Data science and analysis in Neuroscience

Kevin Allen

December 5, 2019

Today's plan

1. Consolidation of dplyr and ggplot
2. Saving a graph
3. Plot with several graphs
4. git: create your git repository on github
5. Relational data with dplyr

dplyr and ggplot

Load the tmaze data set for a few more exercises.

```
myFile="~/repo/dataNeuroCourse/dataSets/tmaze.csv"
df<-read_csv(myFile)
```

```
## Parsed with column specification:
## cols(
##   mouse = col_character(),
##   date = col_date(format = ""),
##   injection = col_character(),
##   block = col_double(),
##   trialNo = col_double(),
##   sample = col_character(),
##   choice = col_character()
## )
```

```
df<-mutate(df, correct = sample != choice)
```

dplyr and ggplot

The is a possibility that some mice had a bad performance because they always visited the same goal arm instead of alternating.

How could we process the data to get a score per mouse telling us if the mouse always choose the same arm?

Break down your programming task into smaller tasks.

- Which column contains this information?
- How can we go from the values in this column to a statistic (number)?
- Once you have the information, choose a plot?

Alternatively

- Imagine which plot would convey your message best (draw some on paper)
- Find out which steps are needed to produce this graph.

To get a statistic, we need to transform a `chr` to something we can count (e.g., `dbl` or `lgl`). Change `R` and `L` to `0` and `1` using `ifelse`.

```
df <- df %>% mutate(turn = ifelse( choice == "R" , 0, 1))
head(df,n=5)
```

```
## # A tibble: 5 x 9
##   mouse  date       injection block trialNo sample choice correct  turn
##   <chr>  <date>     <chr>     <dbl>   <dbl> <chr>  <chr>  <lgl>   <dbl>
## 1 Mn4656 2019-10-09 Saline        1       1 L      R      TRUE        0
## 2 Mn4656 2019-10-09 Saline        1       2 L      L      FALSE       1
## 3 Mn4656 2019-10-09 Saline        1       3 R      R      FALSE       0
## 4 Mn4656 2019-10-09 Saline        1       4 L      R      TRUE        0
## 5 Mn4656 2019-10-09 Saline        1       5 R      L      TRUE        1
```

Now summarize this information as a new variable for each mouse on each block.

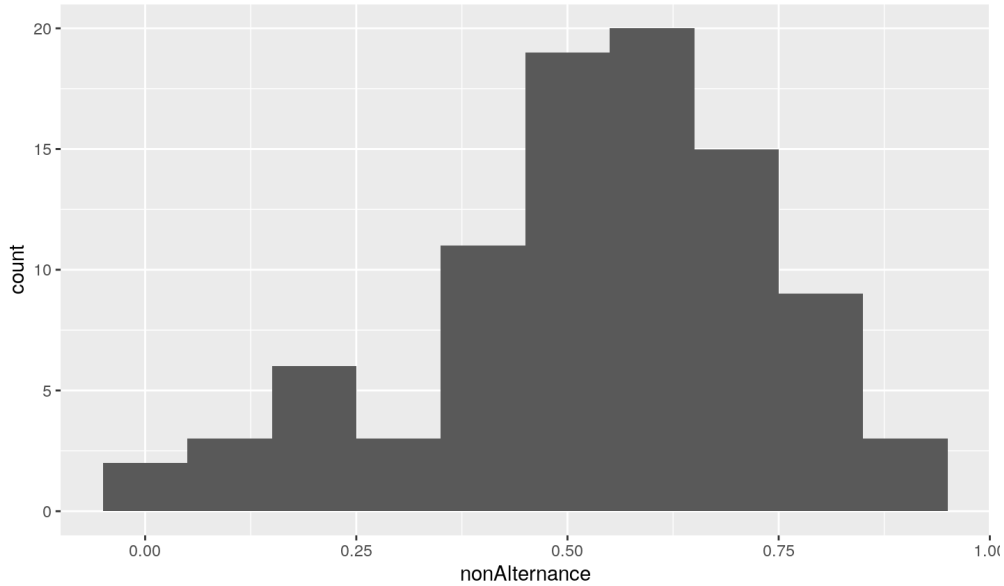
Also get the performance from the `correct` column, as done last week.

```
df1 <- df %>%
  group_by(mouse,block) %>%
  summarise(nonAlternance = mean(turn),performance = mean(correct))
df1
```

```
## # A tibble: 91 x 4
## # Groups:   mouse [7]
##    mouse  block nonAlternance performance
##    <chr>  <dbl>         <dbl>       <dbl>
##  1 Mn4656     1          0.45        0.85
##  2 Mn4656     2          0.55        0.75
##  3 Mn4656     3          0.75        0.55
##  4 Mn4656     4          0.7         0.7
##  5 Mn4656     5          0.6         0.5
##  6 Mn4656     6          0.6         0.7
##  7 Mn4656     7          0.5         1
##  8 Mn4656     8          0.6         0.5
##  9 Mn4656     9          0.8         0.7
## 10 Mn4656    10          0.6         0.7
## # … with 81 more rows
```

To know if mice prefer left or right turns, plot the distribution of nonAlternation.

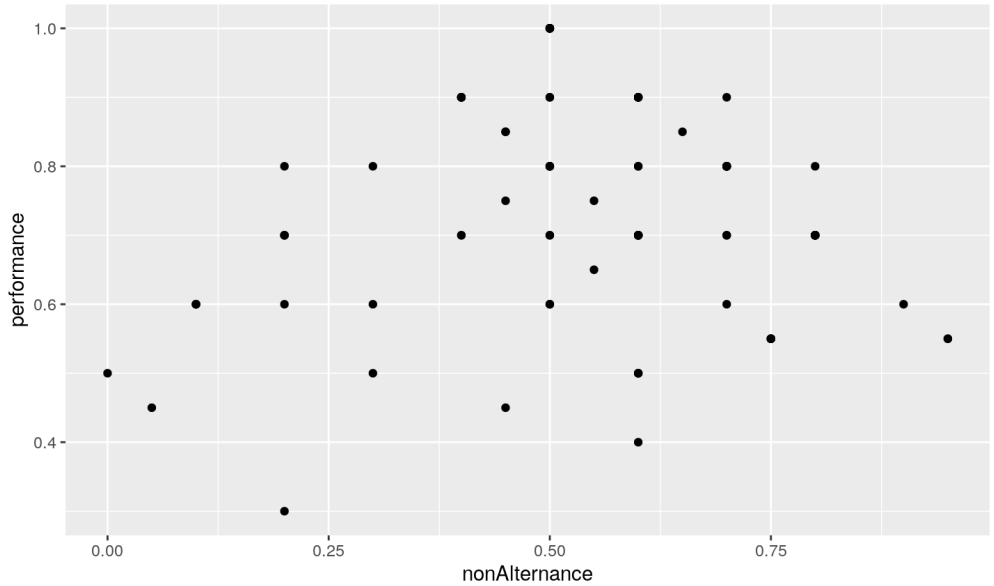
```
df1 %>%
  ggplot() +
  geom_histogram(mapping = aes(x = nonAlternance),binwidth = 0.1)
```



Mice have a small tendency to turn right in our experimental room.

Plot the performance and nonAlternation to see how the two variables are related.

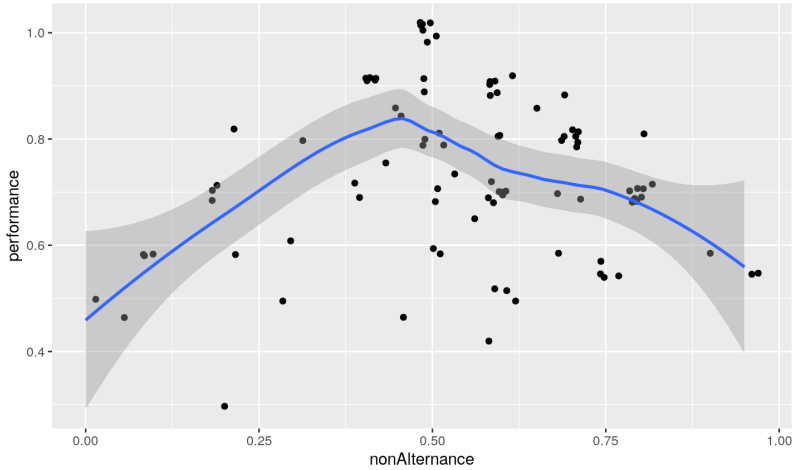
```
df1 %>%
  ggplot() +
  geom_point(mapping = aes(x = nonAlternance, y = performance))
```



Most points are on top of each other. This is called overplotting.

Try solving this by setting `position="jitter"` or `alpha=0.1`.

```
df1 %>%
  ggplot() +
  geom_point(mapping = aes(x = nonAlternance, y = performance),
             position = "jitter") +
  geom_smooth(mapping = aes(x = nonAlternance, y = performance),
              method = "loess")
```

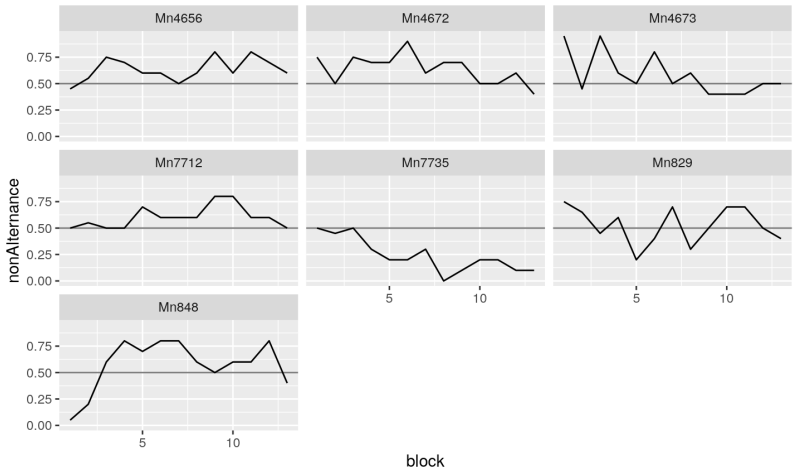


The geom_smooth helps you see the pattern in the data.

If a mouse always goes to the same goal arm (e.g., left), it won't be performing well.

Plot the nonAlternance on every block for every mice, with our chance lever line at 0.5.

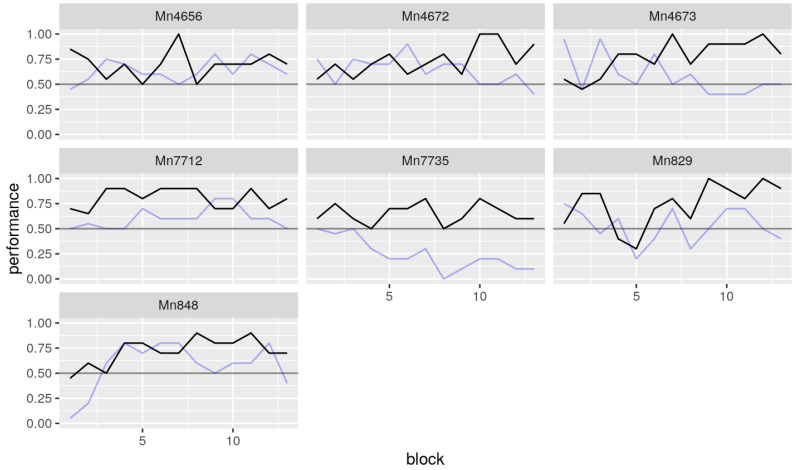
```
df1 %>%
  ggplot() +
  geom_line(mapping=aes(x=block,y=nonAlternance)) +
  geom_hline(yintercept = 0.5, alpha=0.5) +
  facet_wrap(~mouse)
```



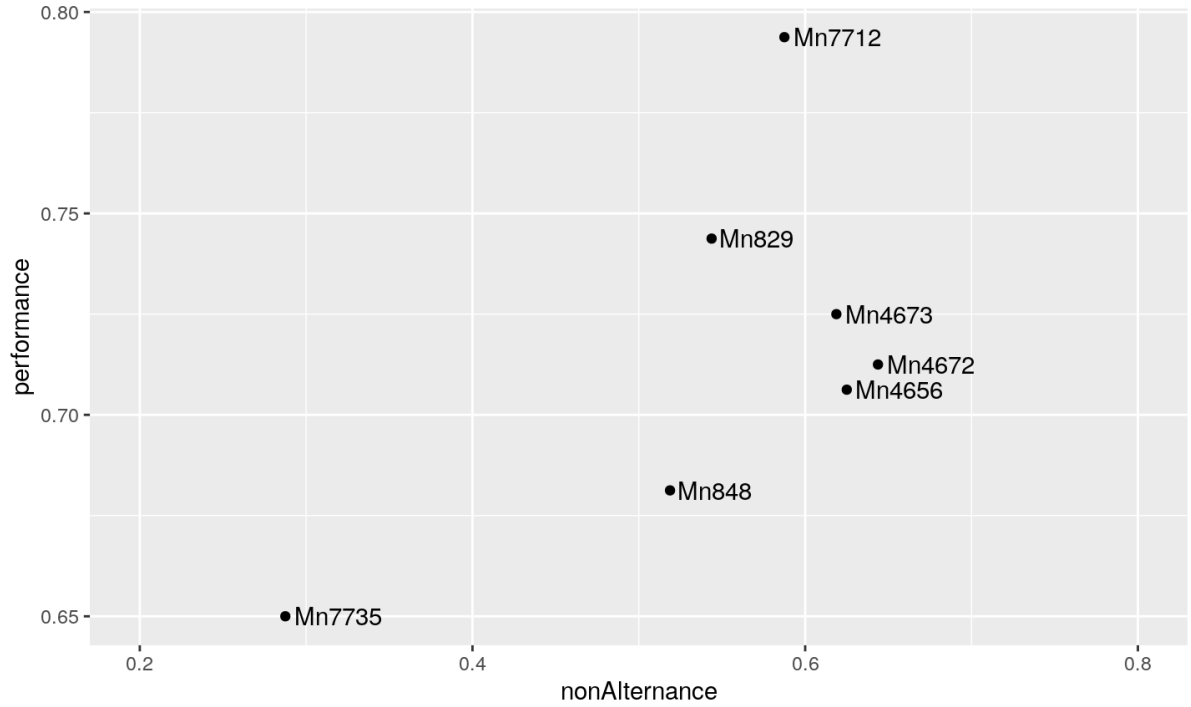
Most mice had a tendency to turn left, but one (Mn7735) had a persistent tendency to turn right.

Can you add the performance to this graph?

```
df1 %>%
  ggplot() +
  geom_line(mapping=aes(x=block,y=performance) ) +
  geom_line(mapping=aes(x=block,y=nonAlternance),color="blue",alpha=0.3) +
  geom_hline(yintercept = 0.5, alpha=0.5) +
  facet_wrap(~mouse)
```



```
df %>%
  group_by(mouse) %>%
  summarise(nonAlternance = mean(turn),performance = mean(correct)) %>%
  ggplot(mapping=aes(x=nonAlternance,y=performance)) +
  geom_point() +
  geom_text(aes(label=mouse),hjust=-0.1,vjust=0.5) +
  xlim(0.2,0.8)
```

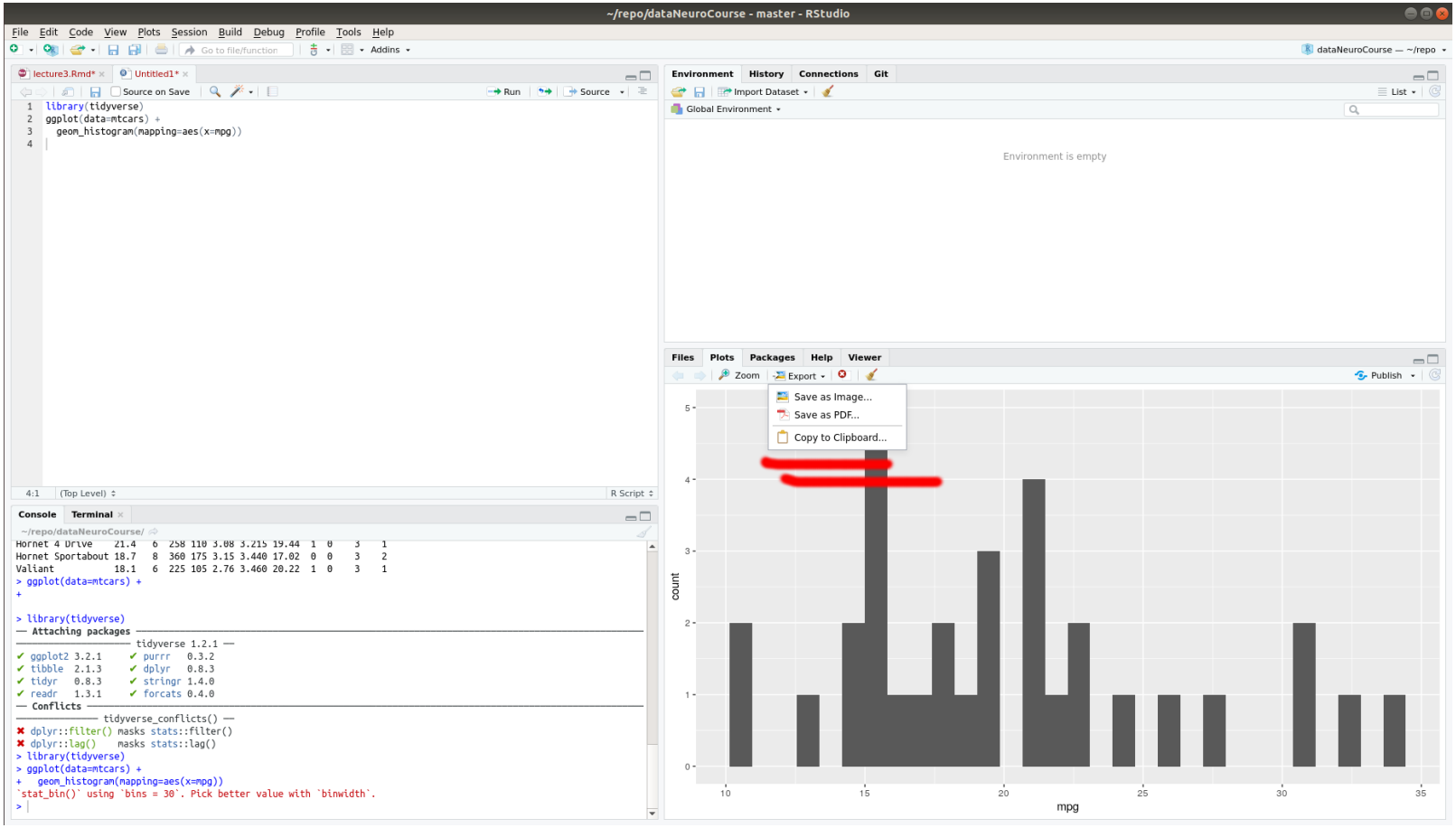


dplyr and ggplot

You are now able to use dplyr and ggplot to extract meaningful information from a dataset!!!

Time to share these nice graphs (export).

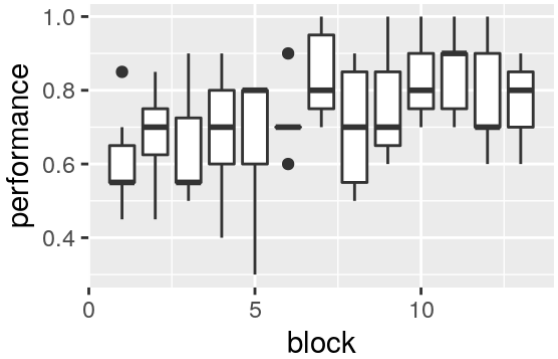
Saving a graph



Saving a graph

`ggsave()` saves the latest plot that was displayed

```
df1 %>%
  ggplot() +
  geom_boxplot(mapping=aes(x=block,y=performance,group=block))
```



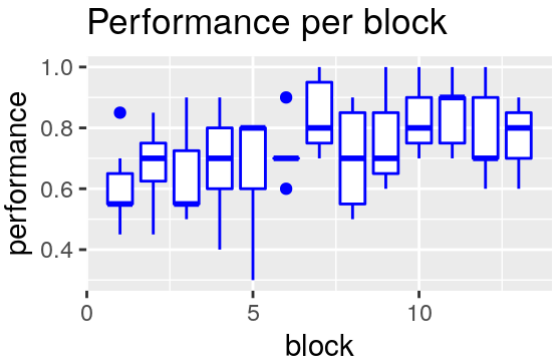

```
myFileName="/home/kevin/Downloads/myPlot.pdf"
ggsave(filename = myFileName,
       device = "pdf", units = "cm",
       width = 10, height = 10)
```

Plot with several graphs

- Store graphs as variables

```
p1<- df1 %>%
  ggplot() +
  geom_boxplot(mapping=aes(x=block,y=performance,group=block),color="blue")+
  ggtitle("Performance per block")


p1
```



Plot with several graphs

Let's store several graphs in variables p1, p2, p3 and p4.

```
p1 <- df %>%
  group_by(mouse, block) %>%
  summarise(nTrials = n(), performance = mean(correct)) %>%
  filter(nTrials == 10) %>%
  ggplot () +
    geom_histogram(mapping = aes(x = performance), binwidth = 0.01) +
    xlab("Performance (prob.)") +
    ylab("Count")
```

```
p2 <- df %>% group_by(mouse,block) %>%
  summarise(performance = mean(correct)*100) %>%
  group_by(block) %>%
  summarise(global_performance = mean(performance)) %>%
  ggplot() +
  geom_line(mapping=aes(x=block,y=global_performance),color="blue")+
  ylim(40,100) +
  xlab("Block") +
  ylab("Performance (%)")+
  geom_hline(mapping = aes(yintercept=50),linetype="dashed")
```

```
p3 <-df %>% group_by(mouse,block) %>%
  summarise(performance = mean(correct)*100) %>%
  ggplot() +
    geom_point(mapping=aes(x=block,y=performance)) +
    geom_line(mapping=aes(x=block,y=performance),color = "blue") +
    ylab("Performance (%)") +
    xlab("Block") +
    geom_hline(mapping = aes(yintercept=50),linetype="dashed") +
    facet_wrap(~mouse)
```

```
p4 <- df %>% group_by(mouse,block) %>%
  summarise(performance = mean(correct)) %>%
  ggplot() +
  geom_boxplot(mapping=aes(x=block,y=performance*100,group=block),color="blue") +
  ylab("Performance (%)") +
  xlab("Block")
```

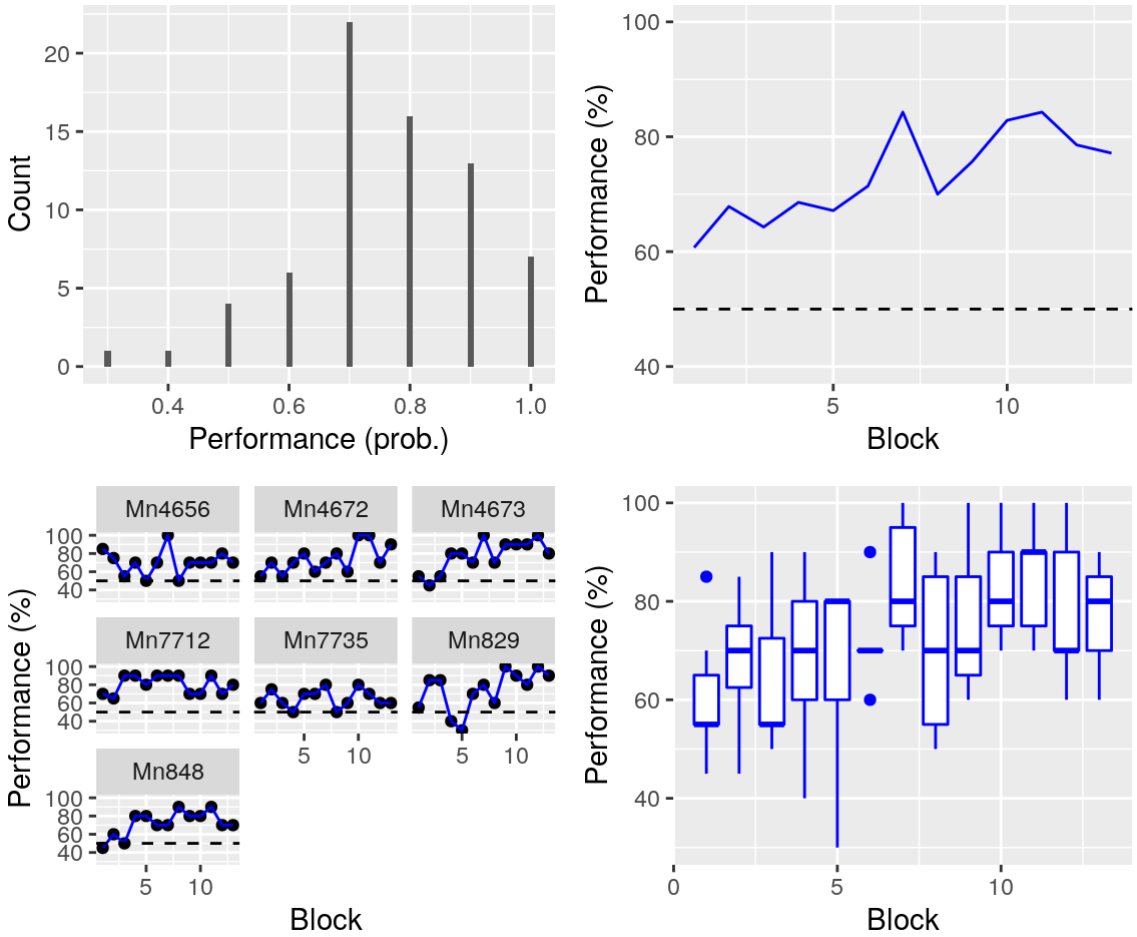
Now put these graphs together.

```
#install.packages("gridExtra")
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
grid.arrange(p1, p2, p3, p4, ncol=2)
```



Use `pdf()` and `dev.off()` to save your new creation.

```
myFileName="/home/kevin/Downloads/allMyPlots.pdf"
pdf(file=myFileName,paper = "a4") # open the file
all_ps <- grid.arrange(p1, p2, p3, p4, ncol=2) # write
dev.off() # close the file
```

```
## png
##   2
```

I usually add the final touch to the graphs using Inkscape.

git: install

Make sure you save any R script open in RStudio.

Install git on your computer

- Windows and Mac: (http://git-scm.com/downloads)
- linux: `sudo apt-get install git-core`

Example for Windows : Download for Windows, Run executable, license next, default location, will install git bash. Use Git and optional Unix tools from the Windows Command Prompt, 3 x next with default

Set up RStudio

1. Go to Global Options (from the Tools menu)
2. Click Git/SVN
3. Click Enable version control interface for RStudio projects

For more information

Create a repository on GitHub.

1. Create an account at GitHub and verify your email
2. Log in
3. Click New repository
4. Set repository name: myNotesDataScience, set to private, check Initialize this repository with a README
5. Click Create repository

Create a project with RStudio from your repository

1. Copy the urs of your repository from the github web site. It ends with `.git`
2. Open RStudio, New Project…, Version Control, Git,
3. Enter the url and set directory.
4. Create project.

Your first commit and push

1. Click on commit.
2. Make sure `myNotesDataScience.Rproj` is staged.
3. Set commit message to `my first commit`
4. Click commit
5. Click on the up arrow to push your commit to the online repository.
6. Refresh your web browser to see `myNotesDataScience.Rproj` online.

Next steps with git…

All you will have to do most of the time.

1. Save the file with your R code in your project directory.
2. Stage the file
3. Commit
4. Push

Relational data with dplyr

In most project, you need to work with several tables.

Relations are defined between a pair of tables.

Relational data with dplyr

Let's assign a genotype to each mouse in a new data frame.

```
dfGeno<-tibble(mouse=c("Mn4656","Mn848","Mn4672","Mn4673",
                       "Mn7712","Mn7735","Mn829"),
          genotype=c("wt","wt","wt","wt",
                     "ko","ko","ko"))
dfGeno
```

```
## # A tibble: 7 x 2
##   mouse  genotype
##   <chr>  <chr>
## 1 Mn4656 wt
## 2 Mn848  wt
## 3 Mn4672 wt
## 4 Mn4673 wt
## 5 Mn7712 ko
## 6 Mn7735 ko
## 7 Mn829  ko
```

How is `df` related to `dfGeno`?

```
colnames(df)
```

```
## [1] "mouse"     "date"      "injection" "block"     "trialNo"   "sample"
## [7] "choice"    "correct"   "turn"
```

```
colnames(dfGeno)
```

```
## [1] "mouse"    "genotype"
```

```
colnames(dfGeno)[colnames(dfGeno) %in% colnames(df)]
```

```
## [1] "mouse"
```

`mouse` is a **key**, a variable that connect a pair of tables.

- A **primary key** uniquely identifies an observation in its table.

```
dfGeno %>%
  count(mouse)
```

```
## # A tibble: 7 x 2
##   mouse      n
##   <chr>  <int>
## 1 Mn4656     1
## 2 Mn4672     1
## 3 Mn4673     1
## 4 Mn7712     1
## 5 Mn7735     1
## 6 Mn829      1
## 7 Mn848      1
```

- A **foreign key** uniquely identifies an observation in *another* table.

```
df %>%
  count(mouse)
```

```
## # A tibble: 7 x 2
##   mouse      n
##   <chr>  <int>
## 1 Mn4656   160
## 2 Mn4672   160
## 3 Mn4673   160
## 4 Mn7712   160
## 5 Mn7735   160
## 6 Mn829    160
## 7 Mn848    160
```

Mutating joins

- It first matches observation by their keys.
- Then copies across variables from one table to the other.

```
df<- df %>%
  left_join(dfGeno,by="mouse") # match with mouse
view(df)
```

We now have an additional variable (genotype) in df.

For more information: Relational data and dplyr

(End - 1)

This is all for today.

Time to use `git` to `commit` and `push` your changes to your online repository.