apenas os registros que serão usados no modelo final. Faz um *join* com as descrições do transportador para referência. Cria uma nova variável chamada *gain* (ganho) que representa a quantidade de tempo que ganhou (ou perdeu) em voo.

```
# Filtre os registros e crie a variável 'gain'
model_data <- tbl(sc,'flights') %>%
  filter(!is.na(arrdelay) & !is.na(depdelay) & !is.na(distance)) %>%
  filter(depdelay > 15 & depdelay < 240) %>%
  filter(arrdelay > -60 & arrdelay < 360) %>%
  filter(year >= 2003 & year <= 2007) %>%
  left_join(tbl(sc,'airlines'), by = c("uniquecarrier" = "code")) %>%
  mutate(gain = depdelay - arrdelay) %>%
  select(year, month, arrdelay, depdelay, distance, uniquecarrier, description, gain);

# Resumir dados pelo operador (carrier)
model_data %>%
  group_by(uniquecarrier) %>%
  summarize(description = min(description), gain=mean(gain),
             distance=mean(distance), depdelay=mean(depdelay)) %>%
  select(description, gain, distance, depdelay) %>%
  arrange(gain)
```

O modelo que resultante do programa acima deverá retornar os seguintes valores:

```
# Source:   lazy query [?? x 4]
# Database:  spark_connection
# Ordered by: gain
```

	description	gain	distance	depdelay
	<chr>	<dbl>	<dbl>	<dbl>
1	ATA Airlines d/b/a ATA	-5.5679651	1240.7219	61.84391
2	Northwest Airlines Inc.	-3.1134556	779.1926	48.84979
3	Envoy Air	-2.2056576	437.0883	54.54923
4	PSA Airlines Inc.	-1.9267647	500.6955	55.60335
5	ExpressJet Airlines Inc. (1)	-1.5886314	537.3077	61.58386
6	JetBlue Airways	-1.3742524	1087.2337	59.80750
7	SkyWest Airlines Inc.	-1.1265678	419.6489	54.04198
8	Delta Air Lines Inc.	-0.9829374	956.9576	50.19338
9	American Airlines Inc.	-0.9631200	1066.8396	56.78222
10	AirTran Airways Corporation	-0.9411572	665.6574	53.38363

```
# ... with more rows
```

Vamos treinar um modelo linear.

Execute os comandos abaixo no *RStudio* para prever o tempo ganho ou perdido no voo em função da distância, atraso de partida e companhia aérea.

```
# Particionar os dados em conjuntos de treinamento e validação
model_partition <- model_data %>%
  sdf_partition(train = 0.8, valid = 0.2, seed = 5555)

# Encontrar / Calcular um modelo linear
ml1 <- model_partition$train %>%
  ml_linear_regression(gain ~ distance + depdelay + uniquecarrier)

# Resumir o modelo linear
summary(ml1)
```

Resultado:

```
Call: ml_linear_regression(., gain ~ distance + depdelay + uniquecarrier)
```

Deviance Residuals: (approximate):

Min	1Q	Median	3Q	Max
-304.269	-5.801	2.628	9.745	104.130

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-1.26566581	0.10385870	-12.1864	< 2.2e-16	***
distance	0.00308711	0.00002404	128.4155	< 2.2e-16	***
depdelay	-0.01397013	0.00028816	-48.4812	< 2.2e-16	***
uniquecarrier_AA	-2.18483090	0.10985406	-19.8885	< 2.2e-16	***
uniquecarrier_AQ	3.14330242	0.29114487	10.7964	< 2.2e-16	***
uniquecarrier_AS	0.09210380	0.12825003	0.7182	0.4726598	
uniquecarrier_B6	-2.66988794	0.12682192	-21.0523	< 2.2e-16	***
uniquecarrier_CO	-1.11611186	0.11795564	-9.4621	< 2.2e-16	***
uniquecarrier_DL	-1.95206198	0.11431110	-17.0767	< 2.2e-16	***
uniquecarrier_EV	1.70420830	0.11337215	15.0320	< 2.2e-16	***
uniquecarrier_F9	-1.03178176	0.15384863	-6.7065	1.994e-11	***
uniquecarrier_FL	-0.99574060	0.12034738	-8.2739	2.220e-16	***
uniquecarrier_HA	-1.16970713	0.34894788	-3.3521	0.0008020	***
uniquecarrier_MQ	-1.55569040	0.10975613	-14.1741	< 2.2e-16	***
uniquecarrier_NW	-3.58502418	0.11534938	-31.0797	< 2.2e-16	***
uniquecarrier_UH	1.40654707	0.12024950	11.6972	< 2.2e-16	***

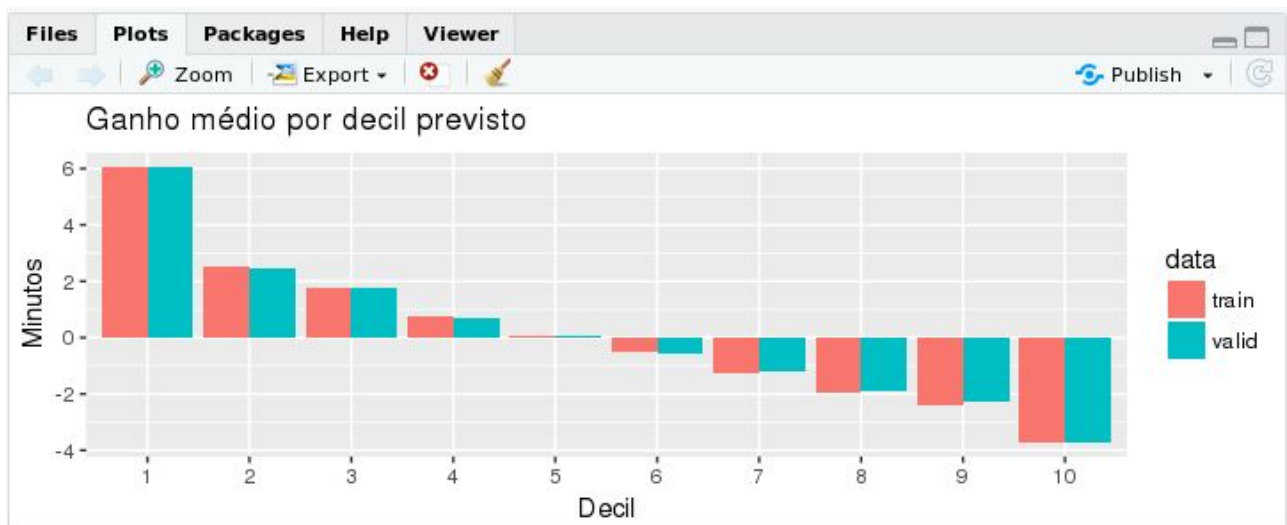
Avaliando o desempenho do modelo

Compare o desempenho do modelo usando os dados de validação.

```
# Calcula os ganhos médios por decil previsto
model_deciles <- lapply(model_partition, function(x) {
  sdf_predict(ml1, x) %>%
    mutate(decile = ntile(desc(prediction), 10)) %>%
    group_by(decile) %>%
    summarize(gain = mean(gain)) %>%
    select(decile, gain) %>%
    collect()
})

# Cria um conjunto de dados de resumo para plotar
deciles <- rbind(
  data.frame(data = 'train', model_deciles$train),
  data.frame(data = 'valid', model_deciles$valid),
  make.row.names = FALSE
)

# Plota ganhos médios por decomposição prevista
deciles %>%
  ggplot(aes(factor(decile), gain, fill = data)) +
  geom_bar(stat = 'identity', position = 'dodge') +
  labs(title = 'Ganho médio por decil previsto', x = 'Decil', y = 'Minutos')
```



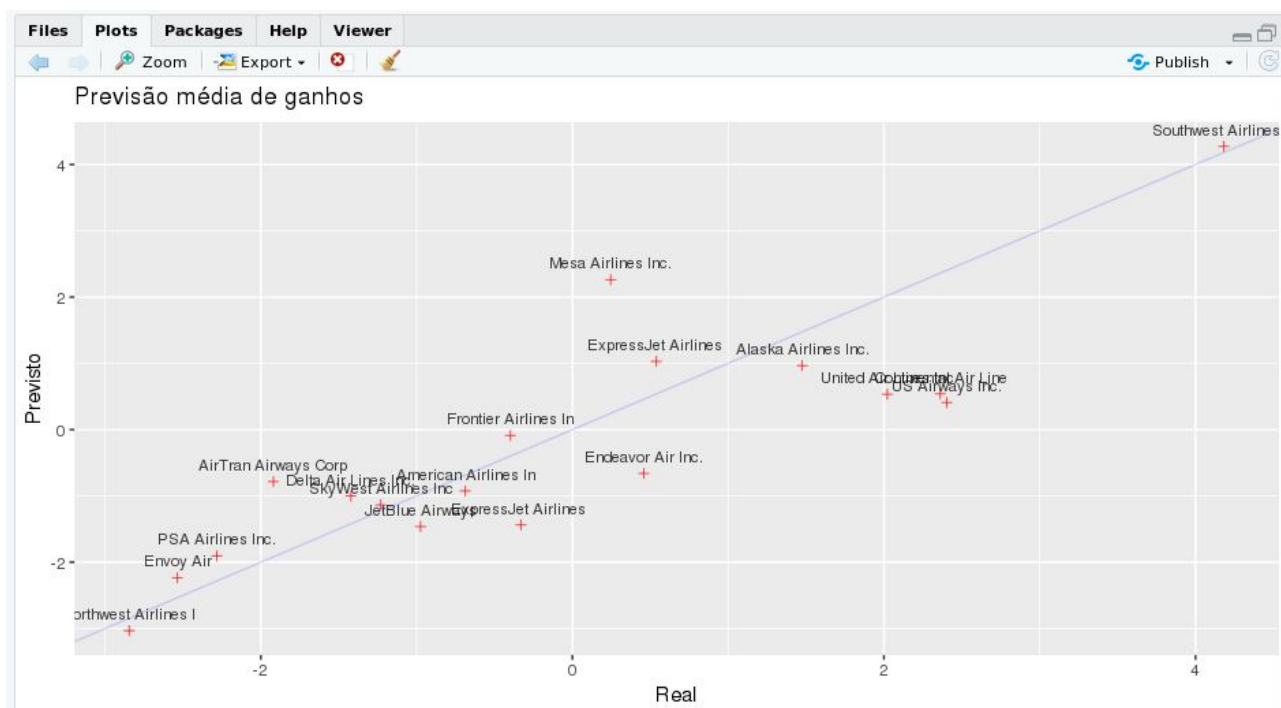
Visualizando as previsões

Compara os ganhos reais com os ganhos previstos para uma amostra fora do tempo.

```
# Seleciona dados de uma amostra fora do tempo
data_2008 <- tbl(sc,'flights') %>%
  filter(!is.na(arrdelay) & !is.na(depdelay) & !is.na(distance)) %>%
  filter(depdelay > 15 & depdelay < 240) %>%
  filter(arrdelay > -60 & arrdelay < 360) %>%
  filter(year == 2008) %>%
  left_join(tbl(sc,'airlines'), by = c("uniquecarrier" = "code")) %>%
  mutate(gain = depdelay - arrdelay) %>%
  select(year, month, arrdelay, depdelay, distance, uniquecarrier, description, gain,
origin,dest)

# Resume os dados pelo operador
carrier <- sdf_predict(ml1, data_2008) %>%
  group_by(description) %>%
  summarize(gain = mean(gain), prediction = mean(prediction), freq = n()) %>%
  filter(freq > 10000) %>%
  collect

# Traça os ganhos reais e ganhos previstos pela companhia aérea
ggplot(carrier, aes(gain, prediction)) +
  geom_point(alpha = 0.75, color = 'red', shape = 3) +
  geom_abline(intercept = 0, slope = 1, alpha = 0.15, color = 'blue') +
  geom_text(aes(label = substr(description, 1, 20)), size = 3, alpha = 0.75, vjust = -1) +
  labs(title='Previsão média de ganhos', x = 'Real', y = 'Previsto')
```



Compartilhamento de *Insights*

Este modelo linear simples contém uma riqueza de informações detalhadas sobre transportadoras, distâncias percorridas e atrasos nos vôos. Essas informações detalhadas podem ser transmitidas para uma audiência não técnica através de um painel flexível e interativo.

Construindo um painel

Agregando os dados marcados por origem, destino e companhia aérea. Salvando os dados agregados e analisados que podem ser verificado em um painel flexível R Markdown no [flexdashboard](#).

```
# Resumo por origem, destino e operador
summary_2008 <- sdf_predict(ml1, data_2008) %>%
  rename(carrier = uniquecarrier, airline = description) %>%
  group_by(origin, dest, carrier, airline) %>%
  summarize(
    flights = n(),
    distance = mean(distance),
    avg_dep_delay = mean(depdelay),
    avg_arr_delay = mean(arrdelay),
    avg_gain = mean(gain),
    pred_gain = mean(prediction)
  )

# Coletar e salvar os objetos em um arquivo de voos
pred_data <- collect(summary_2008)
airports <- collect(select(tbl(sc,'airports'), name, faa, lat, lon))
ml1_summary <- capture.output(summary(ml1))
save(pred_data, airports, ml1_summary, file = '/tmp/flights_pred_2008.RData')
```