Workshop - drake for Reproducible Workflows at Scale

Andrew Stewart

Andrew.Stewart@manchester.ac.uk

@ajstewart_lang

https://github.com/ajstewartlang

Introducing drake

"Data analysis can be slow. A round of scientific computation can take several minutes, hours, or even days to complete. After it finishes, if you update your code or data, your hard-earned results may no longer be valid. How much of that valuable output can you keep, and how much do you need to update? How much runtime must you endure all over again?

For projects in R, the `drake` *package can help. It analyzes your workflow, skips steps with up-to-date results, and orchestrates the rest with optional distributed computing. At the end,* `drake` *provides evidence that your results match the underlying code and data, which increases your ability to trust your research."*

https://ropenscilabs.github.io/drake-manual/index.html

Too many data science projects follow a Sisyphean loop:

- Launch the code.
- Wait while it runs.
- Discover an issue.
- Restart from scratch.

For projects with long runtimes, people tend to get stuck.

But with `drake`, you can automatically:

- Launch the parts that changed since last time.
- Skip the rest.

Tidy Tuesday dataset (Nov 5th, 2019) of commuting via walking/biking in US Cities 2008-2012

Data Dictionary

`commute.csv`

variable	class	description
city	character	City
state	character	State
city_size	character	City Size * Small = 20K to 99,999 * Medium = 100K to 199,999 * Large = >= 200K
mode	character	Mode of transport, either walk or bike
n	double	N of individuals
percent	double	Percent of total individuals
moe	double	Margin of Error (percent)
state_abb	character	Abbreviated state name
state_region	character	ACS State region

```
commute_mode <-
readr::read_csv("https://
raw.githubusercontent.com
/rfordatascience/
tidytuesday/master/data/
2019/2019-11-05/
commute.csv")
```

Getting started...

We first need to install the drake package with `install.packages("drake")`

When writing our script for visualising and understanding the dataset, we need to write our code chunks as *functions*.

When we write our drake_plan (which details the steps of our analysis) we list the various steps in our workflow - these are called *targets*.

Associated with each target is the command that is executed. These commands can be simple R commands, chunks of code or (better yet in terms of readability) functions that you have written for each analysis step.

- Let's write an analysis that does the following:
    - Reads in our datafile.
    - Generates a plot to visualise the top 10 States in the US with the highest % of people commuting via walking.
    - Generates a plot to visualise the % of people commuting via walking for small vs. medium vs. large sized cities.
    - Generates summary descriptives of the mean and sd of % people commuting via walking for small vs. medium vs. large sized cities.
    - Builds a linear model to examine how % of people commuting via walking is predicted by the size of a US city.
    - Produces a summary of the output of this model.

- Let's write an analysis that does the following:
    - Reads in our datafile.
    - Generates a plot to visualise the top 10 States in the US with the highest % of people commuting via walking.
    - Generates a plot to visualise the % of people commuting via walking for small vs. medium vs. large sized cities.
    - Generates summary descriptives of the mean and sd of % people commuting via walking for small vs. medium vs. large sized cities.
    - Builds a linear model to examine how % of people commuting via walking is predicted by the size of a US city.
    - Produces a summary of the output of this model.

Function - visualise the top 10 States in the US with the highest % of people commuting via walking

```
my_overall_plot <- function(x) {
  x %>%
  group_by(state, mode) %>%
  summarise(mean_percent = mean(percent)) %>%
  ungroup() %>%
  filter(mode == "Walk") %>%
  arrange(-mean_percent) %>%
  top_n(10, mean_percent) %>%
  ggplot(aes(x = fct_reorder(state, mean_percent, median),
             y = mean_percent,
             fill = state)) +
  geom_col() +
  guides(fill = FALSE) +
  coord_flip() +
  labs(x = "State",
       y = "Percentage of Walkers",
       title = "States with the Highest Percentage of Walkers") +
  theme(text = element_text(size = 10)) +
  theme_minimal()
}
```

Function - visualise the % of people commuting via walking for small vs. medium vs. large sized cities

```
my_walk_plot <- function(x) {
    x %>%
    filter(mode == "Walk") %>%
    group_by(city_size) %>%
    ggplot(aes(x = city_size, y = percent, colour = city_size)) +
      geom_jitter(width = .1, size = 3, alpha = .25) +
      guides(colour = FALSE) +
    labs(title = "% of Walkers by City Size",
         x = "City Size",
         y = "Percent of Walkers") +
    theme(text = element_text(size = 12))
}
```

Function - summary descriptives of the mean and sd of % people commuting via walking for small vs. medium vs. large sized cities

```
desc_stats <- function(x) {
    x %>%
    filter(mode == "Walk") %>%
    group_by(city_size) %>%
    summarise(mean_walk = mean(percent), sd_walk = sd(percent))
}
```

We can then build our drake plan for our 6 analysis steps and call the functions we have just written...

```
my_plan <- drake_plan(
  commute_mode = read_csv("https://raw.githubusercontent.com/
rfordatascience/tidytuesday/master/data/2019/2019-11-05/
commute.csv"),
  show_overall_plot = my_overall_plot(commute_mode),
  show_walk_plot = my_walk_plot(commute_mode),
  show_stats = desc_stats(commute_mode),
  fit = lm(percent ~ city_size, data = filter(commute_mode, mode ==
"Walk")),
  summary_fit = summary(fit)
)
```

We have 6 targets - `commute_mode`, `show_overall_plot`, `show_walk_plot`, `show_stats`, `fit`, and `summary_fit`

Associated with each target is the appropriate code - for targets 2, 3, and 4 this is for functions we have written.

- You can see the script.R file that contains all the code on the preceding slides here:

https://github.com/ajstewartlang/drake_workshop/blob/master/script.R

We can examine the configuration of our drake plan

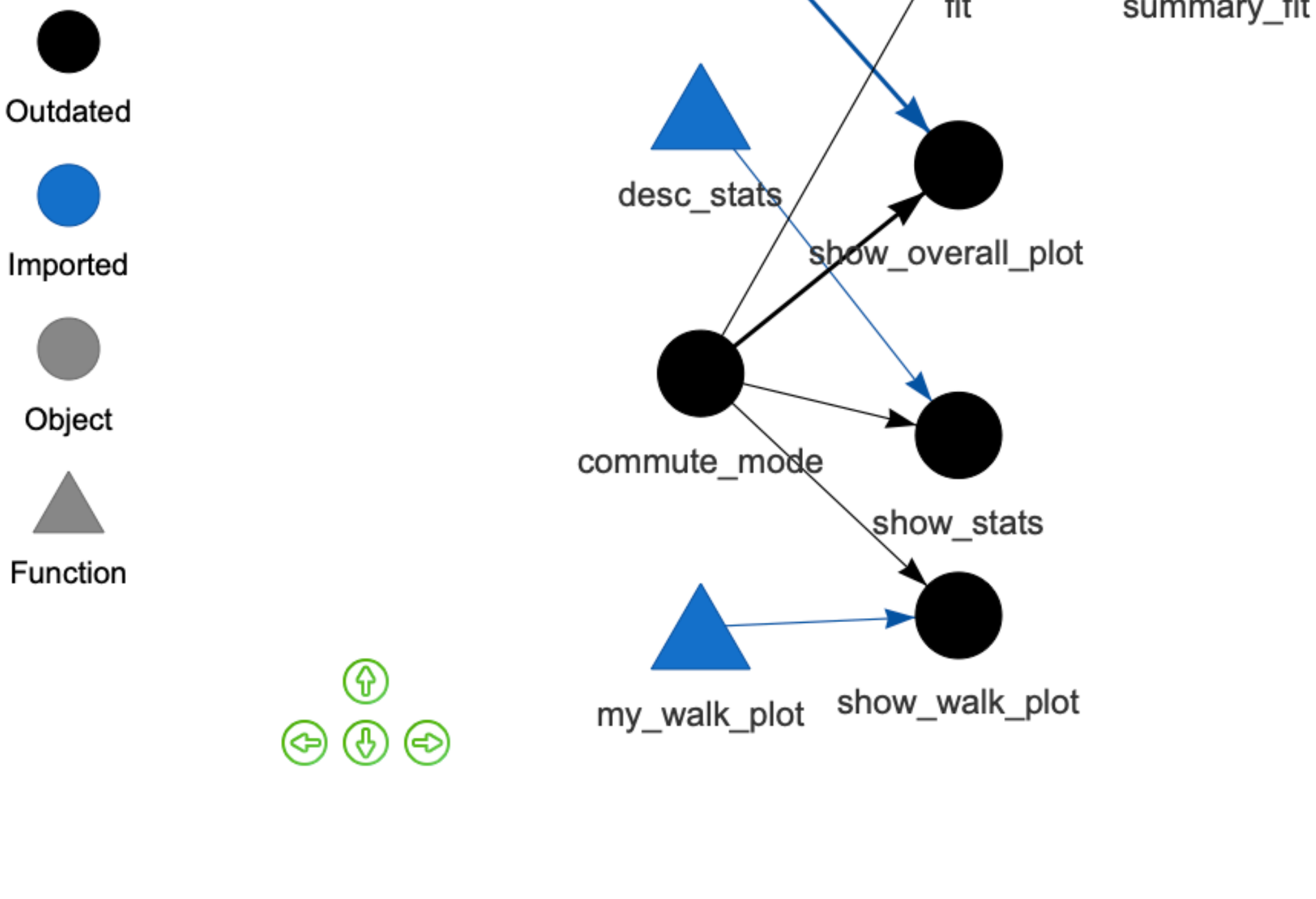
As text...

```
> my_plan
# A tibble: 6 x 2
  target           command
  <chr>            <expr>
1 commute_mode     read_csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/m…
2 show_overall_p… my_overall_plot(commute_mode)                                              …
3 show_walk_plot  my_walk_plot(commute_mode)                                                 …
4 show_stats      desc_stats(commute_mode)                                                   …
5 fit             lm(percent ~ city_size, data = filter(commute_mode, mode == "Walk"))       …
6 summary_fit     summary(fit)
```

Or graphically...

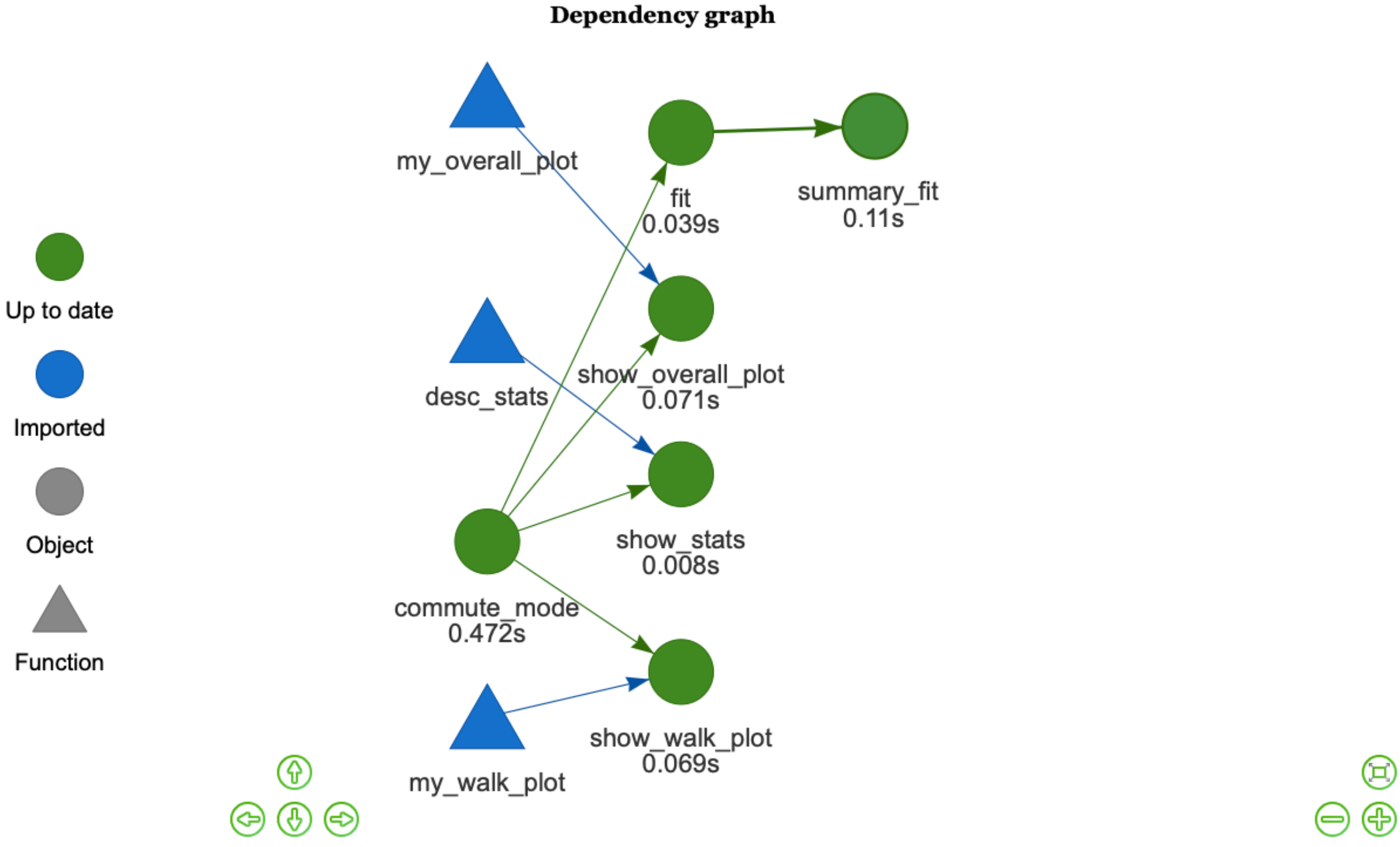
```
config <- drake_config(my_plan)
vis_drake_graph(config)
```

Dependency graph



Executing our plan...

```
> make(my_plan)
target commute_mode
Target commute_mode messages:
  Parsed with column specification:
cols(
  city = col_character(),
  state = col_character(),
  city_size = col_character(),
  mode = col_character(),
  n = col_double(),
  percent = col_double(),
  moe = col_double(),
  state_abb = col_character(),

  state_region = col_character()

)

target fit

target show_overall_plot

target show_walk_plot

target show_stats

target summary_fit
```


Dependency graph
Up to date
Imported
Object
Function
my_overall_plot
fit
0.039s
summary_fit
0.11s
desc_stats
show_overall_plot
0.071s
show_stats
0.008s
commute_mode
0.472s
show_walk_plot
0.069s
my_walk_plot

Examine the history of `making` your `drake` plan

```
> drake_history(analyze = TRUE)
# A tibble: 6 x 8
  target     current built           exists hash     command                                  seed runtime
  <chr>      <lgl>   <chr>           <lgl>  <chr>    <chr>                                   <int>   <dbl>
1 commute_m… TRUE    2019-11-11 17… TRUE   4444ca6… "read_csv(\"https://raw.githubuser…  2.76e8  0.508
2 fit        TRUE    2019-11-11 17… TRUE   6d3d32e… "lm(percent ~ city_size, data = fi…  1.11e9  0.004
3 show_over… TRUE    2019-11-11 17… TRUE   6d72488… my_overall_plot(commute_mode)        1.85e9  0.0410
4 show_stats TRUE    2019-11-11 17… TRUE   a8b32c5… desc_stats(commute_mode)             1.08e9  0.004
5 show_walk… TRUE    2019-11-11 17… TRUE   a16a871… my_walk_plot(commute_mode)           1.67e9  0.0150
6 summary_f… TRUE    2019-11-11 17… TRUE   f430345… summary(fit)                        2.09e9  0.002
```

`drake` keeps track of which components have been run so if you were to update your code and (re)make you drake plan, only the bits of code that have changed will be run. You can look at the runtime column to see which bit of code took the longest to execute.

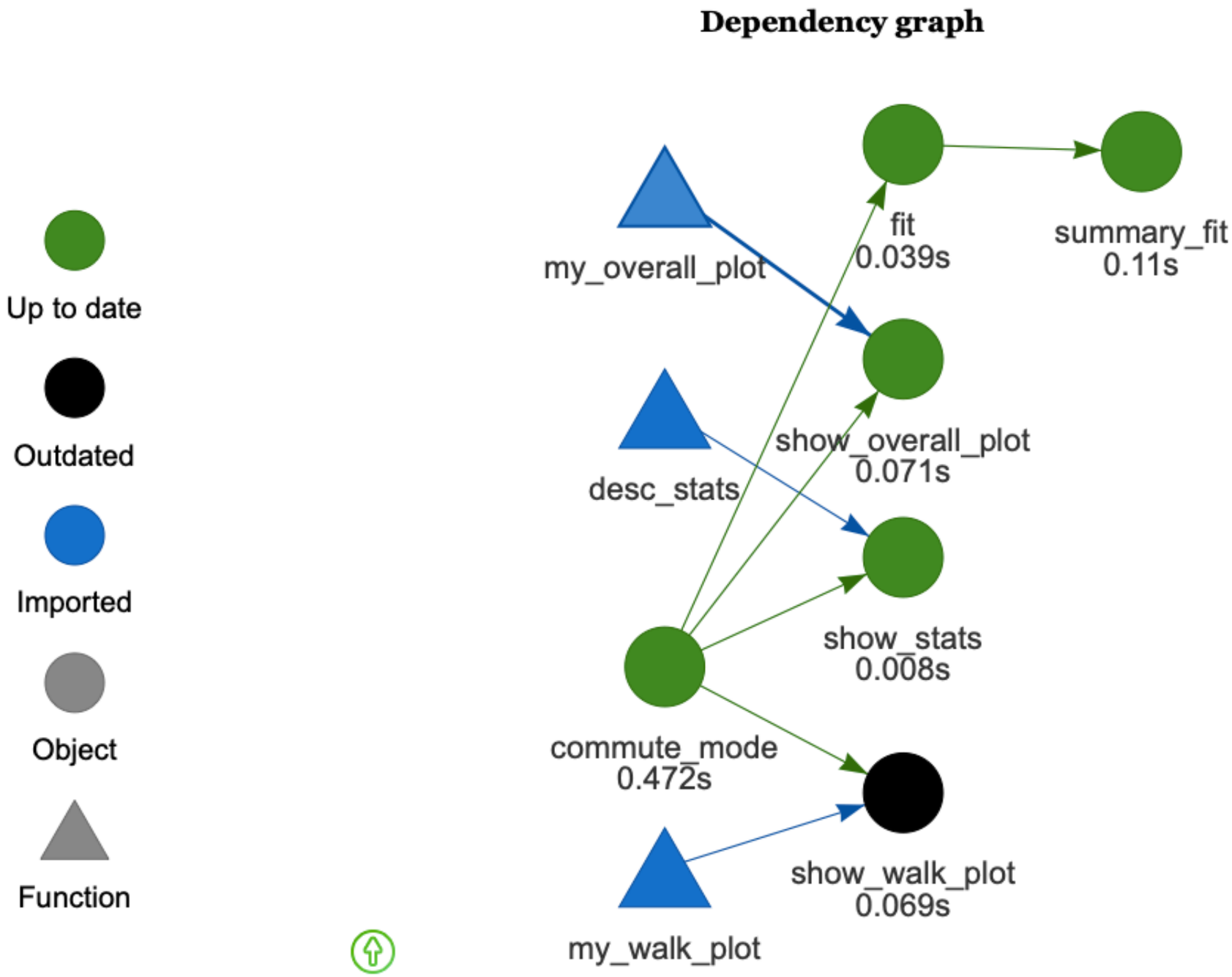
This is super useful is you don't want to have to re-run computationally intensive bits of code again - and only the bits you have changed.

For Example

- Let's change one of our plot functions - let's keep is simple and just add a `coord_flip()` to the `my_walk_plot()` function:

```
my_walk_plot <- function(x) {
    x %>%
    filter(mode == "Walk") %>%
    group_by(city_size) %>%
    ggplot(aes(x = city_size, y = percent, colour = city_size)) +
      geom_jitter(width = .1, size = 3, alpha = .25) +
      guides(colour = FALSE) +
    labs(title = "% of Walkers by City Size",
         x = "City Size",
         y = "Percent of Walkers") +
    theme(text = element_text(size = 12)) +
    coord_flip()
}
```

- Let's run that function again and then look at the configuration of our plan with `vis_drake_graph(config)`



- The `show_walk_plot()` function is in black to indicate it is outdated.

- If we now (re)make our drake plan, only that component is actually run!

```
> make(my_plan)
target show_walk_plot
```

- And if we look at `drake_history()` we see that the `show_walk_plot()` function has now been run twice - with the most recent version being marked a the current one.

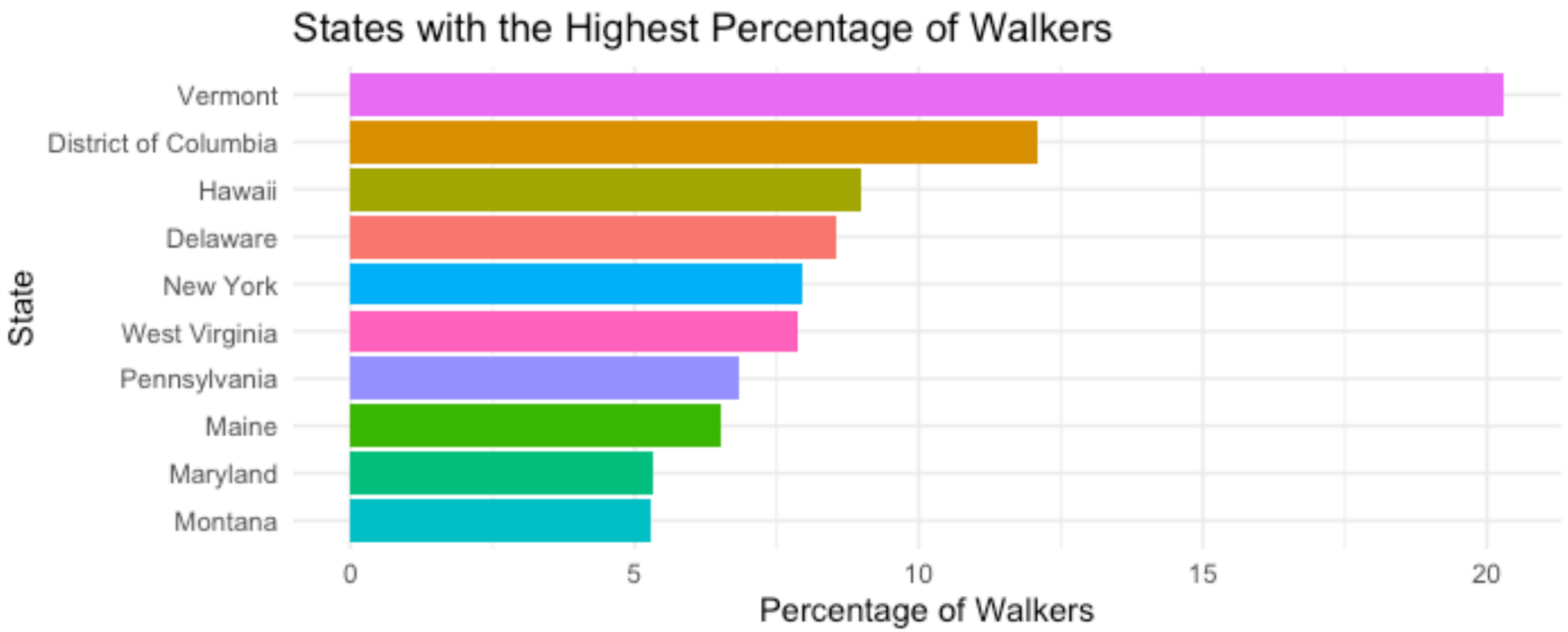
```
> drake_history(analyze = TRUE)
# A tibble: 7 x 8
  target     current built          exists hash     command                                  seed runtime
  <chr>      <lgl>   <chr>          <lgl>  <chr>    <chr>                                   <int>   <dbl>
1 commute_m… TRUE    2019-11-11 17… TRUE   4444ca6… "read_csv(\"https://raw.githubuser…  2.76e8 0.451
2 fit        TRUE    2019-11-11 17… TRUE   6d3d32e… "lm(percent ~ city_size, data = fi…  1.11e9 0.006
3 show_over… TRUE    2019-11-11 17… TRUE   6d72488… my_overall_plot(commute_mode)        1.85e9 0.0350
4 show_stats TRUE    2019-11-11 17… TRUE   a8b32c5… desc_stats(commute_mode)             1.08e9 0.00300
5 show_walk… FALSE   2019-11-11 17… TRUE   a16a871… my_walk_plot(commute_mode)           1.67e9 0.01
6 show_walk… TRUE    2019-11-11 17… TRUE   c965d4f… my_walk_plot(commute_mode)           1.67e9 0.008
7 summary_f… TRUE    2019-11-11 17… TRUE   f430345… summary(fit)                        2.09e9 0
```

- When we run `make()`, drake stores the targets in a cache in the `.drake` folder.

Read and return a target from the cache

- We can use the `readd()` function to read and return the contents of a target in our cache.

- So, `readd(show_overall_plot)` will return:



- While `readd(summary_fit)` will return the results of the linear model we built:

```
> readd(summary_fit)

Call:
lm(formula = percent ~ city_size, data = filter(commute_mode,
    mode == "Walk"))

Residuals:
   Min     1Q Median     3Q    Max
-2.847 -1.847 -1.047  0.353 39.553

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       3.4422     0.3338  10.311   <2e-16 ***
city_sizeMedium  -0.4354     0.4248  -1.025   0.3056
city_sizeSmall   -0.5950     0.3460  -1.720   0.0857 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Residual standard error: 3.485 on 1745 degrees of freedom

Multiple R-squared:  0.001796,    Adjusted R-squared:  0.0006519

F-statistic:  1.57 on 2 and 1745 DF,  p-value: 0.2084
```

A consistent and reliable workflow using `r_make()`

- If you change your code interactively (i.e., as you work in RStudio), you might end up making changes that accidentally invalidate targets - but you won't necessarily notice at the time.

- One way to avoid this issue is to build your workflow in a new, temporary, R session. This will result in a more consistent and reliable workflow.

For `r_make()` you need to do things slightly differently...

- You need to have a configuration script (the default is called `_drake.R` sitting above another folder (which we're calling `R)` which contains separate files for each bit of your make plan (and you need the drake plan itself as a separate file):

```
_drake.R
R/
├── packages.R
├── functions.R
└── plan.R
```

- Here we have 3 .R files saved in our R folder.

- So our _drake.R file contains the following code:

```
source("R/packages.R")

source("R/functions.R")

source("R/plan.R")

drake_config(my_plan, verbose = 2)
```

- The source() function reads the code from the named file (could also be a URL) - this means that with `source("R/functions.R")` you can then use the functions contained within this file in your current environment.

- Our `packages.R` file contains this code:

```
library(drake)
library(tidyverse)
```

- While our `functions.R` file contains the code for all our functions such as the `desc_stats()` function here:

```
desc_stats <- function(x) {
  x %>%
    filter(mode == "Walk") %>%
    group_by(city_size) %>%
    summarise(mean_walk = mean(percent), sd_walk = sd(percent))
}
```

- Our plan.R file is simply the code for your drake plan:

```
my_plan <- drake_plan(
  commute_mode = read_csv("https://
raw.githubusercontent.com/rfordatascience/tidytuesday/
master/data/2019/2019-11-05/commute.csv"),
  show_overall_plot = my_overall_plot(commute_mode),
  show_walk_plot = my_walk_plot(commute_mode),
  show_stats = desc_stats(commute_mode),
  fit = lm(percent ~ city_size, data = filter(commute_mode,
mode == "Walk")),
  summary_fit = summary(fit)
)
```

- With our config file set up and your .R files associated with loading your packages, functions etc. all in the right place you can simply type `r_make()` and your plan etc. will be loaded and run in your new temporary environment.

```
> r_make()
── Attaching packages ─────────────────────────────────────────────────────── tidyverse 1.2.1 ──
✔ ggplot2 3.2.1     ✔ purrr   0.3.2
✔ tibble  2.1.3     ✔ dplyr   0.8.3
✔ tidyr   1.0.0     ✔ stringr 1.4.0
✔ readr   1.3.1     ✔ forcats 0.4.0
── Conflicts ────────────────────────────────────────────────────── tidyverse_conflicts() ──
✖ tidyr::expand() masks drake::expand()
✖ dplyr::filter() masks stats::filter()
✖ tidyr::gather() masks drake::gather()
✖ dplyr::lag()    masks stats::lag()
-
target commute_mode
Target commute_mode messages:
  Parsed with column specification:
cols(
  city = col_character(),
  state = col_character(),
  city_size = col_character(),
  mode = col_character(),
  n = col_double(),
  percent = col_double(),
  moe = col_double(),
  state_abb = col_character(),
  state_region = col_character()
)
target fit
target show_overall_plot
target show_walk_plot
target show_stats
target summary_fit
```

A few other things...

- If you want to force targets to be out of date in the cache (maybe you want to re-run **everything**) you can use the `clean()` function.

- If you accidentally delete targets that you didn't mean to, you can try to recover them using `make(my_plan, recover = TRUE)`

- You can find out the dependencies for your targets like this:

```
> deps_target("show_walk_plot", config)
# A tibble: 2 x 2
  name          type
  <chr>         <chr>
1 my_walk_plot globals
2 commute_mode globals
```

A few other things...

- For analyses that can take hours (or days) to run locally, you can thrown your drake plan (and hence analysis) up to an HPC cluster, run it as a persistent background process, run targets in parallel, or use multiple cores on your own machine.

- drake is really good in terms of scaleability.

- More info. about these aspects here:

https://books.ropensci.org/drake/hpc.html

- You can find all the .R example files (e.g., `_drake.R`, `functions.R`) associated with these slides here:

https://github.com/ajstewartlang/drake_workshop