Docker and H2O to enter leaderboard in Kaggle competitions

We're always fascinated to learn about what Data Scientists doing under the hood of they're methodically procedures or their models when we view their notebook on Kaggle.

Today I'm sharing with you a story. We will start from a real Kaggle Competition, Instacart Market Basket Analysis (https://www.kaggle.com/c/instacart-market-basket-analysis), and from our passionate about data and will try to speak with you about R, H2O, Docker and XGBoost models.

In this article, we explore how self-start in data science with little resources. Naturally, I will give you some advice for aspiring data scientists.

H2O

H2O is an in-memory platform for distributed, scalable machine learning. H2O uses familiar interfaces like R, Python, Scala, Java, JSON and the Flow notebook/web interface, and works seamlessly with big data technologies like Hadoop and Spark. It provides implementations of many popular algorithms such as GBM, Random Forest, Deep Neural Networks, Word2Vec and Stacked Ensembles.

More details on https://www.h2o.ai

Instacart Market Basket Analysis

Which products will an Instacart consumer purchase again?

Instacart is a grocery ordering and delivery app that aims to make it easy to fill your refrigerator and pantry with your favorites and staples. After selecting products through the Instacart app, personal shoppers review your order and do the in-store shopping and delivery for you. The data science plays a big part in providing recommendations and other models and data to improve the shopping experience in Instacart app.

In Kaggle competition, the challenge is to use their anonymized data on customer orders to predict which previously purchased products will be in a user's next order.

The dataset for this competition is a relational set of files describing customers' orders over time. The dataset contains a sample of over 3 million grocery orders from more than 200,000 Instacart users. For each user, there are between 4 and 100 orders, with the sequence of products purchased in each order. Instacart also provide the week and hour of day the order was placed, and a relative measure of time between orders. Each entity (customer, product, order, aisle, etc.) has an associated unique id. Most of the files and variable names should be self-explanatory.

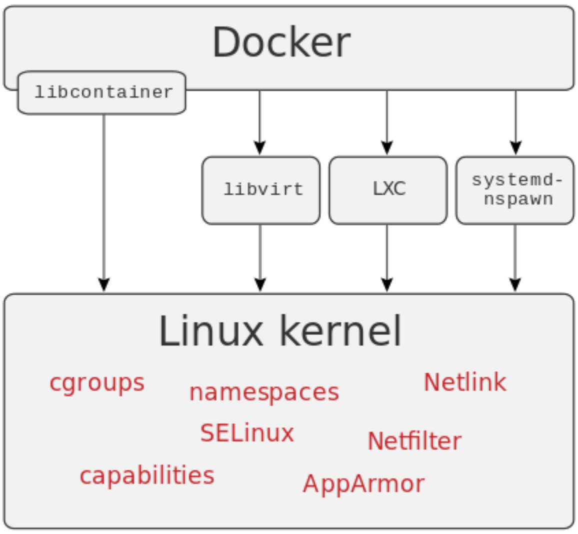
The data are composed by this list of files:
- aisles.csv
- products.csv
- departments.csv
- order_products__*.csv: these files specify which products were purchased in each order; their prior file contains previous order contents for all customers; reordered indicates that the customer has a previous order that contains the product
- orders.csv: this file tells to which set (prior, train, test) an order belongs. We are predicting reordered items only for the test set orders; order_dow is the day of week.

For more information go to Kaggle (https://www.kaggle.com/c/instacart-market-basket-analysis/data) or Instacart blog (https://tech.instacart.com/3-million-instacart-orders-open-sourced-d40d29ead6f2).

Download all the files from your favorite folder and set up the environment.

Using H2O with Docker: setup the environment

From Wikipedia (https://en.wikipedia.org/wiki/Docker_(software)), Docker is a software technology providing operating-system-level virtualization called containers.
Docker uses the resource isolation features of the Linux kernel and a union-capable file system to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.



According to a Linux.com article, Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. This helps enable flexibility and portability on where the application can run, whether on premises, public cloud, private cloud, bare metal, etc.

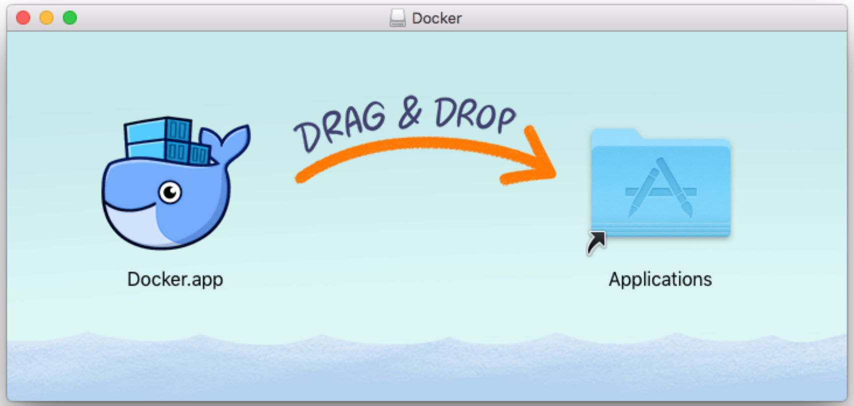
Because Docker containers are so lightweight, a single server or virtual machine can run several containers simultaneously.

The steps to setup the environment are the follows:
- Installing Docker on Mac or Linux OS
- Download the Dockerfile from H2O repository
- Building a Docker image
- Running the Docker container
- Launching H2O and accessing from the web browser or R

To run Docker our machine have to respect some prerequisites:
- Linux kernel version 3.8+ or Mac OS X 10.6+
- Using User directory (not root)
- For Ubuntu, Trusty 14.04 (LTS) or Xenial 16.04 (LTS)

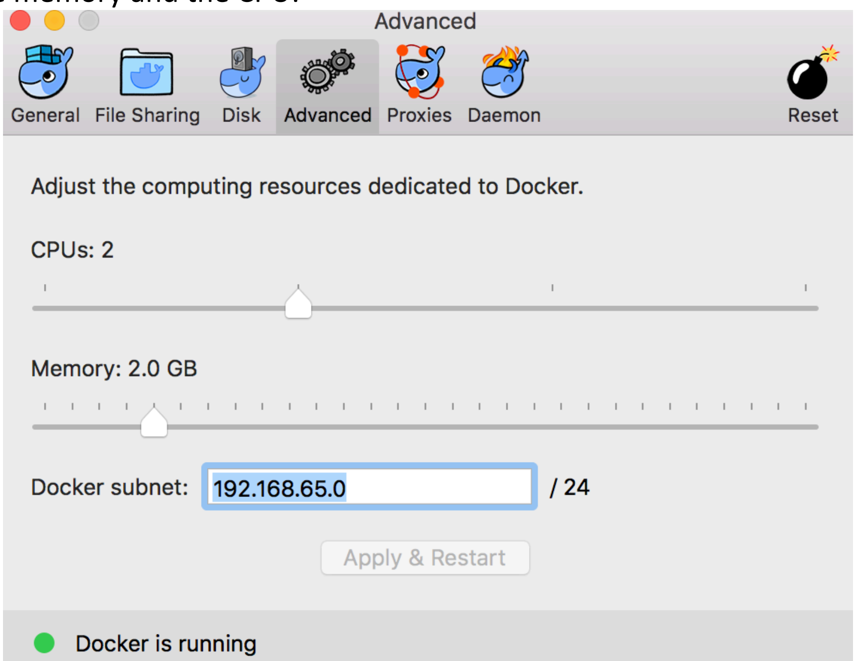
Depending on your OS, select the appropriate installation method:
- Mac Installation (https://docs.docker.com/installation/mac/#installation)
- Linux (Ubuntu) Installation (https://docs.docker.com/installation/ubuntulinux/)

To get Docker for Mac, you have to download the stable (recommended) version from https://docs.docker.com/docker-for-mac/install/.
After download, double-click Docker.dmg to open the installer and drag Moby the whale to the Applications folder.



Double-click Docker.app in the Applications folder to start Docker.
You have to authorize Docker.app with your system password after you launch it. The whale in the top status bar indicates that Docker is running, and accessible from a terminal.

On Mac, access to advanced preferences of your Docker.app (from whale in the top status bar) and setup up the memory and the CPU:



Recommended configuration:
-CPUs: 8
- Memory: 8g

On Ubuntu, install the Linux-image-extra-* packages, which allow Docker to use the aufs storage drivers.

```
sudo apt-get update
sudo apt-get install \
    linux-image-extra-$(uname -r) \
    linux-image-extra-virtual
```

You can install Docker CE on Ubuntu in different ways, depending on your needs. The recommended approach is to set up Docker from Docker's repositories.

```
sudo apt-get update
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

Now update the apt package index and install docker

```
sudo apt-get update
sudo apt-get install docker-ce
```

From terminal, you can check your installation (**in Linux/Ubuntu run each command as sudo**):

```
docker --version
Docker version 17.12.0-ce, build c97c6d6

docker-compose --version
docker-compose version 1.18.0, build 8dd22a9

docker-machine --version
docker-machine version 0.13.0, build 9ba6da9
```

Using H2O with Docker: create the H2O container

Note: if the following commands don't work, prepend them with sudo

First of all, we have to create a folder on your machine to host your Dockerfile

```
cd /opt
mkdir Docker_Image
cd Docker_Image
mkdir h2o_image
cd h2o_image
```

Now download the latest version of Dockerfile from the repository on Git Hub.
For example, on Mac:

```
curl https://raw.githubusercontent.com/h2oai/h2o-3/master/Dockerfile -o
Dockerfile
```

On Ubuntu:

```
wget https://raw.githubusercontent.com/h2oai/h2o-3/master/Dockerfile
```

The Dockerfile:

- Obtains and updates the base image (Ubuntu 14.04)
- Installs Java 7
- Obtains and downloads the H2O build from H2O's S3 repository
- Important: exposes ports **54321** and **54322** in preparation for launching H2O on those ports

From h2o_image folder, now we build the Docker image from Dockerfile:

```
docker build -t "h2oai/3:v5" .
```

Because it assembles all the necessary parts for the image, this process can take a few minutes. At the end of the assemble step, we can run Docker build image.

```
docker run -ti -p 54321:54321 -m 16g h2oai/3:v5 /bin/bash
```

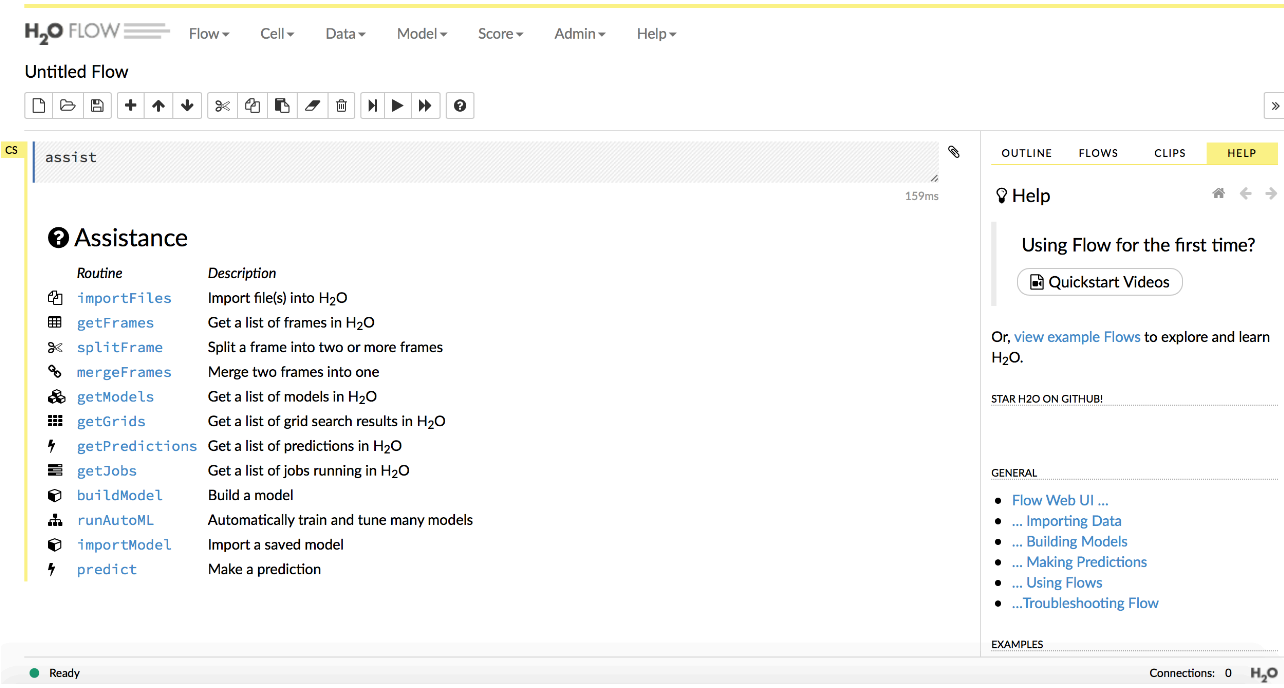
Note for this run:

- The parameter –m specify the memory reserved for the container (in our case 16g of RAM)
- The parameter –p map the port 54321 of the your machine to the port 54321 of the container

Navigate to the /opt directory and launch H2O.
Remember to change the value of -Xmx to the amount of memory you want to allocate to the H2O instance. By default, H2O launches on port 54321.
cd /opt

```
java -Xmx10g -jar h2o.jar
```

With port mapping, you can access to H2O from web browser of your host machine using http://localhost:54321/flow/index.html



Useful Docker commands

List of all Docker containers (run or exited):

```
docker ps –a
```

```
CONTAINER ID    IMAGE         COMMAND       CREATED        STATUS                    PORTS                                  NAMES
dd80542a87fe    h2oai/3:v5    "/bin/bash"   17 hours ago   Up 17 hours               54322/tcp, 0.0.0.0:4444->54321/tcp     gracious_bartik
2674a8078c23    h2oai/3:v5    "/bin/bash"   18 hours ago   Exited (130) 17 hours ago                                        serene_knuth
cfd3e494bcc3    h2oai/3:v5    "/bin/bash"   18 hours ago   Exited (0) 18 hours ago                                          sleepy_ritchie
1f13b9e63ac1    h2oai/3:v5    "/bin/bash"   18 hours ago   Exited (0) 18 hours ago                                          quirky_poitras
290f7254a899    h2oai/3:v5    "/bin/bash"   20 hours ago   Exited (130) 18 hours ago                                        dazzling_mirzakhani
0252df526d90    h2oai/3:v5    "/bin/bash"   20 hours ago   Exited (127) 20 hours ago                                        zealous_rosalind
b9d90cc9c7ef    h2oai/3:v5    "/bin/bash"   20 hours ago   Exited (130) 20 hours ago                                        hopeful_edison
```

Attach to a Docker container
```
docker attach <CONTAINER-ID>
```

Manipulating Data with dplyr

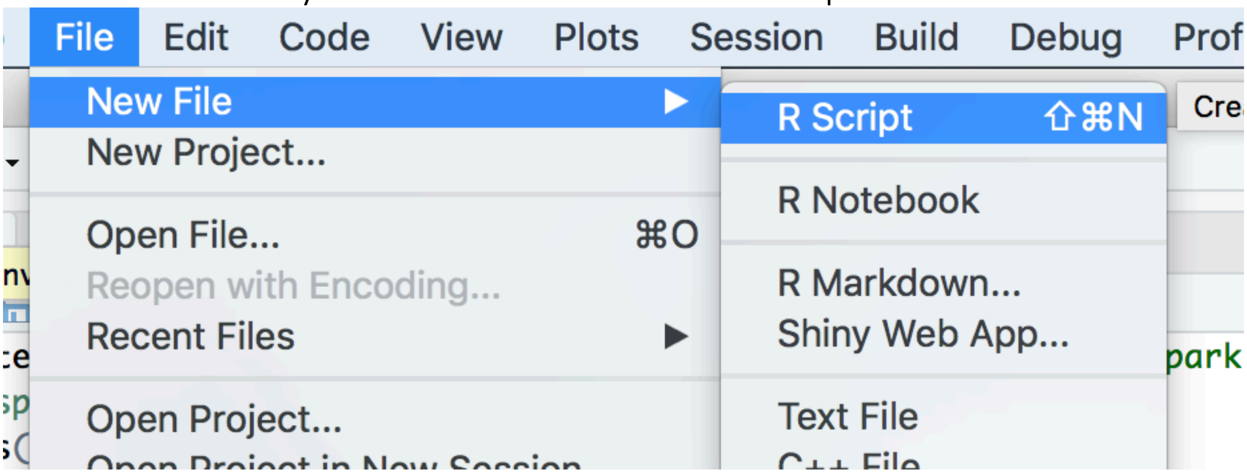
dplyr is an R package for working with structured data both in and outside of R.
With dplyr you can:
- Select, filter, and aggregate data
- Use window functions
- Perform joins on DataFrames

Verbs are dplyr commands for manipulating data. The basic dplyr verbs are:
- select: perform a SELECT
- filter: apply filtering condition (like a WHERE in SQL)
- arrange: sort the data
- summarise: aggregate the data (e.g. mean, sum, min, sd, etc.)
- mutate: add a column and apply some operators like +, *, log, etc.

R + H2O Kernel

Now, I will present one basic solution to predict which previously purchased products will be in a user's next order. Full code are available here: https://github.com/mauropelucchi/machine-learning-course/blob/master/xgboost/h2o_instacart.r

Run RStudio or R from your host machine and create a new script.



Load the mandatory library
```
library(dplyr)
library(h2o)
library(data.table)
```

and check the version of the package ('3.16.0.2' in my case)
```
packageVersion("h2o")
```

Now init the connection to H2O container (set up max_mem_size and nthreads based on your container)

```
h2o.init(ip = "localhost", port = 54321, nthreads = 15, max_mem_size = "8g",
strict_version_check = FALSE)
```

The follow script load the csv files and run some join and merge operations to obtain a denormalized table with all features and categorical variables.

```
# Load datasets
ais <- fread("/Users/mauropelucchi/Desktop/Instacart/aisles.csv", key = "aisle_id")
dept <- fread("/Users/mauropelucchi/Desktop/Instacart/departments.csv", key = "department_id")
prod <- fread("/Users/mauropelucchi/Desktop/Instacart/products.csv", key =
c("product_id","aisle_id", "department_id"))
opp <- fread("/Users/mauropelucchi/Desktop/Instacart/order_products__prior.csv")
opt <- fread("/Users/mauropelucchi/Desktop/Instacart/order_products__train.csv")
ord <- fread("/Users/mauropelucchi/Desktop/Instacart/orders.csv")


# Get product department and aisle names
prod <- merge(prod, ais, by="aisle_id", all.x=TRUE, sort=FALSE)
prod <- merge(prod, dept, by="department_id", all.x=TRUE, sort=FALSE)

# For the prior orders get the associated product, aisle, departments, and users
opp <- merge(opp, prod, by="product_id", all.x=TRUE, sort=FALSE)
opp <- merge(opp, ord, by="order_id", all.x=TRUE, sort=FALSE)
```

Now it's time to add some new features like
- last order id
- number of purchases by user
- average hour of purchases
- number of purchaes by user and product
- reorderd rate (calculate among user and product)
- average number of products in a basket (by user)
- last purchased order
- first purchased oder
- average days between order by user and product
(and over...).
For this scope we use the verbs from dplyr library (filer, join, ...).
Remeber:
- left_join: return all rows from x, and all columns from x and y. If there are multiple matches between x and y, all combination of the matches are returned;
- inner_join: return all rows from x where there are matching values in y;
- anti_join: return all rows from x where there are not matching values in y, keeping just columns from x (a filtering join).

(for more information read this http://stat545.com/bit001_dplyr-cheatsheet.html)

```
ord_max <- opp %>%
  group_by(user_id) %>%
  summarize(order_max = max(order_number), purch_count = n())
head(ord_max)

opp <- left_join(opp, ord_max, by="user_id")
opp <- opp %>% mutate(orders_ago=order_max - order_number + 1)
select(opp, order_number, user_id, order_max, orders_ago, purch_count)

# Create a few simple features

user_prod_list <- opp %>%
```

```
  group_by(user_id, product_id) %>%
  summarize(last_order_number = max(order_number), purch_count = n(), avg_hour =
mean(order_hour_of_day))

# Compure reordered rate
user_prod_list_reordered <- opp %>%
  filter(reordered == 1) %>%
  group_by(user_id, product_id) %>%
  summarize(reordered_count = n())
user_prod_list <- left_join(user_prod_list, user_prod_list_reordered, by=c("user_id",
"product_id"))
user_prod_list <- user_prod_list %>% mutate(reorder_rate=reordered_count/purch_count)
user_prod_list <- user_prod_list %>% mutate(reorder_rate=ifelse(is.na(reorder_rate), 0,
reorder_rate))
user_prod_list <- user_prod_list %>% mutate(reordered_count=ifelse(is.na(reordered_count), 0,
reordered_count))
head(user_prod_list)

user_summ <- opp %>%
  group_by(user_id) %>%
  summarize(user_total_products_ordered_hist = n(),
            uniq_prod = n_distinct(product_name),
            uniq_aisle = n_distinct(aisle),
            uniq_dept = n_distinct(department),
            prior_orders = max(order_number),
            avg_hour = mean(order_hour_of_day),
            average_days_between_orders = mean(days_since_prior_order),
            total_order = n_distinct(order_number),
            average_basket = n() / n_distinct(order_number)
  )
head(user_summ)
user_prior_prod_cnt <- opp %>%
  group_by(user_id, product_id) %>%
  summarize(prior_prod_cnt = n(),
            last_purchased_orders_ago = min(orders_ago),
            first_purchased_orders_ago = max(orders_ago),
            average_days_between_ord_prods = mean(days_since_prior_order)
  )
head(user_prior_prod_cnt)
```

Finally, it's time to create train frame and test frame.

```
# Merge datasets to create training frame
opt_user <- left_join(filter(opt, reordered==1), ord, by="order_id")
dt_expanded <- left_join(user_prod_list, opt_user, by=c("user_id", "product_id"))
dt_expanded <- dt_expanded %>% mutate(curr_prod_purchased=ifelse(!is.na(order_id), 1, 0))
#head(dt_expanded)
train <- left_join(dt_expanded, user_summ, by="user_id")
train <- left_join(train, user_prior_prod_cnt, by=c("user_id", "product_id"))
varnames <- setdiff(colnames(train), c("user_id","order_id","curr_prod_purchased"))
head(train)

# Create the test frame
test_orders <- filter(ord, eval_set=="test")
dt_expanded_test <- inner_join(user_prod_list, test_orders, by=c("user_id"))
dt_expanded_test <- dt_expanded_test %>% mutate(curr_prod_purchased=sample(c(0,1), n(),
replace=TRUE))
#head(dt_expanded_test)
test <- inner_join(dt_expanded_test, user_summ, by="user_id")
test <- inner_join(test, user_prior_prod_cnt, by=c("user_id", "product_id"))
head(test)

# Check target 1/0
test %>% ungroup %>% distinct(curr_prod_purchased)
train %>% ungroup %>% distinct(curr_prod_purchased)

# Sample users for the validation set
set.seed(2222)
unique_user_id <- select(ord_max, user_id)
head(unique_user_id)
```

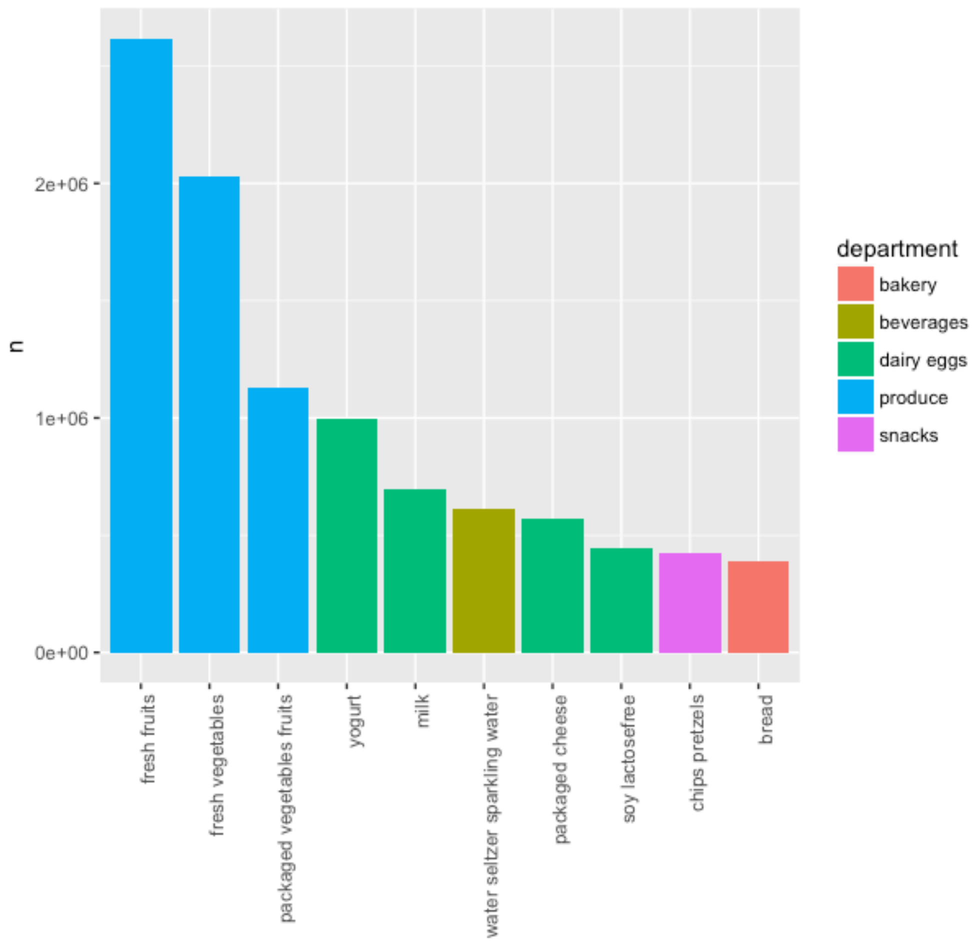
```
val_users <- sample_n(unique_user_id, size=10000, replace=FALSE)
head(val_users)

# Ungroup and convert to factor
train <- train %>% ungroup() %>% mutate(curr_prod_purchased=as.factor(curr_prod_purchased))
test <- test %>% ungroup() %>% mutate(curr_prod_purchased=as.factor(curr_prod_purchased))
test %>% distinct(curr_prod_purchased)
train %>% distinct(curr_prod_purchased)
```

Some preliminary exploratory analysis:

```
# Some exploratory analysis
library(ggplot2)
tmp <- filter(opp, reordered == 1) %>%
  group_by(aisle,department) %>%
  tally(sort=TRUE) %>%
  mutate(perc = round(100*n/nrow(opp),2)) %>%
  ungroup() %>%
  top_n(10,n)
tmp %>%
  ggplot(aes(x=reorder(aisle, -n), y=n, fill=department)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())
```

The top 10 aisles that represent the 45% of sales:

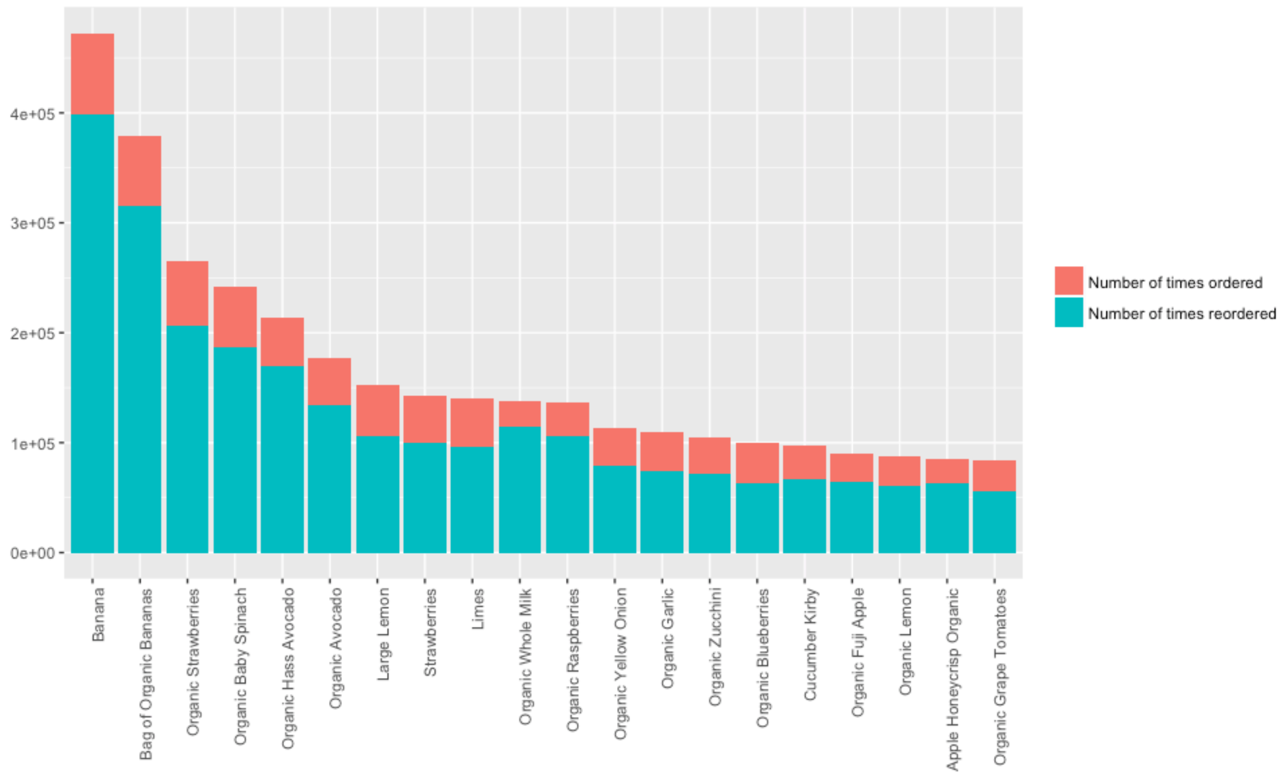


Products vs number of times ordered/reordered:

```
# Products vs number of times ordered/reordered
t <- filter(opp) %>% select(product_id, product_name) %>% group_by(product_id, product_name)
%>% summarize(ncount=n()) %>% ungroup()
r <- filter(opp, reordered == 1) %>% select(product_id, product_name) %>% group_by(product_id,
product_name) %>% summarize(rcount=n()) %>% ungroup()
```

```
t <- left_join(t, r, by="product_id") %>% top_n(20,ncount)
t

t %>%
  ggplot() +
  geom_bar(aes(x=reorder(product_name.x , -ncount), y=ncount, fill="Number of times ordered"
), stat="identity") +
  geom_bar(aes(x=reorder(product_name.x , -ncount), y=rcount, fill="Number of times reordered"
), stat="identity") +
  guides(fill=guide_legend(title=element_blank())) +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank(),
axis.title.y=element_blank())#
```



Finally, we train the XGBoost model (the target is binary so one approach is use logloss as stopping metric).
XGBoost is short for “Extreme Gradient Boosting”, where the term “Gradient Boosting” is proposed in the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman. (https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf)
Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models (decision trees).

```
# Add data to H20
colnames(train)
train_tpl <- anti_join(train, val_users, by="user_id")  %>% select(c("curr_prod_purchased",
"user_id", "reordered_count", "product_id", "last_order_number", "purch_count", "avg_hour.x",
                                                                       "order_dow",
"add_to_cart_order", "reordered", "user_total_products_ordered_hist", "uniq_prod",
"uniq_aisle", "uniq_dept",
                                                                       "prior_orders",
"avg_hour.y", "average_days_between_orders", "total_order", "average_basket",
"prior_prod_cnt",
```

```
"last_purchased_orders_ago", "first_purchased_orders_ago", "average_days_between_ord_prods"))
val_tpl <- inner_join(train, val_users, by="user_id")  %>% select(c("curr_prod_purchased",
"user_id", "reordered_count", "product_id", "last_order_number", "purch_count", "avg_hour.x",
                                                                    "order_dow",
"add_to_cart_order", "reordered", "user_total_products_ordered_hist", "uniq_prod",
"uniq_aisle", "uniq_dept",
                                                                    "prior_orders",
"avg_hour.y", "average_days_between_orders", "total_order", "average_basket",
"prior_prod_cnt",

"last_purchased_orders_ago", "first_purchased_orders_ago", "average_days_between_ord_prods"))

train.hex <- as.h2o(train_tpl, destination_frame = "train.hex")
val.hex <- as.h2o(val_tpl, destination_frame = "val.hex")

# Free up some memory
rm(train, opp, opt, ord, prod, dept, ais, user_prod_list, user_summ);gc()
rm(train_tpl, val_tpl);gc();

# Train xgboost model
xgb <- h2o.xgboost(x = c("user_id", "reordered_count", "product_id", "last_order_number",
"purch_count", "avg_hour.x",
                         "order_dow", "add_to_cart_order", "reordered",
"user_total_products_ordered_hist", "uniq_prod", "uniq_aisle", "uniq_dept",
                         "prior_orders", "avg_hour.y", "average_days_between_orders",
"total_order", "average_basket", "prior_prod_cnt",
                         "last_purchased_orders_ago", "first_purchased_orders_ago",
"average_days_between_ord_prods")
                   ,y = "curr_prod_purchased"
                   ,training_frame = train.hex
                   ,validation_frame = val.hex
                   ,model_id = "xgb_model_1"
                   ,stopping_rounds = 3
                   ,stopping_metric = "logloss"
                   ,distribution = "bernoulli"
                   ,score_tree_interval = 1
                   ,learn_rate=0.1
                   ,ntrees=20
                   ,subsample = 0.75
                   ,colsample_bytree = 0.75
                   ,tree_method = "hist"
                   ,grow_policy = "lossguide"
                   ,booster = "gbtree"
                   ,gamma = 0.0
)
```

Some output from the model:

```
INFO: Variable Importances:
INFO:                       Variable Relative Importance Scaled Importance Percentage
INFO:      last_purchased_orders_ago           95.000000          1.000000   0.325342
INFO:                reordered_count           79.000000          0.831579   0.270548
INFO:     first_purchased_orders_ago           20.000000          0.210526   0.068493
INFO:                    purch_count           20.000000          0.210526   0.068493
INFO:                   prior_orders           18.000000          0.189474   0.061644
INFO:                        user_id           16.000000          0.168421   0.054795
INFO:                      order_dow           14.000000          0.147368   0.047945
INFO:                    total_order            6.000000          0.063158   0.020548
INFO:              add_to_cart_order            5.000000          0.052632   0.017123
INFO:                     avg_hour.x            4.000000          0.042105   0.013699
INFO:                      uniq_prod            4.000000          0.042105   0.013699
INFO:              last_order_number            4.000000          0.042105   0.013699
INFO:                     uniq_aisle            3.000000          0.031579   0.010274
INFO: average_days_between_ord_prods            3.000000          0.031579   0.010274
INFO:                 average_basket            1.000000          0.010526   0.003425
```

```
Description: Metrics reported on training frame
model id: xgb_model_1
frame id: train.hex
MSE: 0.0044923713
RMSE: 0.067025155
AUC: 1.0
logloss: 0.069314316
```

```
# Make predictions
test.hex <- as.h2o(test, destination_frame = "test.hex")
predictions <- as.data.table(h2o.predict(xgb, test.hex))

predictions <- data.table(order_id=test$order_id, product_id=test$product_id,
testPreds=predictions$predict, p0=predictions$p0, p1=predictions$p1)
filter(predictions, testPreds==1)
testPreds <- predictions[,.(products=paste0(product_id[p0>0.23], collapse=" ")), by=order_id]
set(testPreds, which(testPreds[["products"]]==""), "products", "None")
# Create submission file
fwrite(testPreds, "/Users/mauropelucchi/Desktop/Instacart/submission.csv")
```

What to do now? Spark!

R is great for statistical computing, graphics, and small-scale data preparation. And H2O is an amazing distributed machine learning platform written in R designed for scale and speed. Spark is great for super-fast data processing at mega scale.

Apache Spark (https://spark.apache.org) is a fast and general engine for big data processing with built-in modules for streaming, SQL, machine learning, and graph processing.

Useful packages:
- sparklyr (http://spark.rstudio.com)
- rsparkling (https://github.com/h2oai/rsparkling): the rsparkling R package is an extension package for sparklyr that creates an R front-end for the Sparkling Water Spark package from H2O;
- SparkR (https://spark.apache.org/docs/latest/sparkr.html) is an R package that provides a lightweight frontend to use Apache Spark from R. SparkR also supports distributed machine learning using Spark MLlib;
- Sparkling Water (https://www.h2o.ai/sparkling-water/) integrates H2O's fast scalable machine learning engine with Spark.