

# Analogsea: Using R for Big Data Analytics

## Introduction

Analogsea is a community project created by and for statisticians from the R community. The package's goal is to provide analytical capacity for Lesser Developed Countries and users that don't have access to large computational power. This package provides an interface to DigitalOcean, a cloud service such as AWS, Linode and others, for R users, but at a much lower cost and providing more gradual scalability options.

Even when DigitalOcean provides an API, computing across clusters composed by virtual servers is not straightforward, since vanilla Ubuntu servers ('droplets' in DO jargon) require additional configurations to install R itself and numerical libraries such as OpenBLAS.

Cutting edge research methods are really computationally intensive. As an example, on my laptop, using all CPU cores, a bayesian model can take up to 45 minutes to run one set of models, and with all the different model specifications and robustness checks, it takes hours to run the complete analysis.

By scaling computing hardware, this can be reduced to less than five minutes, but not all universities and research centers in Latin America and Asia Pacific have access to servers, and analogsea provides the option to rent a server and access to large computational resources to fit large scale models even on a tight budget.

However, because R packages require additional configuration, and the tradeoff between setup times and Kubernetes configuration versus just doing it all manually is not favorable. This is why analogsea also implements readily available server side solutions to use R out-of-the-box.

## Fitting models in the cloud

Here I'll provide an elementary example that benefits from distributed computing, by fitting an elemental regression for a large taxi fares dataset.

The model to fit shall be of the form

$$\text{Total Amount}_i = \beta_0 + \beta_1 \text{Trip Distance}_i + \varepsilon_i$$

The error is of the form  $\varepsilon \sim N(0, \sigma^2)$  and regardless of the simplicity of this OLS model, it can be fitted on most common laptops since it's a ~14 million rows dataset for a single month, and therefore fitting a regression for a complete year would be unfeasible.

You can have a look at the NYC Taxi dataset, which is stored in DigitalOcean Spaces with S3 compatible features. You can copy and explore the data locally like this.

```
space <- arrow::S3FileSystem$create(
  anonymous = TRUE,
  scheme = "https",
  endpoint_override = "sfo3.digitaloceanspaces.com"
)
# just 2009/01 for a quick exploration
try(dir.create("~/nyc-taxi/2009/01", recursive = T))
arrow::copy_files(space$path("nyc-taxi/2009/01"), "~/nyc-taxi/2009/01")
```

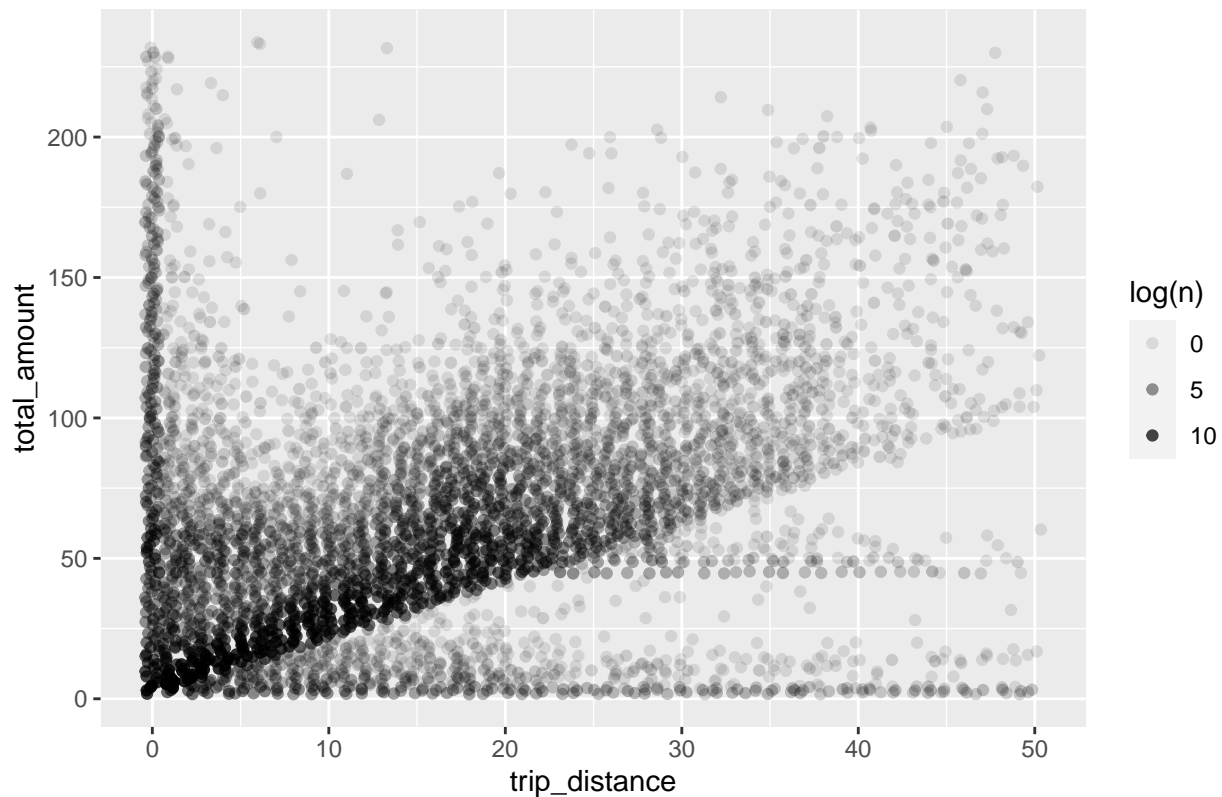
One interesting exercise is to explore how trip distance and total amount are related. You can fit a regression to do that! But you should explore the data first, and a good way to do so is to visualize how the observations are distributed.

```
library(arrow)
d <- open_dataset(
  "~/nyc-taxi",
  partitioning = c("year", "month")
)
dreg <- d %>%
  select(trip_distance, total_amount) %>%
  collect()
dreg
# # A tibble: 14,092,413 x 2
#   trip_distance total_amount
#   <dbl>         <dbl>
# 1      2.63         9.40
# 2      4.55        14.6
# 3     10.4        28.4
# 4       5         18.5
# 5     0.400        3.70
# # ... with 14,092,408 more rows
```

As this is a ~14 million rows for a single month, it will be hard to plot without at least some aggregation. There are repeated rows so you can count to get the frequencies for the pairs (distance, amount), and even round the numbers before doing aggregation since the idea is to explore if there's a trend.

```
dplot <- dreg %>%
  mutate_if(is.numeric, function(x) round(x,0)) %>%
  group_by(trip_distance, total_amount) %>%
  count()
dplot
```

## Exploring distance versus total amount



There is not an obvious trend in this data, but it's still possible to obtain a trend line.

```
ggplot(dplot, aes(x = trip_distance, y = total_amount, alpha = log(n))) +  
  geom_jitter() +  
  labs(title = "Exploring distance versus total amount")
```

You can fit a model with Gaussian errors such as the next model with intercept.

```
summary(lm(total_amount ~ trip_distance, data = dreg))  
# Call:  
#   lm(formula = total_amount ~ trip_distance, data = dreg)  
#  
# Residuals:  
#   Min       1Q   Median       3Q      Max   
# -122.736  -1.242   -0.542    0.509   228.125   
#  
# Coefficients:  
#   Estimate Std. Error t value Pr(>|t|)      
# (Intercept)  4.0253576  0.0013786   2920  <2e-16 ***  
# trip_distance 2.4388382  0.0003532   6905  <2e-16 ***  
# ---  
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
#
```

```
# Residual standard error: 3.909 on 14092411 degrees of freedom
# Multiple R-squared: 0.7718, Adjusted R-squared: 0.7718
# F-statistic: 4.767e+07 on 1 and 14092411 DF, p-value: < 2.2e-16
```

Let's say we are interested in fitting one model per month and then average the obtained coefficients to smooth a potential seasonal effect (e.g. such as holidays, Christmas, etc).

To do this you can create 1,2,...,N droplets and assign different months to each droplet and read the data from S3, to then get the regression results to your laptop.

You can start by loading the `analogsea` package and create two droplets that already include `arrow`. The next code assumes that you already registered your SSH key at `digitalocean.com`.

```
library(analogsea)
s <- "c-8" # 4 dedicated CPUs + 8GB memory, run sizes()
droplet1 <- droplet_create("RDemo1", region = "sfo3", size = s,
                           image = "rstudio-20-04", wait = F)
droplet2 <- droplet_create("RDemo2", region = "sfo3", size = s,
                           image = "rstudio-20-04", wait = T)
```

What if you need to install additional R packages in the droplets? Well, you can control that remotely after updating the droplet status and install, for example, `eflm` to speed-up the regressions.

```
droplet1 <- droplet(droplet1$id)
droplet2 <- droplet(droplet2$id)
install_r_package(droplet1, "eflm")
install_r_package(droplet2, "eflm")
```

Before proceeding, you need the droplet's IPs to create a cluster that will use each droplet as a CPU core from your laptop.

```
ip1 <- droplet(droplet1$id)$networks$v4[[2]]$ip_address
ip2 <- droplet(droplet2$id)$networks$v4[[2]]$ip_address
ips <- c(ip1, ip2)
```

Now you can create a cluster, and you need to specify which SSH key to use.

```
library(future)
ssh_private_key_file <- "~/ssh/id_rsa"
cl <- makeClusterPSOCK(
  ips,
  user = "root",
  rshopts = c(
    "-o", "StrictHostKeyChecking=no",
    "-o", "IdentitiesOnly=yes",
    "-i", ssh_private_key_file
  ),
  dryrun = FALSE
)
```

The next step is to create a working plan (i.e. 1 droplet = 1 ‘CPU core’) which allows to run processes in parallel and a function to specify which data to read and how to fit the models in the droplets.

```
plan(cluster, workers = cl)
fit_model <- function(y, m) {
  message(paste(y,m))
  suppressMessages(library(arrow))
  suppressMessages(library(dplyr))
  space <- S3FileSystem$create(
    anonymous = TRUE,
    scheme = "https",
    endpoint_override = "sfo3.digitaloceanspaces.com"
  )
  d <- open_dataset(
    space$path("nyc-taxi"),
    partitioning = c("year", "month")
  )
  d <- d %>%
    filter(year == y, month == m) %>%
    select(total_amount, trip_distance) %>%
    collect()
  fit <- try(eflm::elm(total_amount ~ trip_distance, data = d, model = F))
  if (class(fit) == "lm") fit <- fit$coefficients
  rm(d); gc()
  return(fit)
}
```

The `fit_model` function is a function that you should iterate over months and years. The `furrr` package is very efficient for these tasks and works well with the working plan defined previously, and the iteration can be run in a very similar way to `purrr`.

```
library(furrr)
fitted_models <- future_map2(
  c(rep(2009, 12), rep(2010, 12)), rep(1:12, 2), ~fit_model(.x, .y))

fitted_models[[1]]
# (Intercept) trip_distance
# 4.025358 2.438838
```

In the previous code, even when the iteration was made with two years there are some clear efficiencies. On the one hand, the data is read from S3 without leaving DigitalOcean network, and any problem with home connections is therefore eliminated because you only have to upload a text with the function to the droplets and then you get the estimated coefficients without moving gigabytes of data to your laptop. On the other, and the computation is run in parallel so I could create more droplets to reduce the reading and fitting times even more.

## Using the results from the cloud

If you want to average the estimated coefficients, you need to subset and discard the unusable months.

```
intercept <- c()
slope <- c()
for (i in seq_along(fitted_models)) {
  intercept[i] <- ifelse(is.numeric(fitted_models[[i]]),
                        fitted_models[[i]][1], NA)
  slope[i] <- ifelse(is.numeric(fitted_models[[i]]),
                    fitted_models[[i]][2], NA)
}
avg_coefficients <- c(mean(intercept, na.rm = T), mean(slope, na.rm = T))
names(avg_coefficients) <- c("intercept", "slope")
avg_coefficients
# intercept      slope
# 4.406586 2.533299
```

Now you can say that, on average, if you travelled 5 miles by taxi, you'd expect to pay  $4.4 + 2.5 \times 5 \approx 17$  USD, which is just an estimate depending on traffic conditions, finding many red lights, etc.

Don't forget to delete the droplets when you are done using them.

```
droplet_delete(droplet1)
droplet_delete(droplet2)
```

## Final remarks on cloud estimation

Arrow and S3 provide a powerful combination for big data analysis. You can even create a droplet to control other droplets if you fear that your internet connection is not that good like to transfer complete regression summaries.

Think about the possibility of fitting many other models: generalized linear models, random forest, non-linear regression, posterior bootstrap, or basically any output that R understands.