Intro to R agenda

1. Introductions + goals/R experience
2. Starting from scratch
    a. Overview of scripted data science workflows
    b. R basics
3. More advanced skills
    a. Data frames
    b. Plotting examples
4. Scheduling future sessions

TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Introduction to R

Kelly Pierce, Scalable Computational Intelligence
kpierce@tacc.utexas.edu

With thanks to David Walling and Chris Ramos for contributing materials to these lessons

Scripting languages in data science

- Analysis scripts are a self-documenting reference for your workflow
  - promote increased transparency in methods
  - make methods easier to share

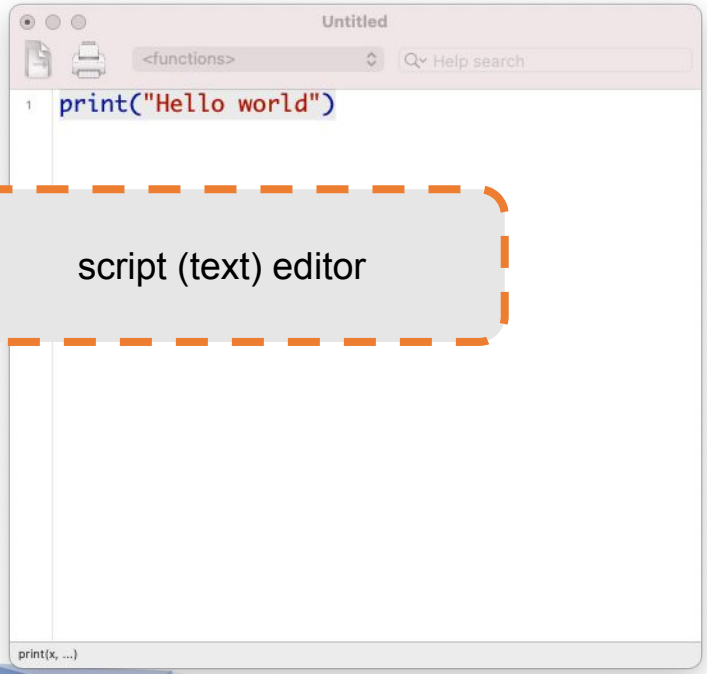
- Open source tools have active user communities
  - more analytic options from community-developed packages
  - users have the ability to create or contribute to packages

Scripted data science workflows

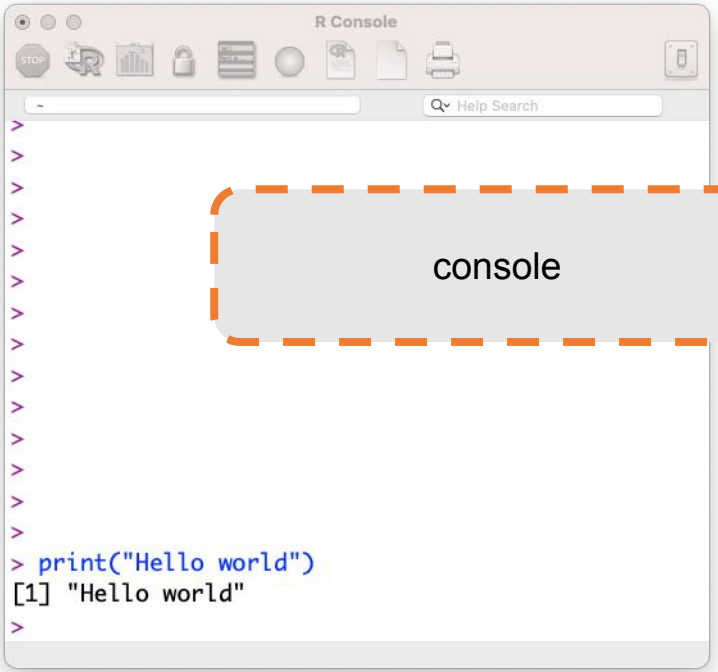
1. Question formulation
2. Data collection
3. Data cleaning and exploration
4. Data analysis
5. Analysis validation
6. Results reporting (dashboard, publication, etc.)

scripted component

Scripted data science in R

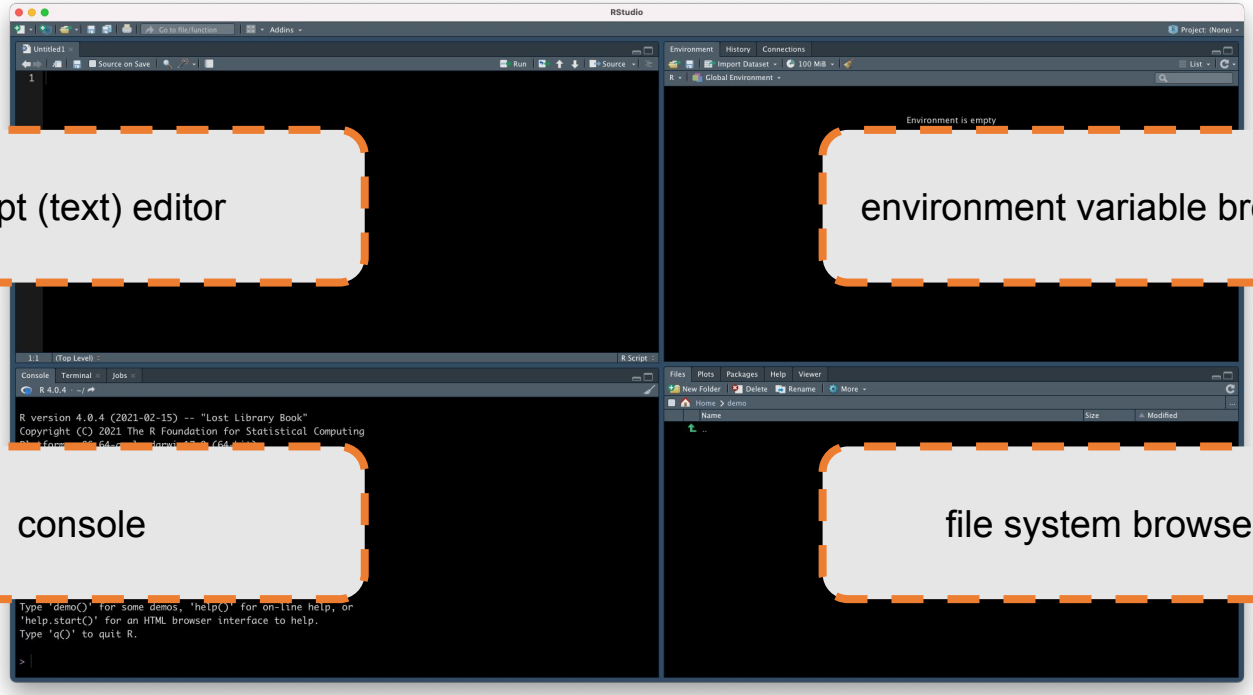


script (text) editor



console

Scripted data science in RStudio



script (text) editor

environment variable browser

console

file system browser

RStudio is an “integrated development environment” (IDE)

- interactive code development
- run R scripts
- explore local files
- view data files
- view graphical output from R

You must have R installed to run RStudio.

R/RStudio installation
Environment setup

Online alternatives to local installation (for basics only)

1. https://colab.research.google.com/#create=true&language=r
2. https://rdrr.io/snippets/

Hello world!

1. Open RStudio
2. Create a new script

   File > New File > R Script
3. Type the following on the first line:

```
print("Hello world")
```

4. Click the  button (while your cursor is on line 1)
5. Inspect the output in the Console pane
6. Type `?print` on line 2 and click 

Basic R syntax

- Math operations
- Logical operators
- Variables

Math operations

You can use R as a calculator for operations such as

+, -, /, *, ^, log, exp, %%

```
log(2.71828)
```

```
## [1] 0.9999993
```

```
exp(1)
```

```
## [1] 2.718282
```

```
log(10, base=10)
```

```
## [1] 1
```

```
log10(10)
```

```
## [1] 1
```

```
print(10 %% 10)
```

```
## [1] 0
```

```
print(10 %% 6)
```

```
## [1] 4
```

Variables

- Variables have different types
- Variables are assigned values
- Variables can be reused

Variables have different types

- Numeric

```
> a <- 800
> b <- 3.14
```

- Character string

```
> c <- "Hello world!"
```

- Logical

```
> d <- FALSE
```

Variables are assigned values

`<-` is the assignment operator

Variables can be reused

Variables can be used in math operations

```
> a <- 800
> b <- 3.14
```

```
> result <- a * b
> print(result)
[1] 2512
```

Variables can also be used in string operations, function calls, etc.

More variable assignment examples

Multiple values can be assigned to a single variable.
The `c()` function combines values into vectors:

```
my_vector <- c(1, 1, 2, 3, 5, 8)
print(my_vector)
```

```
## [1] 1 1 2 3 5 8
```

The `seq()` function makes a sequence:

```
my_sequence <- seq(30, 20, -1)
print(my_sequence)
```

```
##  [1] 30 29 28 27 26 25 24 23 22 21 20
```

Variable assignments can be overwritten:

```
> a <- 4
> a
[1] 4
> a <- 40
> a
[1] 40
```

Working with basic data structures

- Vectors
- Matrices
- Dataframes

Selecting elements from vectors

- Elements in vectors can be referenced by index number.
- The first element in a vector is in “index 1”.

```
my_vector <- c(1, 1, 2, 3, 5, 8)
```

```
print(my_vector[1])
```

```
## [1] 1
```

```
print(my_vector[4])
```

```
## [1] 3
```

Math operations on vectors

- Element-wise operations

```
my_vector*3
```

```
## [1]  3  3  6  9 15 24
```

- Aggregation operations

```
sum(my_vector)
```

```
## [1] 20
```

Lists

- Iterable, named, and ordered (allowing positional indexes)

```
phonebook <- list(name="Jenny", number="867-5309")
print(phonebook$name)
```

```
## [1] "Jenny"
```

```
print(phonebook[1])
```

```
## $name
## [1] "Jenny"
```

Matrices

- Iterable and positionally indexed by [row, column]

```
A = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Getting help

Two built-in ways to bring up help documents:

```
help("<function name>")
?<function name>
```

Exercise #1: Basic Data Structures

A. Create a vector of integers from 1 to 2022 and sum the result.

B. Create a 3x3 matrix of i^3 for i=1 to 9. Sum the third row.

Flow control

- Loops
- Conditional statements

For-loops are used to iterate over values

Every-day iteration examples

- Grocery shopping: “**For** every item on my list, find the item and add it to the cart.”
- Netflix: “**For** every episode of my favorite show, watch the episode.”

For-loop syntax

```
for(i in 1:10){
  print(i*i)
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81
## [1] 100
```

While-loops iterate until a pre-specified condition is met

Every-day iteration examples

- Weather: “**While** it is raining, use an umbrella.”
- Netflix: “**While** there are unwatched episodes of my favorite show, watch the next episode.”

While-loop syntax

```
i <- 1
while(i < 6){
  print(i)
  i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Infinite loops never met the termination criterion

```
j <- 1
while(j < 6){
  print(j)
  j <- j - 1
}
```

If/Else conditional statements

Every-day conditional examples

- Weather: “**If** it is raining, use an umbrella.”
- Netflix: “**If** there are unwatched episodes of my favorite show, watch the next episode.”

(“else” can be implied)

If/Else conditional statements

```
bigger_than_breadbox <- FALSE
if(bigger_than_breadbox == TRUE){
  print('The item bigger than breadbox')
}else{
  print('The item is not bigger than a breadbox')
}
```

```
## [1] "The item is not bigger than a breadbox"
```

If/Else If/Else conditional statements

```
object <- 'plastic'

if(object == 'vegetable'){
  print('Object is a vegetable')
}else if(object == 'animal'){
  print('Object is an animal')
}else if(object == 'mineral'){
  print('Object is a mineral!')
}else{
  print('Object must be something else.')
}
```

```
## [1] "Object must be something else."
```

Exercise #2: Conditionals and Loops

A. Assign a positive numeric value to variable `x`, and write an if/else statement that will multiply `x` by 10 if `x` is less than 100 or by 3 if `x` is greater than 100. Assign the value to a new variable.

B. Write a code snippet that prints integers 1 to 25, and prints “fizz” if the integer is divisible by 3, “buzz” if the integer is divisible by 5, and “fizzbuzz” if the integer is divisible by both 3 and 5.

Exercise #3: Modifying Vectors

A. Create a vector with 10 even numbers using the **`seq()`** function, then create a new vector with 10 odd numbers by modifying the vector.

B. Use a for-loop to combine your two vectors from part A. Can you devise a way to sort the values in ascending order?

Dataframes

- overview
- loading R packages
- inspecting data
- selecting data
- summarizing data

Dataframe overview

- [row, column] indexing like matrices
- reference columns by name as **`df$column_name`**
- built-in **`data.frame`** variable type
  - no external dependencies
  - widely used
  - unexpected handling of categorical data
  - more difficult to subset and slice*
- **`tbl_df`** from “tidyverse” package
  - requires an additional package
  - popular but less common than data.frame
  - easy to subset and slice*

*my subjective opinion

Built-in example data: Motor Trends car dataset ("mtcars")

- load "tidyverse" to get access to the **`tbl_df`** type
- load the built-in dataset
- put the row names in a new column
- convert the data to a **`tbl_df`**
- inspect the first few lines of the data with **`head()`** (**`tail()`** prints the last few lines)

```
library('tidyverse')
```

```
data("mtcars")
mtcars <- rownames_to_column(mtcars, var="car_name")
mtcars <- tibble(mtcars)
```

```
head(mtcars)
```

```
## # A tibble: 6 x 12
##   car_name       mpg   cyl  disp    hp  drat    wt  qsec    vs
##   <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Mazda RX4     21       6   160   110  3.9   2.62  16.5     0
## 2 Mazda RX4 W~  21       6   160   110  3.9   2.88  17.0     0
## 3 Datsun 710    22.8     4   108    93  3.85  2.32  18.6     1
```

Other ways to inspect dataframes

- print the column names

```
names(mtcars)
```

```
##  [1] "car_name" "mpg"      "cyl"      "disp"     "hp"       "drat"
##  [7] "wt"       "qsec"     "vs"       "am"       "gear"     "carb"
```

- view the dimensions (row, column)

```
dim(mtcars)
```

```
## [1] 32 12
```

- select all values in a single column

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Other ways to inspect dataframes

- print the mean of a single column

```
print(mean(mtcars$mpg))
```

```
## [1] 20.09062
```

- print the standard deviation of a single column

```
print(sd(mtcars$mpg))
```

```
## [1] 6.026948
```

Build operation sequences with %>%

The **%>%** operator (also known as a “pipe”) allows you to build sequences of operations.

Here we use **%>%** to select two columns from **mtcars**

```
mtcars %>% dplyr::select(mpg, disp)
```

use the **select()** function from the **dplyr** package, which is loaded as part of the **tidyverse**

```
## # A tibble: 32 x 2
##      mpg  disp
##    <dbl> <dbl>
##  1  21    160
##  2  21    160
##  3  22.8  108
##  4  21.4  258
##  5  18.7  360
##  6  18.1  225
##  7  14.3  360
##  8  24.4  147.
##  9  22.8  141.
## 10  19.2  168.
## # ... with 22 more rows
```

Select and summarize numeric columns

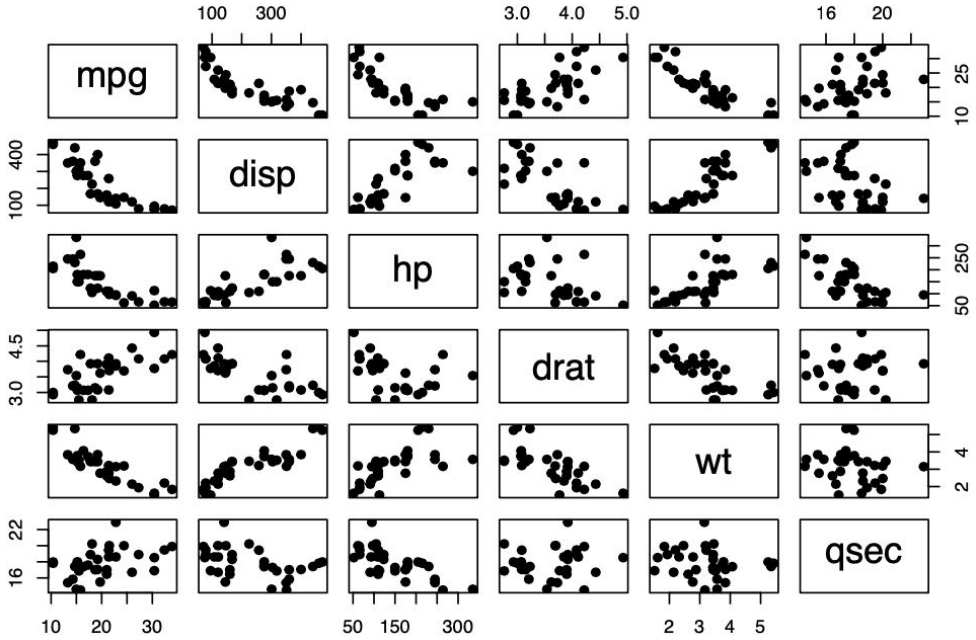
- `%>%` and `dplyr::select()` to grab columns
- `summary()` detects column type and summarizes accordingly

```
numeric_mtcars <- mtcars %>% dplyr::select(car_name, mpg, disp, hp, drat, wt, qsec)
summary(numeric_mtcars)
```

```
##    car_name              mpg             disp             hp
##  Length:32          Min.   :10.40   Min.   : 71.1   Min.   : 52.0
##  Class :character   1st Qu.:15.43   1st Qu.:120.8   1st Qu.: 96.5
##  Mode  :character   Median :19.20   Median :196.3   Median :123.0
##                     Mean   :20.09   Mean   :230.7   Mean   :146.7
##                     3rd Qu.:22.80   3rd Qu.:326.0   3rd Qu.:180.0
##                     Max.   :33.90   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec
##  Min.   :2.760   Min.   :1.513   Min.   :14.50
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89
##  Median :3.695   Median :3.325   Median :17.71
##  Mean   :3.597   Mean   :3.217   Mean   :17.85
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90
##  Max.   :4.930   Max.   :5.424   Max.   :22.90
```

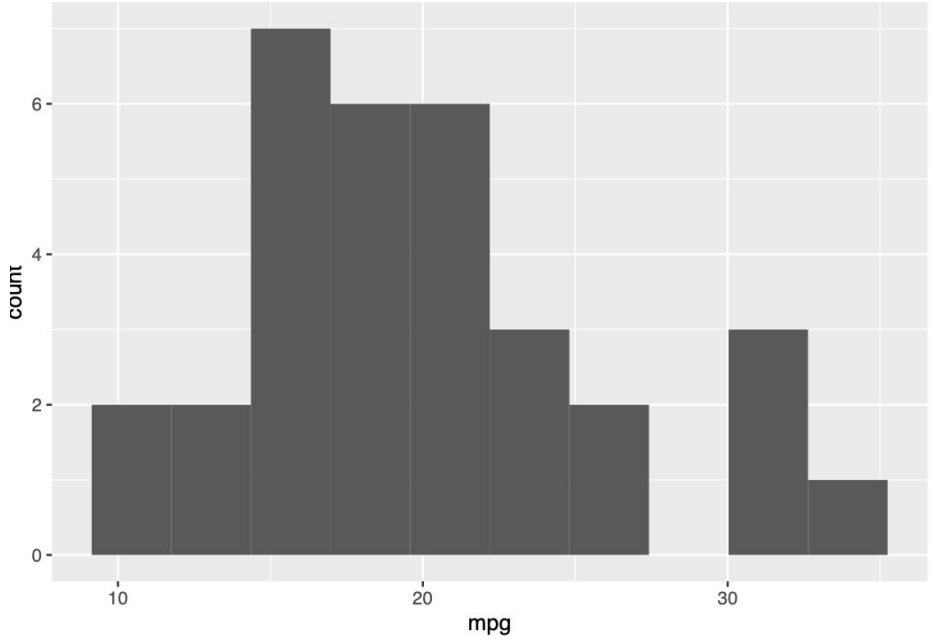
Pairwise scatter plots

```
numeric_mtcars %>% dplyr::select(-car_name) %>% pairs(pch=19)
```



Histograms

```
ggplot(data=mtcars, aes(x=mpg)) + geom_histogram(bins=10)
```



Reshaping data

Wide Format

Team	Points	Assists	Rebounds
A	88	12	22
B	91	17	28
C	99	24	30
D	94	28	31

Each value is unique in first column

Long Format

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

The values in the first column repeat

Which format is the `mtcars` data?

image source: https://www.statology.org/long-vs-wide-data/

Pivot wide data into long format

- **`pivot_longer()`** takes the columns to pivot (all but **`car_name`**), the name of the new grouping column, and the name of the new values column
- see **`?pivot_longer()`** for more details about calling this function

```
numeric_mtcars_long <- numeric_mtcars %>%
  pivot_longer(!car_name, names_to="variable", values_to="value")
head(numeric_mtcars_long)
```

```
## # A tibble: 6 x 3
##   car_name  variable  value
##   <chr>     <chr>     <dbl>
## 1 Mazda RX4 mpg        21
## 2 Mazda RX4 disp      160
## 3 Mazda RX4 hp        110
## 4 Mazda RX4 drat        3.9
## 5 Mazda RX4 wt          2.62
## 6 Mazda RX4 qsec       16.5
```

Facet plots for long format data

- Long format data is useful for plotting different subsets of data automatically – plotting functions scan the grouping column and subset the data internally
- **`ggplot()`** is a plotting library (also in the **`tidyverse`**) that builds plots in layers

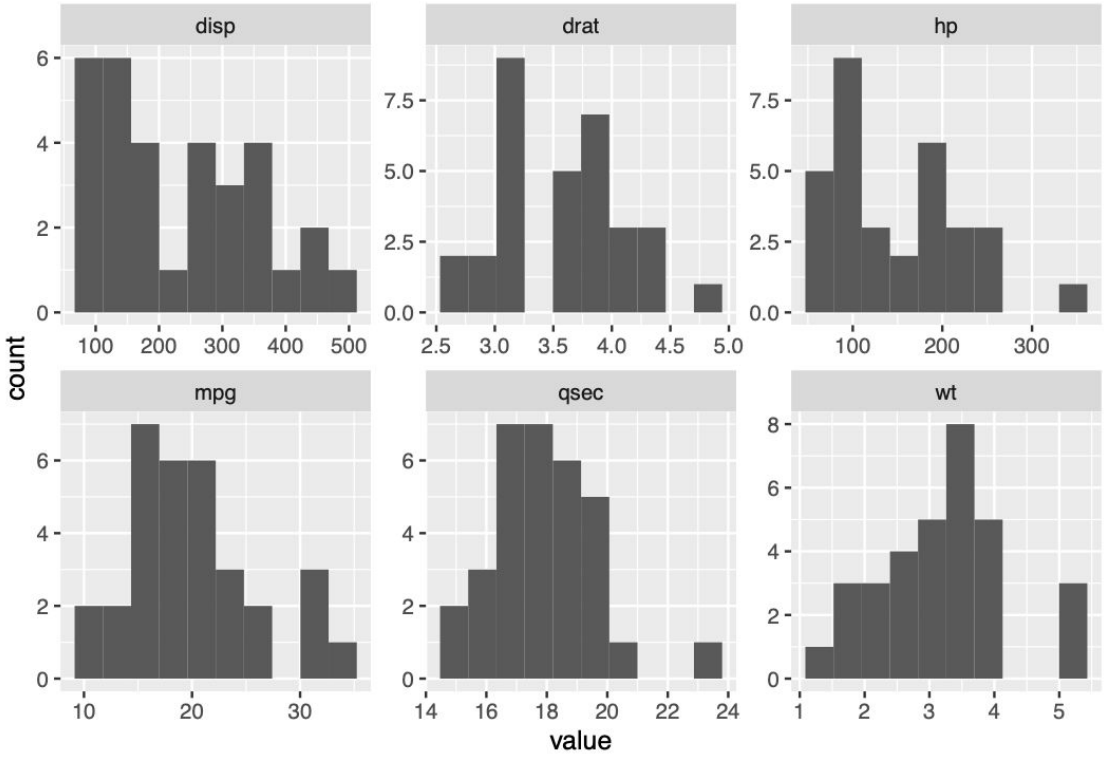
Facet plots for long format data

- build the first layer with **`numeric_mtcars_long`** data, plotting the “value” column
- add a histogram layer with 10 bins per histogram
- make a separate plot facet for each grouping variable in the **`variable`** column

```
ggplot(data=numeric_mtcars_long, aes(value)) +
  geom_histogram(bins=10) +
  facet_wrap(~variable, scales="free")
```

```
## # A tibble: 6 x 3
##   car_name  variable  value
##   <chr>     <chr>     <dbl>
## 1 Mazda RX4 mpg        21
## 2 Mazda RX4 disp      160
## 3 Mazda RX4 hp        110
## 4 Mazda RX4 drat        3.9
## 5 Mazda RX4 wt          2.62
## 6 Mazda RX4 qsec       16.5
```

Facet plots for long format data



Exercise #4: Working with dataframes

A. Load the iris dataset and convert it to a **`tbl_df`**
B. Make a summary of the dataset that includes descriptive statistics. Which descriptive stats do you think are most appropriate for these data?
C. Make a pairwise scatterplot
D. Pivot the iris data to long format and use the **`facet_wrap`** histogram. Change the **`bins`** argument to get an appropriate value.

Review

1. Starting from scratch
    a. Variable assignment and variable types
    b. Mathematical operations
    c. Data structures
    d. Flow control
    e. Getting help
2. More advanced skills
    a. Data frames and data summarization
    b. Basic plots
    c. Advanced plots with ggplot

Topic	Date
R Basics + environment setup	2/4/2022
Intro to spatial data	
Working with US Census Bureau data	
Geocoding data	
Data cleaning and decoding	
Machine learning basics	
Regression analysis	
Predictive modeling	
Geospatial visualizations	