Supporting Data Processing with an Object Repository

Lukasz A. Bartnik

2018-03-18

```
library(readr)
library(lubridate)
library(magrittr)
library(dplyr)
library(ggplot2)
library(broom)
library(modelr)
library(forecast)
library(tidyr)
```

Dictionary

- `data project`
- `R session`
- `tracker`
- `session cache` (cache)
- `project repository` (repository)
- `artifact`
- `commit`
- `branch`
- `time view`
- `data view`

Example

Iterative

Iterate over a single idea, fix errors in the expression, generate a number of versions of the object and plots, finally choose objects that should make it to the persistent storage.

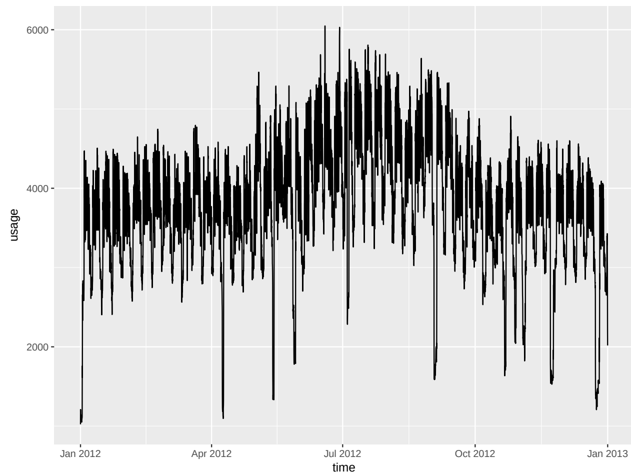
Iterate over a model

Here is a very simple example of what might happen during the initial phase of a data project. We load the data, build a few simple models and create a few plots to get the initial insights.

```
file <- 'site_10.csv'
data <- read_csv(file)
```

Let's look at the whole data, one year worth of hourly readings.

```
ggplot(data, aes(x = time, y = usage)) + geom_line()
```



Let's start with monthly averages.

```
m <- data %>%
  mutate(month = as.factor(month(time, label = TRUE))) %>%
  {lm(usage ~ month, .)}
glance(m)
#>   r.squared adj.r.squared    sigma statistic p.value df    logLik      AIC
#> 1 0.2492309     0.2482895 672.7854   264.729       0 12 -69654.32 139334.6
#>        BIC   deviance df.residual
#> 1 139426.7 3970560289        8772
```

There seems to be some regularity in the data. Let's see if there average usage depends on the day of the week.

```
m <- data %>%
  mutate(wday = wday(time, label = TRUE)) %>%
  {lm(usage ~ wday, .)}
glance(m)
#>   r.squared adj.r.squared    sigma statistic p.value df    logLik      AIC
#> 1 0.2556058     0.2550969 669.7322  502.2992       0  7 -69616.87 139249.7
#>        BIC   deviance df.residual
#> 1 139306.4 3936845781        8777
tidy(m)
#>          term   estimate std.error   statistic       p.value
#> 1 (Intercept) 3821.50985  7.146133 534.7661578  0.000000e+00
#> 2      wday.L  533.24684 18.874839  28.2517299 3.838396e-168
#> 3      wday.Q -878.66604 18.904755 -46.4785743  0.000000e+00
#> 4      wday.C  -12.75333 18.898348  -0.6748383  4.997963e-01
#> 5      wday^4  -38.25626 18.890579  -2.0251500  4.288217e-02
#> 6      wday^5 -106.69017 18.921828  -5.6384706  1.768815e-08
#> 7      wday^6   35.16645 18.950895   1.8556616  6.353531e-02
```

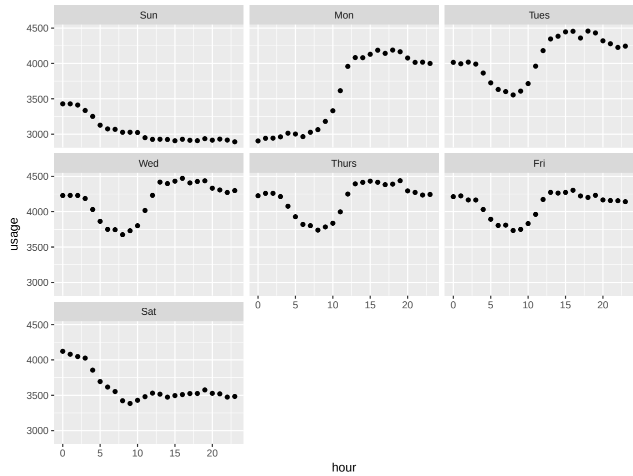
Now let's look at the hourly profile.

```
m <- data %>%
  mutate(hour = as.factor(hour(time))) %>%
  {lm(usage ~ hour, .)}
glance(m)
#>    r.squared adj.r.squared   sigma statistic      p.value df   logLik
#> 1 0.05945789    0.05698844 753.546  24.07729 5.111599e-99 24 -70644.1
```

```
#>        AIC      BIC   deviance df.residual
#> 1 141338.2 141515.2 4974204677        8760
```

Interesting, both day of the week and the hour of the day can be used to model the usage. Let's see what happens when we combine them.

```
data %>%
  group_by(hour = hour(time), wday = wday(time, label = TRUE)) %>%
  summarise(usage = mean(usage, na.rm = TRUE)) %>%
  ggplot(aes(x = hour, y = usage)) + geom_point() + facet_wrap(~wday)
```



And now a quantitative assessment of the same model.

```
m <-
  data %>%
  mutate(hour = as.factor(hour(time)), wday = wday(time, label = TRUE)) %>%
  {lm(usage ~ hour:wday, .)}
glance(m)
#>   r.squared adj.r.squared    sigma statistic p.value  df    logLik
#> 1 0.3992273     0.3875828 607.2604  34.28462       0 168 -68675.43
#>        AIC      BIC   deviance df.residual
#> 1 137688.9 138885.5 3177280619        8616
```

Finally, let's combine all three input features of our model.

```
m <-
  data %>%
  mutate(hour = as.factor(hour(time)), wday = wday(time, label = TRUE),
         month = as.factor(month(time))) %>%
  {lm(usage ~ month + hour:wday, .)}
glance(m)
#>   r.squared adj.r.squared    sigma statistic p.value  df    logLik
#> 1 0.6420992     0.6346958 469.0062  86.73018       0 179 -66400.54
#>        AIC      BIC   deviance df.residual
#> 1 133161.1 134435.6 1892814807        8605
```

Browse the history

Up to this point the session cache collected a number of artifacts. Let's see the summary of the session cache.

```
tracker
#> Tracker :  ON
#> Branch  :  "default"
#> Commits :  15 in branch, 15 total
```

This is what the history graph looks like. First, in the dimension of time, that is, where nodes are connected according to the order in which they were created.

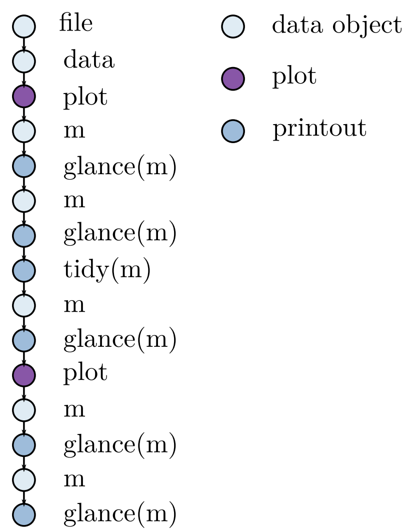


Figure 1: svg image

Another way to look at the history is the dimension of origin, that is, the parent-child relationship between artifacts.

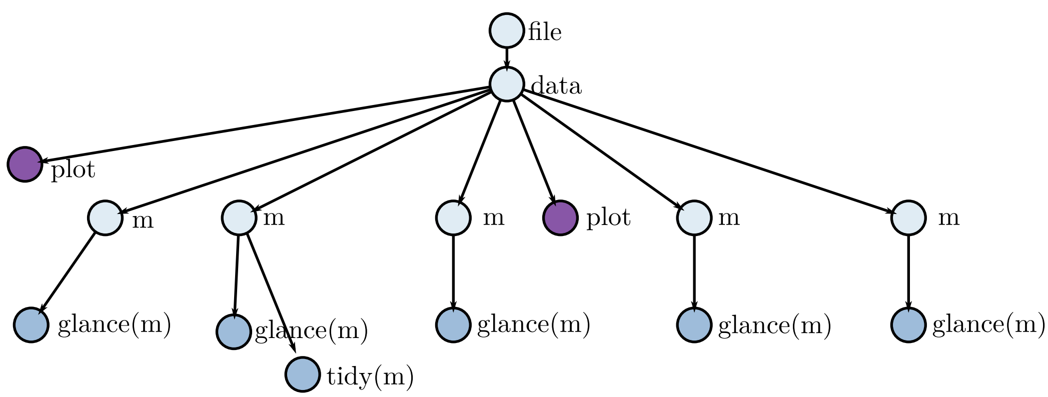


Figure 2: svg image

The information depicted in the first graph can be accessed in R session via the `tracker$history` facility. The example below follows up on our sample R session. Commits recorded in the session repository are printed out starting with the oldest. The most recent commit is assigned index `a`, the one before is assigned index `b` and so on. Indices can be used to point to a specific step in the history. For each step we print out the list of artifact names and the expression abbreviated to one line.

In actual R session variables which are introduced or replaced are printed in green.

```
tracker$history(n = 50)
#> o: file
```

```
#>    file <- 'site_10.csv'
#> n: data file
#>    data <- read_csv(file)
#> m: data file [plot]
#>    ggplot(data, aes(x = time, y = usage)) + geom_line()
#> l: data file m
#>    m <- data %>% mutate(month = as.factor(month(time, ...
#> k: data file m [printout]
#>    glance(m)
#> j: data file m
#>    m <- data %>% mutate(wday = wday(time, label = TRUE)) ...
#> i: data file m [printout]
#>    glance(m)
#> h: data file m [printout]
#>    tidy(m)
#> g: data file m
#>    m <- data %>% mutate(hour = as.factor(hour(time))) ...
#> f: data file m [printout]
#>    glance(m)
#> e: data file m [plot]
#>    data %>% group_by(hour = hour(time), wday = wday(time, ...
#> d: data file m
#>    m <- data %>% mutate(hour = as.factor(hour(time)), ...
#> c: data file m [printout]
#>    glance(m)
#> b: data file m
#>    m <- data %>% mutate(hour = as.factor(hour(time)), ...
#> a: data file m [printout]
#>    glance(m)
```

Each commit can be printed in more detail. This is where the letters-indices come in handy.

```
tracker$history$b
#> Commit afb67ad6
#> data :  data.frame[8784, 2]
#> file :  character(1)
#> m    :  lm(formula = usage ~ month + hour:wday, data = .)
#>
#>   m <-
#>    data %>%
#>    mutate(hour = as.factor(hour(time)), wday = wday(time, label = TRUE),
#>           month = as.factor(month(time))) %>%
#>    {lm(usage ~ month + hour:wday, .)}
```

Since we have found an interesting artifact, we can now instruct `tracker` to store it in the persistent repository of artifacts. We point to one object but the whole sequence of objects required to create it is copied from the session repository to the persistent one. If any of the objects is already present in the persistent repository (`session cache` vs `repository`) the ends are connected and a consistent graph of artifacts is maintained.

```
tracker$history$b$m %>% persist
#> Persisted 3 objects: m, data, file
```

Let's all persist the final profile plot.

```
tracker$history$e$`[plot]` %>% persist
#> Persisted 1 plot.
```

Repeat the Sequence

Another useful tool is the `replay` function. It accepts a list of objects that were changed, finds their originals, and re-computes all nodes descendant in the `data` view. In our example we replace the data file name and then ask R to recalculate all object below it. It creates a new branch in the session cache and names it `"replay(file) #1"`.

```
file <- 'site_88.csv'
replay(file)
#> Objects changed: file
#> Recalculating: 6 objects, 2 plots, and 6 printouts.
#> Branch name: "replay(file) #1"
```

Let's look at the state of the tracker.

```
tracker
#> Tracker :  ON
#> Branch  :  "replay(file) #1"
#> Commits :  15 in branch, 30 total
```

The current R session contains a new model, built with the same pipeline but for a different data set. Let's take a look at the model and compare it to the one already stored in the project repository. The new model:

```
glance(m)
```

```
#> Parsed with column specification:
#> cols(
#>   time = col_datetime(format = ""),
#>   usage = col_double()
#> )
#>   r.squared adj.r.squared    sigma statistic p.value  df    logLik     AIC
#> 1 0.6695356     0.6626998 23.37094  97.94447       0 179 -40056.25 80472.5
#>        BIC deviance df.residual
#> 1 81747.02  4700058        8605
```

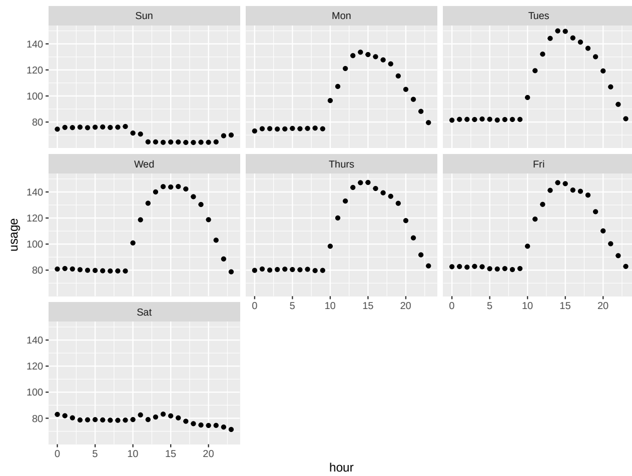
The model built for **site 10**:

```
glance(tracker$repository$m)
```

```
#>   r.squared adj.r.squared    sigma statistic p.value  df    logLik
#> 1 0.6420992     0.6346958 469.0062  86.73018       0 179 -66400.54
#>        AIC      BIC   deviance df.residual
#> 1 133161.1 134435.6 1892814807        8605
```

Let's also take a look at the final profile plot for **site 88**. In the session history it can be accessed via:

```
tracker$history$e$`[plot]`
```



Look back, change input, generate two new models for two other buildings; use `repeat`

Browse new branches, choose the two new models and store them persistently

Browse the Tree of Understanding

Look at the branches stored in the persistent store, this time presented according to their origin