

Data preparation

- Set up the cluster
- Download flights data
- Create Hive tables
- Connect to Spark

Data analysis

- Cache the tables into memory
- Create a model data set
- Train a linear model
- Assess model performance
- Visualize predictions

Share Insights

- Build dashboard
- Publish dashboard

Using sparklyr with an Apache Spark cluster

This document demonstrates how to use `sparklyr` with an Apache Spark cluster. Data are downloaded from the web and stored in Hive tables on HDFS across multiple worker nodes. RStudio Server is installed on the master node and orchestrates the analysis in spark. Here is the basic workflow.



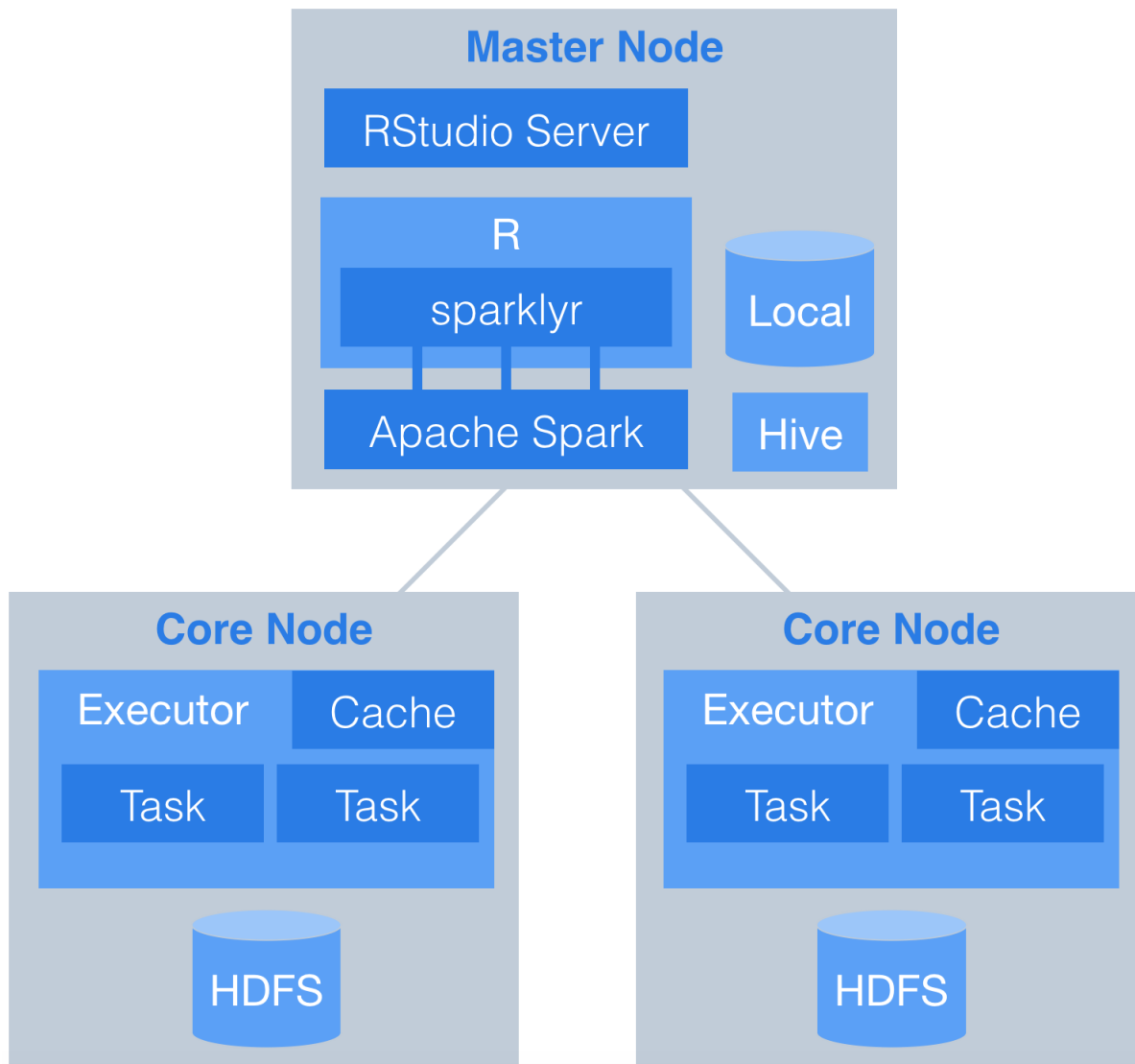
Data preparation

Set up the cluster

This demonstration uses Amazon Web Services (AWS), but it could just as easily use Microsoft, Google, or any other provider. We will use Elastic Map Reduce (EMR) to easily set up a cluster with two core nodes and one master node. Nodes use virtual servers from the Elastic Compute Cloud (EC2). *Note: There is no free tier for EMR, charges will apply.*

Before beginning this setup we assume you have:

- Familiarity with and access to an AWS account
- Familiarity with basic linux commands
- Sudo privileges in order to install software from the command line



Build an EMR cluster

Before beginning the EMR wizard setup, make sure you create the following in AWS:

- An AWS key pair (.pem key) so you can SSH into the EC2 master node
- A security group that gives you access to port 22 on your IP and port 8787 from anywhere

Create Security Group

Security group name

spark-demo

Description

Spark Demo

VPC

No VPC

Security group rules:

Inbound

Type	Protocol	Port Range	Source
SSH	TCP	22	My IP 68.134.36.58/32
Custom TCP Rule	TCP	8787	Anywhere 0.0.0.0/0

Add Rule

Cancel

Create

Step 1: Select software

Make sure to select Hive and Spark as part of the install. Note that by choosing Spark, R will also be installed on the master node as part of the distribution.

Create Cluster - Advanced Options [Go to quick options](#)

Step 1: Software and Steps

Step 2: Hardware

Step 3: General Cluster Settings

Step 4: Security

Software Configuration

Vendor ☒ Amazon ☐ MapRRelease

- | | | |
|--|--|--|
| <input checked="" type="checkbox"/> Hadoop 2.7.2 | <input type="checkbox"/> Tez 0.8.3 | <input type="checkbox"/> Ganglia 3.7.2 |
| <input type="checkbox"/> Presto-Sandbox 0.148 | <input type="checkbox"/> HBase 1.2.1 | <input checked="" type="checkbox"/> Pig 0.14.0 |
| <input checked="" type="checkbox"/> Hive 1.0.0 | <input type="checkbox"/> Mahout 0.12.2 | <input type="checkbox"/> Sqoop-Sandbox 1.4.6 |
| <input type="checkbox"/> Zeppelin-Sandbox 0.5.6 | <input checked="" type="checkbox"/> Hue 3.7.1 | <input type="checkbox"/> Phoenix 4.7.0 |
| <input checked="" type="checkbox"/> Spark 1.6.2 | <input type="checkbox"/> ZooKeeper-Sandbox 3.4.8 | <input type="checkbox"/> HCatalog 1.0.0 |
| <input type="checkbox"/> Oozie-Sandbox 4.2.0 | | |

Edit software settings (optional)

☒ Enter configuration ☐ Load JSON from S3

```
classification=config-file-name,properties=[myKey1=myValue1,myKey2=myValue2]
```

Add steps (optional)

Step type ☐ Auto-terminate cluster after the last step is completed

Cancel

Step 2: Select hardware

Install 2 core nodes and one master node with m3.xlarge 80 GiB storage per node. You can easily increase the number of nodes later.

Create Cluster - Advanced Options [Go to quick options](#)[Step 1: Software and Steps](#)**Step 2: Hardware**[Step 3: General Cluster Settings](#)[Step 4: Security](#)

Hardware Configuration ⓘ

If you need more than 20 EC2 instances, [complete this form](#).Network [Create a VPC](#) ⓘEC2 availability zone

Type	Name	EC2 instance type	Instance count	Storage per instance	Request spot
Master	Master instance group - 1	<input type="text" value="m3.xlarge"/>	1	80 GiB Add EBS volumes	<input type="checkbox"/>
Core	Core instance group - 2	<input type="text" value="m3.xlarge"/>	2	80 GiB Add EBS volumes	<input type="checkbox"/>
Task	Task instance group - 3	<input type="text" value="m3.xlarge"/>	0	80 GiB Add EBS volumes	<input type="checkbox"/>

[Add task instance group](#)[Cancel](#)[Previous](#)[Next](#)

Step 3: Select general cluster settings

Click next on the general cluster settings.

Create Cluster - Advanced Options [Go to quick options](#)[Step 1: Software and Steps](#)[Step 2: Hardware](#)**Step 3: General Cluster Settings**[Step 4: Security](#)

General Options

Cluster name ☒ Logging ⓘS3 folder ☒ Debugging ⓘ☒ Termination protection ⓘ

Tags ⓘ

Key	Value (optional)
<input type="text" value="Add a key to create a tag"/>	<input type="text"/>

Additional Options

☐ EMRFS consistent view ⓘ[▶ Bootstrap Actions](#)[Cancel](#)[Previous](#)[Next](#)

Step 4: Select security

Enter your EC2 key pair and security group. Make sure the security group has ports 22 and 8787 open.

Create Cluster - Advanced Options

[Go to quick options](#)[Step 1: Software and Steps](#)[Step 2: Hardware](#)[Step 3: General Cluster Settings](#)**Step 4: Security**

Security Options

EC2 key pair spark-demo☒ Cluster visible to all IAM users in account

Permissions

☒ Default ☐ Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role EMR_DefaultRoleEC2 instance profile EMR_EC2_DefaultRole

EC2 Security Groups

An EC2 security group acts as a virtual firewall for your cluster nodes to control inbound and outbound traffic. There are two types of security groups you can configure, [EMR managed security groups](#) and [additional security groups](#). EMR will [automatically update](#) the rules in the EMR managed security groups in order to launch a cluster. [Learn more](#).

Type	EMR managed security groups EMR will automatically update the selected group	Additional security groups EMR will not modify the selected groups
Master	sg-31ba3902 (spark-demo)	No security groups selected
Core & Task	sg-31ba3902 (spark-demo)	No security groups selected

EMR will [automatically update](#) the rules in the custom EMR managed security groups selected above to launch a cluster
[Create a security group](#)

Encryption Options

[Cancel](#)[Previous](#)[Create cluster](#)

Connect to EMR

The cluster page will give you details about your EMR cluster and instructions on connecting.

[Add step](#)
[Resize](#)
[Clone](#)
[Terminate](#)
[AWS CLI export](#)
Cluster: Spark Demo Waiting Cluster ready after last step completed.[C](#)Connections: [Enable Web Connection](#) – Spark History Server, Ganglia, Resource Manager ... (View All)Master public DNS: ec2-52-11-18-196.us-west-2.compute.amazonaws.com [SSH](#)Tags: -- [View All](#) / [Edit](#)

Summary

ID: j-2ZJ0QKI8DLT7R
 Creation 2016-07-30 14:42
 date: (UTC-4)
 Elapsed 8 minutes
 time:
 Auto- No
 terminate:
 Termination Off [Change](#)
 protection:

Configuration Details

Release emr-4.7.2
 label:
 Hadoop Amazon 2.7.2
 distribution:
 ApplicationsGanglia 3.7.2, Spark
 1.6.2
 Log URI: s3://aws-logs-263245908434-us-west-2/elasticmapreduce/
 EMRFS Disabled
 consistent
 view:

Network and Hardware

Availability us-west-2a
 zone:
 Subnet ID: subnet-eddb079a
 Master: Running 1 m3.xlarge
 Core: Running 2 m3.xlarge
 Task: --

Security and Access

Key name: solutions-eng
 EC2 EMR_EC2_DefaultR
 instance ole
 profile:
 EMR role: EMR_DefaultRole
 Visible to all All [Change](#)
 users:
 Security sg-68e1bc0e
 groups for (ElasticMapReduce-
 Master: master)
 Security sg-77e1bc11
 groups for (ElasticMapReduce-
 Core & slave)
 Task:

[Monitoring](#)[Hardware](#)[Steps](#)[Configurations](#)[Bootstrap Actions](#)

Connect to the master node via SSH using your key pair. Once you connect you will see the EMR welcome.

```
# Log in to master node
ssh -i ~/spark-demo.pem hadoop@ec2-52-10-102-11.us-west-2.compute.amazonaws.com
```

```
Nathans-MBP-2:~ nwstephens$ ssh -i ~/spark-demo.pem hadoop@ec2-52-10-102-11.us-west-2.compute.amazonaws.com
Last login: Sat Jul 30 19:01:37 2016 from pool-68-134-36-58.bltmmd.fios.verizon.net
```

```
  _| _|_)
 _| ( /  Amazon Linux AMI
  _|\_|_|
```

```
https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/
16 package(s) needed for security, out of 26 available
Run "sudo yum update" to apply all updates.
```

```
EEEEEEEEEEEEEEEEEEEE MMMMMMM      MMMMMMM RRRRRRRRRRRRRR
E::::::::::::::::::::E M::::::::M      M::::::::M R:::::::::R
EE::::::::EEEEEEEE::E M::::::::M      M::::::::M R:::RRRRR:::R
E:::E      EEEEE M::::::::M      M::::::::M RR:::R      R:::R
E:::E      M::::::::M      M::M::::::::M      R:::R      R:::R
E:::EEEEEEEE M:::M M:::M M:::M M:::M      R::RRRRR:::R
E::::::::::::E M:::M M::M::M M:::M      R:::::::::RR
E:::EEEEEEEE M:::M M:::M M:::M      R::RRRRR:::R
E:::E      M:::M M:::M M:::M      R:::R      R:::R
E:::E      EEEEE M:::M      MMM      M:::M      R:::R      R:::R
EE::::::::EEEEEEEE::E M:::M      M:::M      R:::R      R:::R
E::::::::::::E M:::M      M:::M      RR:::R      R:::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM      MMMMMMM RRRRRRR      RRRRRR
```

```
[hadoop@ip-172-30-0-221 ~]$
```

Install RStudio Server

EMR uses Amazon Linux which is based on Centos. Update your master node and install dependencies that will be used by R packages.

```
# Update
sudo yum update
sudo yum install libcurl-devel openssl-devel # used for devtools
```

The installation of RStudio Server is easy. Download the preview version (<https://www.rstudio.com/products/rstudio/download/preview/>) of RStudio and install on the master node.

```
# Install RStudio Server
wget -P /tmp https://s3.amazonaws.com/rstudio-dailybuilds/rstudio-server-rhel-0.99-x86_64.rpm
sudo yum install --nogpgcheck /tmp/rstudio-server-rhel-0.99-x86_64.rpm
```

Create a User

Create a user called `rstudio-user` that will perform the data analysis. Create a user directory for `rstudio-user` on HDFS with the `hadoop fs` command.

```
# Make User
sudo useradd -m rstudio-user
sudo passwd rstudio-user

# Create new directory in hdfs
hadoop fs -mkdir /user/rstudio-user
hadoop fs -chmod 777 /user/rstudio-user
```

Download flights data

The flights (<http://stat-computing.org/dataexpo/2009/the-data.html>) data is a well known data source representing 123 million flights over 22 years. It consumes roughly 12 GiB of storage in uncompressed CSV format in yearly files.

Switch User

For data loading and analysis, make sure you are logged in as regular user.

```
# create directories on hdfs for new user
hadoop fs -mkdir /user/rstudio-user
hadoop fs -chmod 777 /user/rstudio-user

# switch user
su rstudio-user
```

Download data

Run the following script to download data from the web onto your master node. Download the yearly flight data and the airlines lookup table.

```
# Make download directory
mkdir /tmp/flights

# Download flight data by year
for i in {1987..2008}
do
    echo "$(date) $i Download"
    fnam=$i.csv.bz2
    wget -O /tmp/flights/$fnam http://stat-computing.org/dataexpo/2009/$f
nam
    echo "$(date) $i Unzip"
    bunzip2 /tmp/flights/$fnam
done

# Download airline carrier data
wget -O /tmp/airlines.csv http://www.transtats.bts.gov/Download_Lookup.as
p?Lookup=L_UNIQUE_CARRIERS

# Download airports data
wget -O /tmp/airports.csv https://raw.githubusercontent.com/jpatokal/open
flights/master/data/airports.dat
```

Distribute into HDFS

Copy data into HDFS using the `hadoop fs` command.

```
# Copy flight data to HDFS
hadoop fs -mkdir /user/rstudio-user/flights/
hadoop fs -put /tmp/flights /user/rstudio-user/

# Copy airline data to HDFS
hadoop fs -mkdir /user/rstudio-user/airlines/
hadoop fs -put /tmp/airlines.csv /user/rstudio-user/airlines

# Copy airport data to HDFS
hadoop fs -mkdir /user/rstudio-user/airports/
hadoop fs -put /tmp/airports.csv /user/rstudio-user/airports
```

Create Hive tables

Launch Hive from the command line.

```
# Open Hive prompt
hive
```

Create the metadata that will structure the flights table. Load data into the Hive table.


```
# Create metadata for flights
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 into table
LOAD DATA INPATH '/user/rstudio-user/flights' INTO TABLE flights;
```

Create the metadata that will structure the airlines table. Load data into the Hive table.

```
# Create metadata for airlines
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 into table
LOAD DATA INPATH '/user/rstudio-user/airlines' INTO TABLE airlines;
```

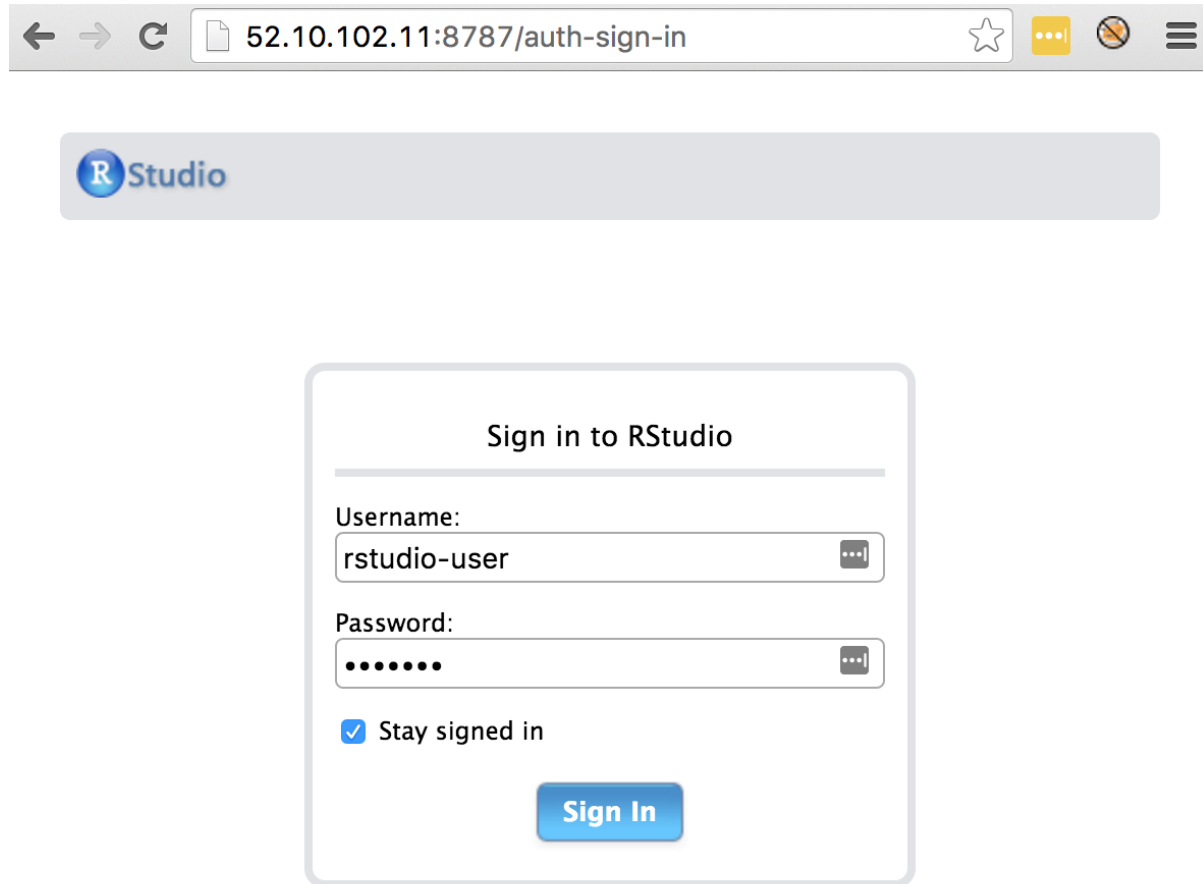
Create the metadata that will structure the airports table. Load data into the Hive table.

```
# Create metadata for airports
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 into table
LOAD DATA INPATH '/user/rstudio-user/airports' INTO TABLE airports;
```

Connect to Spark

Log in to RStudio Server by pointing a browser at your master node IP:8787.



← → ↻ 52.10.102.11:8787/auth-sign-in ☆ ⋮ 🔒 ≡

RStudio

Sign in to RStudio

Username:
rstudio-user

Password:
••••••

☒ Stay signed in

Sign In

Set the environment variable `SPARK_HOME` and then run `spark_connect`. After connecting you will be able to browse the Hive metadata in the RStudio Server Spark pane.

```
# Connect to Spark
library(sparklyr)
library(dplyr)
library(ggplot2)
Sys.setenv(SPARK_HOME="/usr/lib/spark")
config <- spark_config()
sc <- spark_connect(master = "yarn-client", config = config, version =
'1.6.2')
```

Once you are connected, you will see the Spark pane appear along with your hive tables.

The screenshot shows the Spark tab in RStudio. The 'Environment' pane on the left lists 'yarn-client' and 'flights'. The 'flights' table is selected, and its schema is displayed in the main pane. The schema shows columns for year, month, dayofmonth, dayofweek, deptime, crsdeptime, arrtime, crsarrrtime, uniquecarrier, flightnum, and tailnum, with their respective data types and sample values.

```

year : int NA 1987 1987 1987 1987
month : int NA 10 10 10 10
dayofmonth : int NA 14 15 17 18
dayofweek : int NA 3 4 6 7
deptime : int NA 741 729 741 729
crsdeptime : int NA 730 730 730 730
arrtime : int NA 912 903 918 847
crsarrrtime : int NA 849 849 849 849
uniquecarrier : chr "UniqueCarrier" "PS" "..
flightnum : int NA 1451 1451 1451 1451
tailnum : chr "TailNum" "NA" "NA" "NA"

```

You can inspect your tables by clicking on the data icon.

The screenshot shows the RStudio interface with the 'flights' table selected. The 'Environment' pane on the left lists 'flights.R' and 'flights'. The 'flights' table is selected, and its data is displayed in the main pane. The table has 15 columns: year, month, dayofmonth, dayofweek, deptime, crsdeptime, arrtime, crsarrrtime, uniquecarrier, flightnum, tailnum, and act. The data is displayed in a grid format, showing the first 15 rows of the table. The 'uniquecarrier' column contains the value 'PS' for all rows. The 'flightnum' column contains the value '1451' for all rows. The 'tailnum' column contains the value 'NA' for all rows. The 'act' column contains the value 'NA' for all rows. The table is titled 'flights' and is displaying up to 1,000 records.

	year	month	dayofmonth	dayofweek	deptime	crsdeptime	arrtime	crsarrrtime	uniquecarrier	flightnum	tailnum	act
1	NA	NA	NA	NA	NA	NA	NA	NA	UniqueCarrier	NA	TailNum	
2	1987	10	14	3	741	730	912	849	PS	1451	NA	
3	1987	10	15	4	729	730	903	849	PS	1451	NA	
4	1987	10	17	6	741	730	918	849	PS	1451	NA	
5	1987	10	18	7	729	730	847	849	PS	1451	NA	
6	1987	10	19	1	749	730	922	849	PS	1451	NA	
7	1987	10	21	3	728	730	848	849	PS	1451	NA	
8	1987	10	22	4	728	730	852	849	PS	1451	NA	
9	1987	10	23	5	731	730	902	849	PS	1451	NA	
10	1987	10	24	6	744	730	908	849	PS	1451	NA	
11	1987	10	25	7	729	730	851	849	PS	1451	NA	
12	1987	10	26	1	735	730	904	849	PS	1451	NA	
13	1987	10	28	3	741	725	919	855	PS	1451	NA	
14	1987	10	29	4	742	725	906	855	PS	1451	NA	
15	1987	10	31	6	726	725	848	855	PS	1451	NA	

Showing 1 to 15 of 1,000 entries

Data analysis

Is there evidence to suggest that some airline carriers make up time in flight? This analysis predicts time gained in flight by airline carrier.

The screenshot shows the RStudio IDE with the 'flights' table loaded from a Spark cluster. The top pane displays a data table with columns: year, month, dayofmonth, dayofweek, dep_time, crsdeptime, arr_time, crsarrrtime, uniquecarrier, flightnum, tailnum, and actual_time. The bottom-left pane shows the R console with commands to load sparklyr and dplyr, set the SPARK_HOME environment variable, and connect to the Spark cluster using spark_connect. The bottom-right pane shows the 'Connect to Spark' documentation page.

Cache the tables into memory

Use `tbl_cache` to load the flights table into memory. Caching tables will make analysis much faster. Create a dplyr reference to the Spark DataFrame.

```
# Cache flights Hive table into Spark
tbl_cache(sc, 'flights')
flights_tbl <- tbl(sc, 'flights')

# Cache airlines Hive table into Spark
tbl_cache(sc, 'airlines')
airlines_tbl <- tbl(sc, 'airlines')

# Cache airports Hive table into Spark
tbl_cache(sc, 'airports')
airports_tbl <- tbl(sc, 'airports')
```

Create a model data set

Filter the data to contain only the records to be used in the fitted model. Join carrier descriptions for reference. Create a new variable called `gain` which represents the amount of time gained (or lost) in flight.

```
# Filter records and create target variable 'gain'
model_data <- flights_tbl %>%
  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(airlines_tbl, by = c("uniquecarrier" = "code")) %>%
  mutate(gain = depdelay - arrdelay) %>%
  select(year, month, arrdelay, depdelay, distance, uniquecarrier, description, gain)

# Summarize data by 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)
```

Source: query [?? x 4]

Database: spark connection master=yarn-client app=sparklyr local=FALSE

	description	gain	distance	depdelay
	<chr>	<dbl>	<dbl>	<dbl>
1	ATA Airlines d/b/a ATA	-3.3480120	1134.7084	56.06583
2	ExpressJet Airlines Inc. (1)	-3.0326180	519.7125	59.41659
3	Envoy Air	-2.5434415	416.3716	53.12529
4	Northwest Airlines Inc.	-2.2030586	779.2342	48.52828
5	Delta Air Lines Inc.	-1.8248026	868.3997	50.77174
6	AirTran Airways Corporation	-1.4331555	641.8318	54.96702
7	Continental Air Lines Inc.	-0.9617003	1116.6668	57.00553
8	American Airlines Inc.	-0.8860262	1074.4388	55.45045
9	Endeavor Air Inc.	-0.6392733	467.1951	58.47395
10	JetBlue Airways	-0.3262134	1139.0443	54.06156
#	... with more rows			

Train a linear model

Predict time gained or lost in flight as a function of distance, departure delay, and airline carrier.

```
# Partition the data into training and validation sets
model_partition <- model_data %>%
  sdf_partition(train = 0.8, valid = 0.2, seed = 5555)

# Fit a linear model
ml1 <- model_partition$train %>%
  ml_linear_regression(gain ~ distance + depdelay + uniquecarrier)

# Summarize the linear model
summary(ml1)
```

Deviance Residuals: (approximate):

Min	1Q	Median	3Q	Max
-305.422	-5.593	2.699	9.750	147.871

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-1.24342576	0.10248281	-12.1330	< 2.2e-16 ***
distance	0.00326600	0.00001670	195.5709	< 2.2e-16 ***
depdelay	-0.01466233	0.00020337	-72.0977	< 2.2e-16 ***
uniquecarrier_AA	-2.32650517	0.10522524	-22.1098	< 2.2e-16 ***
uniquecarrier_AQ	2.98773637	0.28798507	10.3746	< 2.2e-16 ***
uniquecarrier_AS	0.92054894	0.11298561	8.1475	4.441e-16 ***
uniquecarrier_B6	-1.95784698	0.11728289	-16.6934	< 2.2e-16 ***
uniquecarrier_CO	-2.52618081	0.11006631	-22.9514	< 2.2e-16 ***
uniquecarrier_DH	2.23287189	0.11608798	19.2343	< 2.2e-16 ***
uniquecarrier_DL	-2.68848119	0.10621977	-25.3106	< 2.2e-16 ***
uniquecarrier_EV	1.93484736	0.10724290	18.0417	< 2.2e-16 ***
uniquecarrier_F9	-0.89788137	0.14422281	-6.2257	4.796e-10 ***
uniquecarrier_FL	-1.46706706	0.11085354	-13.2343	< 2.2e-16 ***
uniquecarrier_HA	-0.14506644	0.25031456	-0.5795	0.5622
uniquecarrier_HP	2.09354855	0.12337515	16.9690	< 2.2e-16 ***
uniquecarrier_MQ	-1.88297535	0.10550507	-17.8473	< 2.2e-16 ***
uniquecarrier_NW	-2.79538927	0.10752182	-25.9983	< 2.2e-16 ***
uniquecarrier_OH	0.83520117	0.11032997	7.5700	3.730e-14 ***
uniquecarrier_OO	0.61993842	0.10679884	5.8047	6.447e-09 ***
uniquecarrier_TZ	-4.99830389	0.15912629	-31.4109	< 2.2e-16 ***
uniquecarrier_UA	-0.68294396	0.10638099	-6.4198	1.365e-10 ***
uniquecarrier_US	-0.61589284	0.10669583	-5.7724	7.815e-09 ***
uniquecarrier_WN	3.86386059	0.10362275	37.2878	< 2.2e-16 ***
uniquecarrier_XE	-2.59658123	0.10775736	-24.0966	< 2.2e-16 ***
uniquecarrier_YV	3.11113140	0.11659679	26.6828	< 2.2e-16 ***

NA

R-Squared: 0.02385

Root Mean Squared Error: 17.74

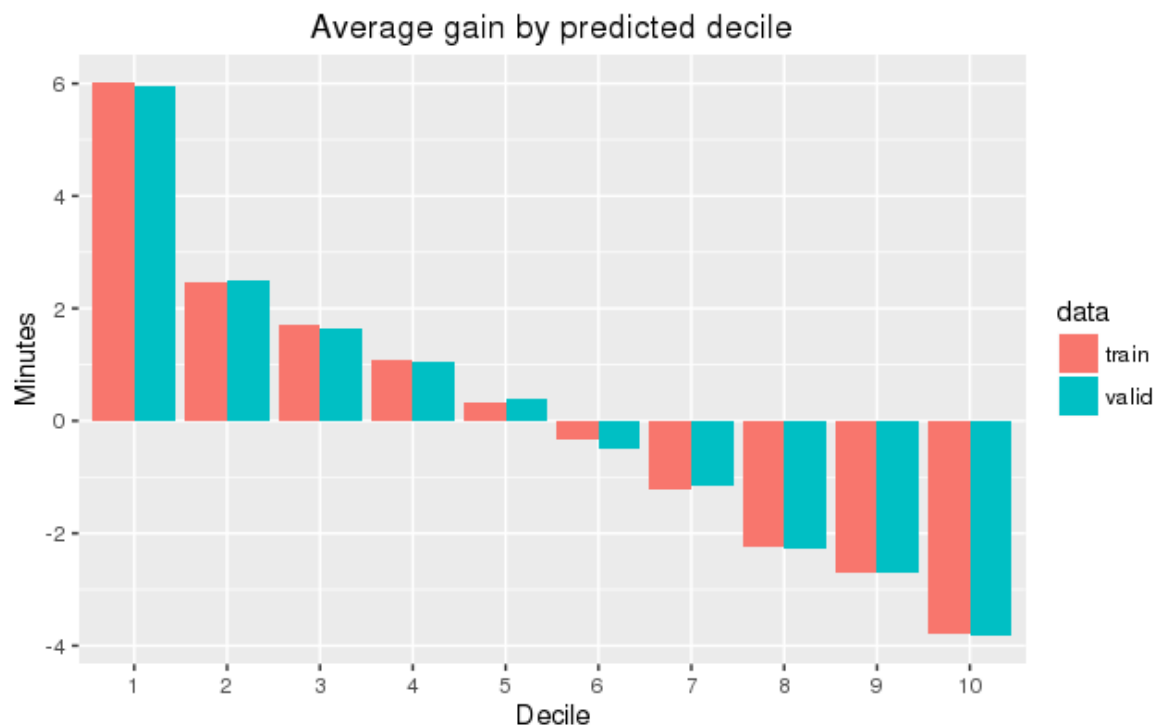
Assess model performance

Compare the model performance using the validation data.

```
# Calculate average gains by predicted decile
model_deciles <- lapply(model_partition, function(x) {
  sdf_predict(mll, x) %>%
    mutate(decile = ntile(desc(prediction), 10)) %>%
    group_by(decile) %>%
    summarize(gain = mean(gain)) %>%
    select(decile, gain) %>%
    collect()
})

# Create a summary dataset for plotting
deciles <- rbind(
  data.frame(data = 'train', model_deciles$train),
  data.frame(data = 'valid', model_deciles$valid),
  make.row.names = FALSE
)

# Plot average gains by predicted decile
deciles %>%
  ggplot(aes(factor(decile), gain, fill = data)) +
  geom_bar(stat = 'identity', position = 'dodge') +
  labs(title = 'Average gain by predicted decile', x = 'Decile', y = 'Minutes')
```

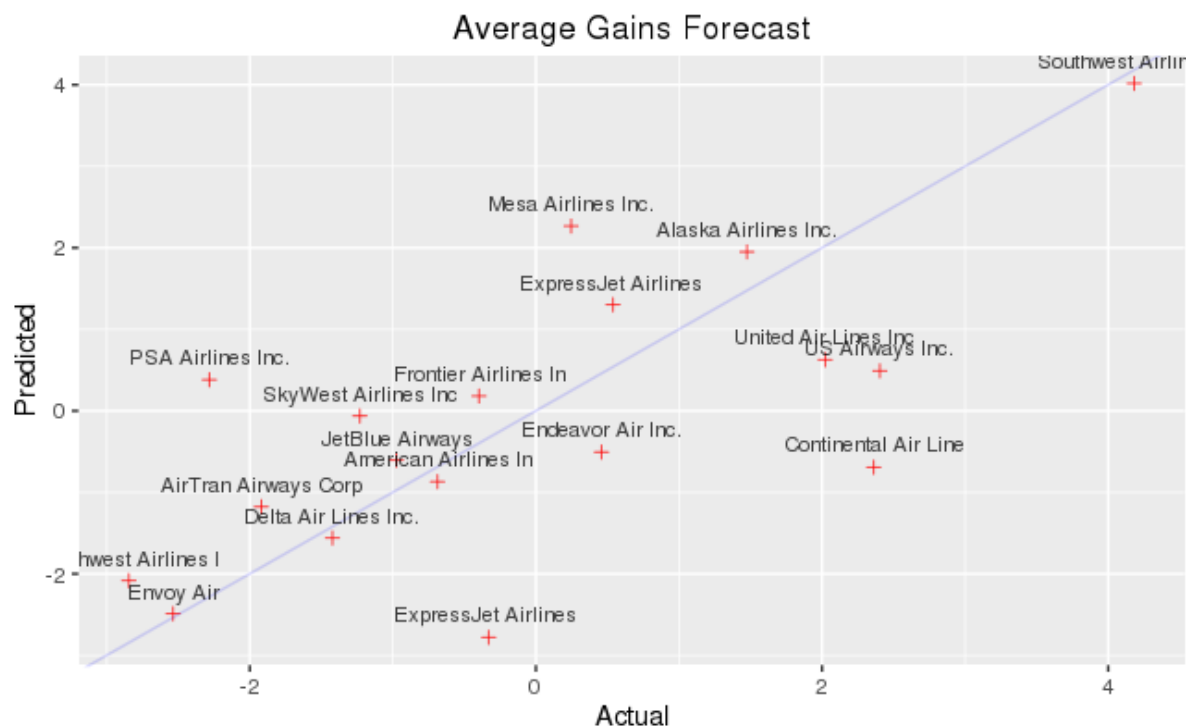
Visualize predictions

Compare actual gains to predicted gains for an out of time sample.

```
# Select data from an out of time sample
data_2008 <- flights_tbl %>%
  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(airlines_tbl, by = c("uniquecarrier" = "code")) %>%
  mutate(gain = depdelay - arrdelay) %>%
  select(year, month, arrdelay, depdelay, distance, uniquecarrier, description, gain, origin, dest)

# Summarize data by carrier
carrier <- sdf_predict(ml1, data_2008) %>%
  group_by(description) %>%
  summarize(gain = mean(gain), prediction = mean(prediction), freq = n()) %>%
  filter(freq > 10000) %>%
  collect

# Plot actual gains and predicted gains by airline carrier
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='Average Gains Forecast', x = 'Actual', y = 'Predicted')
```



Some carriers make up more time than others in flight, but the differences are relatively small. The average time gains between the best and worst airlines is only six minutes. The best predictor of time gained is not carrier but flight distance. The biggest gains were associated with the longest flights.

Share Insights

This simple linear model contains a wealth of detailed information about carriers, distances traveled, and flight delays. These detailed insights can be conveyed to a non-technical audience via an interactive flexdashboard (<http://rmarkdown.rstudio.com/flexdashboard/index.html>).

Build dashboard

Aggregate the scored data by origin, destination, and airline. Save the aggregated data.

```
# Summarize by origin, destination, and carrier
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)
  )

# Collect and save objects
pred_data <- collect(summary_2008)
airports <- collect(select(airports_tbl, name, faa, lat, lon))
ml1_summary <- capture.output(summary(ml1))
save(pred_data, airports, ml1_summary, file = 'flights_pred_2008.RData')
```

Publish dashboard

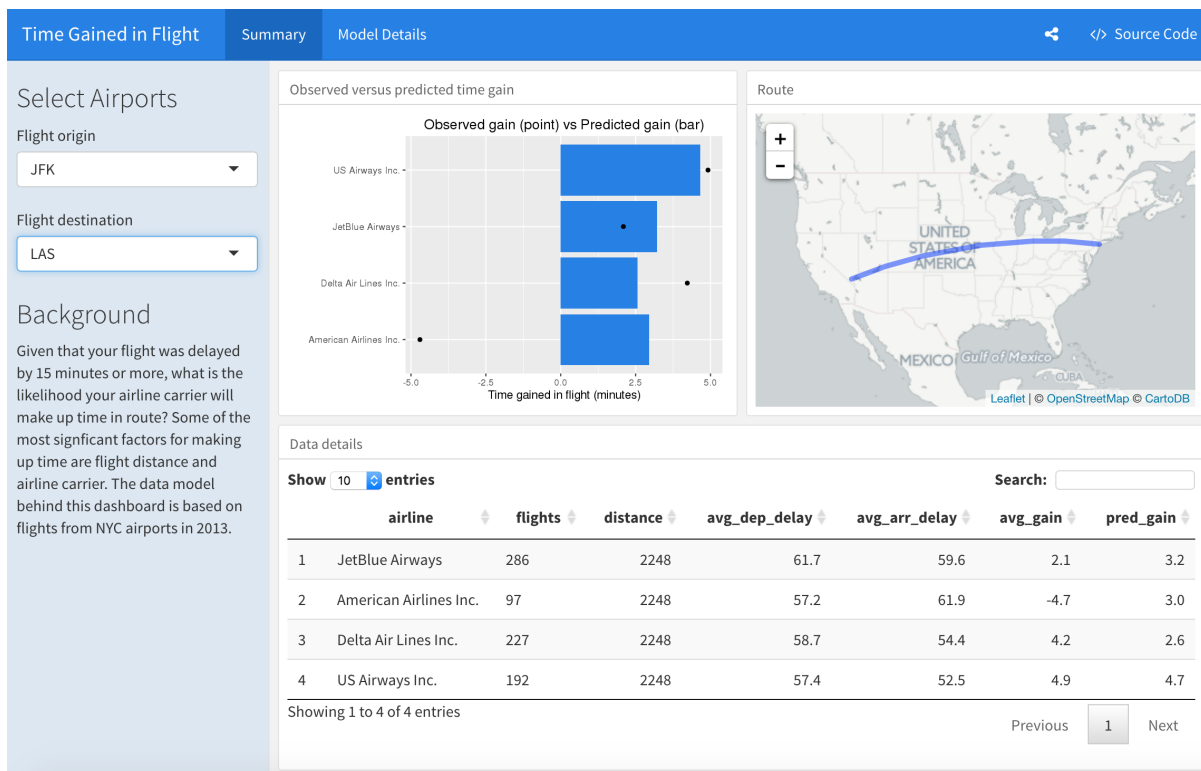
Use the saved data to build an R Markdown flexdashboard

(<http://rmarkdown.rstudio.com/flexdashboard/index.html>). Publish the flexdashboard to Shiny Server

(<https://www.rstudio.com/products/shiny-server-pro/>), Shinyapps.io

(<https://www.rstudio.com/products/shinyapps/>) or RStudio Connect

(<https://www.rstudio.com/products/connect/>).



(<https://beta.rstudioconnect.com/content/1439/>)

sparklyr is an RStudio (<https://www.rstudio.com>) project. © 2016 RStudio, Inc.

