

---

# LEARNING R PROGRAMMING AND PYTHON TOGETHER

DATA SCIENCE PRIMER FOR DEVELOPERS AND MANAGERS

PIERRE DEBOIS  
JANUARY 9TH 2024

# MY BACKGROUND

- ▶ Marketing Analytics for Small/Medium Businesses (Google Analytics, R Programming)
- ▶ Contributor: CMSWire, VKTR, ITProToday,
- ▶ Adjunct Prof: Trine University



## OVERVIEW

- ▶ Explain R Programming & Python
- ▶ The basics of RStudio (the IDE used for R as well as Python via reticulate)
- ▶ Work With Libraries & Modules functions – Tips for developing models to explore data for business applications
- ▶ The basic programming protocols for importing data
- ▶ Incorporating Data sources and ideas for creating models, from regression to sentiment analysis
- ▶ Data visualizations
- ▶ AI Assistants (ChatGPT ADA, Copilot)
- ▶ Best data model practices

## DATA SCIENCE: THE BASICS

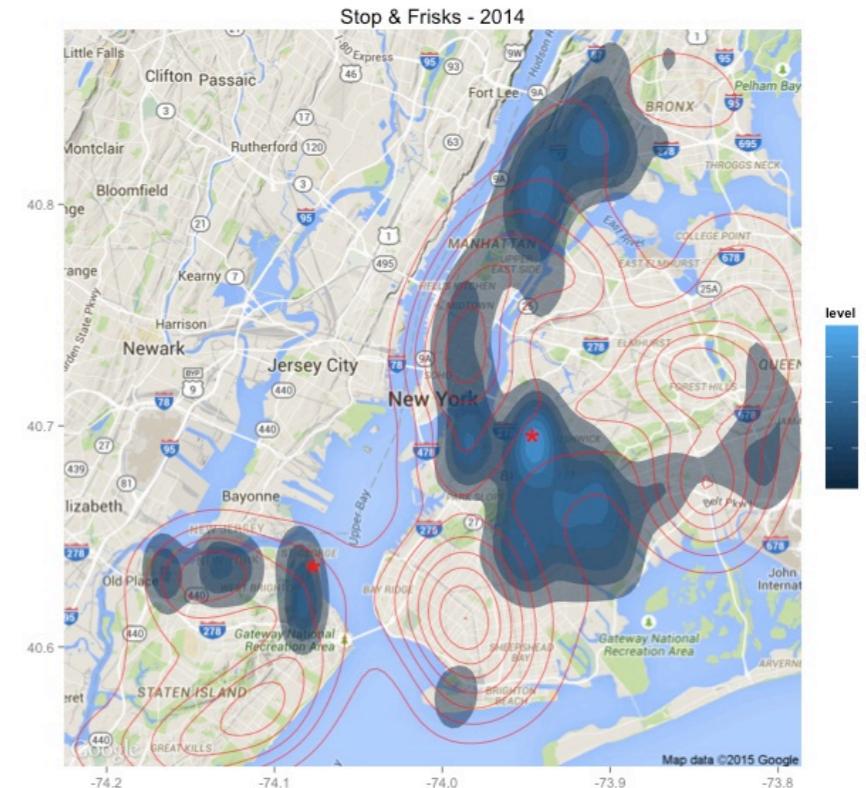
- ▶ Data reflects activity
- ▶ Digital media introduced an association of activity to statistics
- ▶ Data science reveals insights and ideas on human activity associated with digital media measurements
- ▶ Most statistical relationships (time series, regressions, clustering) are first steps for analysis
- ▶ Data science is multidisciplinary - programming, statistics, data visualizations

## DATA MODEL

- ▶ Data is information that underlie products, be it by API, user input, or gathered data from a source.
- ▶ Data Models examine data as features relative to the value of a product, service, or business model

## EXAMPLE USES

- ▶ Sentiment Analysis
- ▶ Mapping Out Public Services & Deployment
- ▶ Stop and Frisk: Spatial Analysis of Racial Discrepancies (Baltimore)
- ▶ New York Building Inspections for Fire Safety
- ▶ Hospital Capacity for COVID-19 cases



## OTHER EXAMPLES

- ▶ How BBC Data Journalist Use R for Data Visualization <https://medium.com/hacks-hackers-london/how-bbc-data-journalists-use-r-for-data-visualization-e29ee74948e1>
- ▶ Drug Crime Density
- ▶ Congressional Density

## WHAT GOES INTO A DATA MODEL

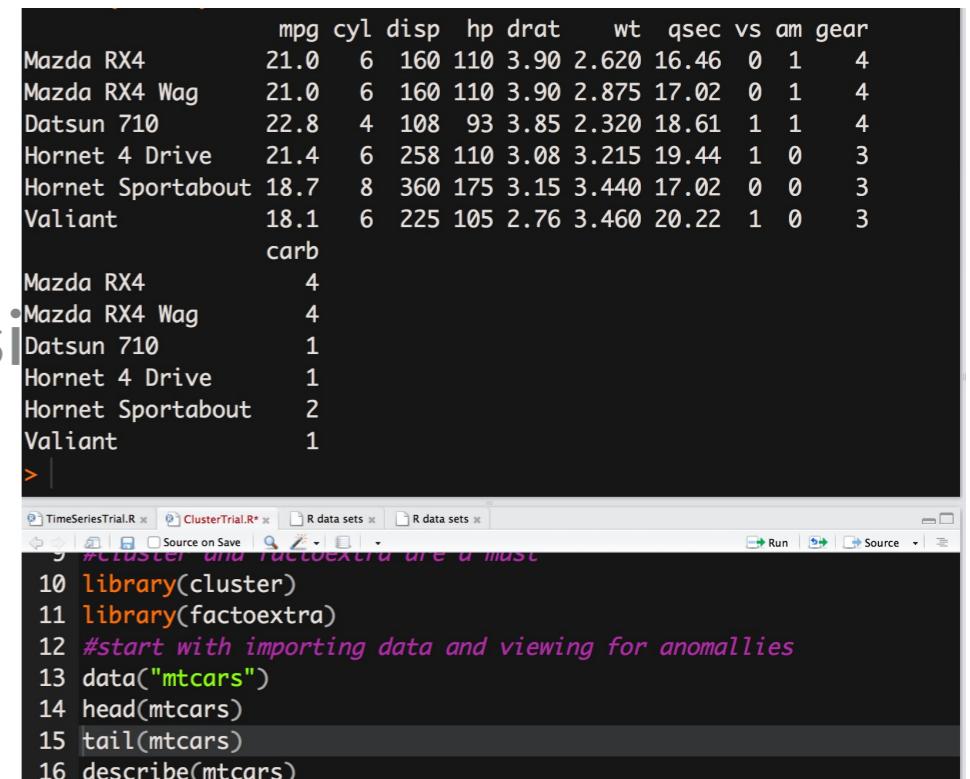
- ▶ Selecting our data
- ▶ Placing data into objects
- ▶ Applying functions (via library/module)
- ▶ Inspect data relationships
- ▶ Develop visualizations and documentation to share information



WHERE TO BEGIN

# R PROGRAMMING

- ▶ Open source programming language w/ command line interpreter based on C and Fortran
- ▶ Allows a wide array of statistical analysis
- ▶ Easier reproduction of large datasets, reading many type of data sources
- ▶ Provides Fast Computation of Vectors
- ▶ Useful for those new to programming but are more familiar with the purpose of the examined data



The screenshot shows the RStudio interface with the following content:

Console output (top half):

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3

carb

Mazda RX4 4  
Mazda RX4 Wag 4  
Datsun 710 1  
Hornet 4 Drive 1  
Hornet Sportabout 2  
Valiant 1

>

Code editor (bottom half):

```
10 library(cluster)
11 library(factoextra)
12 #start with importing data and viewing for anomalies
13 data("mtcars")
14 head(mtcars)
15 tail(mtcars)
16 describe(mtcars)
```

## PYTHON

- ▶ High-level, general-purpose programming language
- ▶ Allows a wide array of computational applications
- ▶ Syntax is like Legos – meant to “snap together” parameters
- ▶ Where R has libraries for nuanced science and data sources, Python has modules for a variety of applications.

## HOW OBJECTS ARE MANAGED - R PROGRAMMING

- ▶ `x=c(1,2,3,4)`
- ▶ `y=c(1,2,3)`
- ▶ `x*y = 5 12 21 20`

## HOW OBJECTS ARE MANAGED - PYTHON

- ▶ `x = np.array([1, 2, 3, 4])`
- ▶ `y = np.array([1, 2, 3])`
- ▶ `result = x * y`
- ▶ `print(result) # Output: [ 1 4 9 12]`

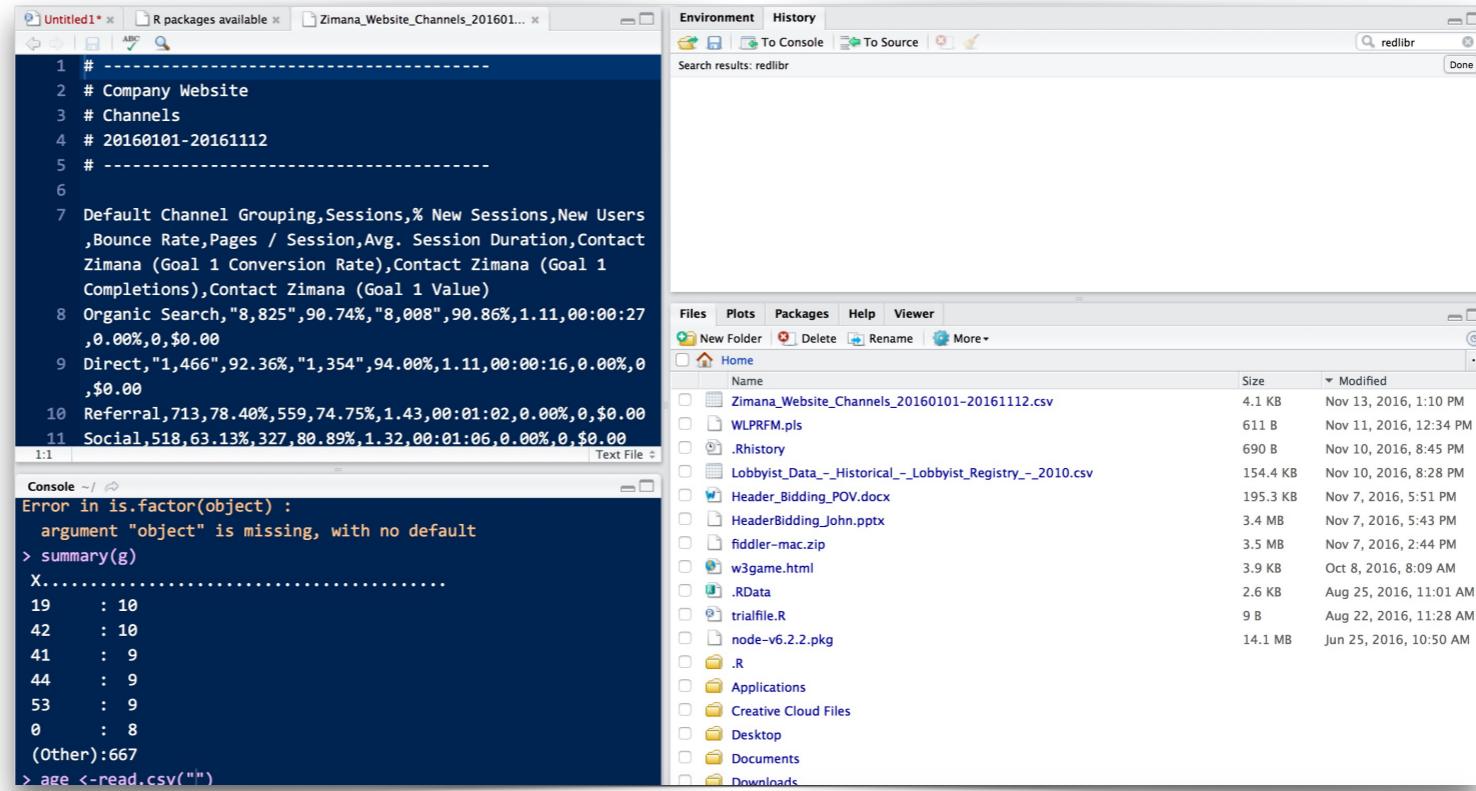
---

# SETTING UP YOUR BRUSH AND CANVAS..AKA YOUR IDE

## OBTAINING R // PYTHON

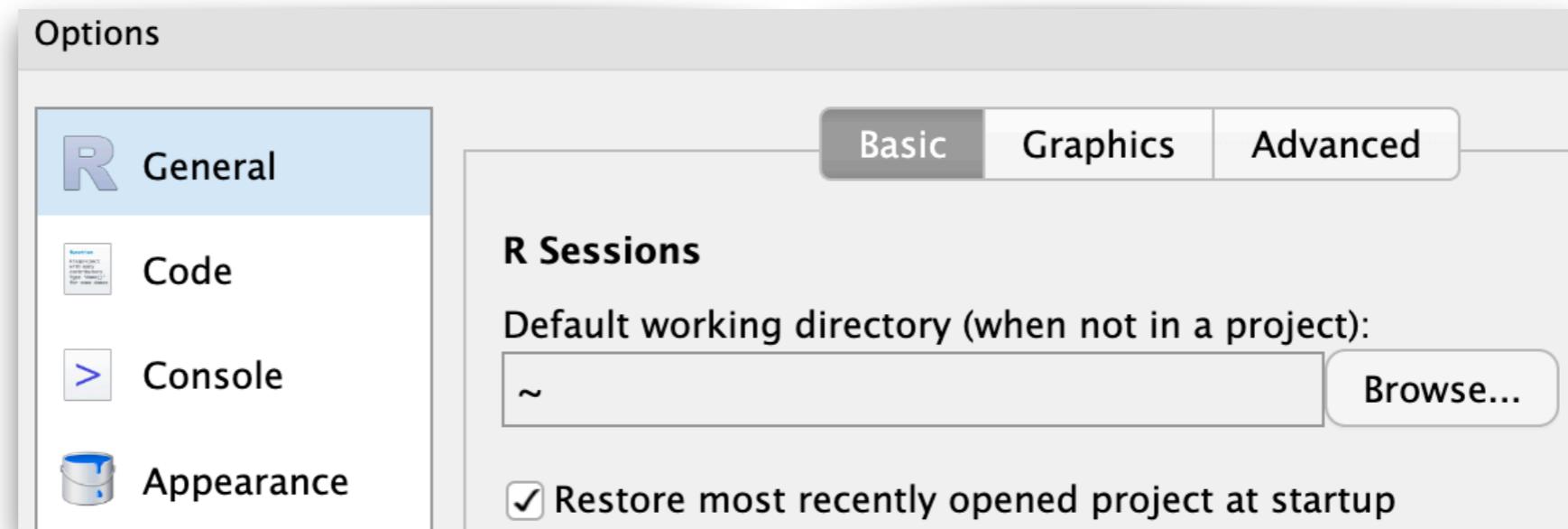
- ▶ First step: download R here: <http://cran.r-project.org/>.
- ▶ Other repositories can provide R
  - ▶ University of Minnesota
  - ▶ Microsoft MRAN <https://mran.microsoft.com/>
- ▶ Python: <https://www.python.org/>
- ▶ Python Package Index - PyPI: <https://pypi.org/>

# R STUDIO



- ▶ Console - runs code and presents results
- ▶ Source - where files can be edited
- ▶ Environment, History
- ▶ Files, Plots, Help

# SET WORKING DIRECTORY



- ▶ Type `getwd()` to verify directory
- ▶ Type the `setwd()` function at the console
  - ▶ eg. `setwd("C:/yourdirectory")`
- ▶ RStudio menu: Tools >> Global Options >> Change Working Directory

## SET A GITHUB DIRECTORY

- ▶ **Optional Step if GitHub Repository is a preferred for version control**
  - ▶ Set up GitHub repository first
  - ▶ Navigate to GitHub settings; copy HTTPS URL
  - ▶ Navigate to New Project, then version control in RStudio
  - ▶ set URL and Project Name

## GITHUB COPILOT

- ▶ Optional Step
- ▶ Evaluates the Code of a Program and Makes Suggestions
- ▶ Appears in the Global Options pane
- ▶ Users can indicate the amount of time passed between the last word typed and when the Copilot kicks in to make
- ▶ Copilot Chat



# LIBRARIES, PACKAGES, FUNCTIONS

# PACKAGES, LIBRARIES, FUNCTIONS



- ▶ Functions = performs calculations on data
- ▶ Packages = collection of functions
- ▶ Libraries = directory of packages

## WHERE TO GET R LIBRARIES

- ▶ CRAN <http://cran.r-project.org/.>
- ▶ MRAN (Microsoft's version on CRAN): <https://mran.microsoft.com/packages/.>
- ▶ GitHub via install-github function from the devtools package, `devtools::install_github("githubaccountname/packagename")`. First install the devtools `install.packages("devtools")`

## HOW TO INSTALL LIBRARIES IN R

- ▶ Installed with the `install.package` function

Type `install.package('packagename')`

- ▶ Or conduct a search in the RStudio
- ▶ Can verify in the packages panel in R Studio
- ▶ To remove type `remove.package('packagename')`

## HOW TO CALL A R LIBRARY

Call `library("libraryname")`

`package::functionname`

`pacman::p_load()`

- ▶ `library(help="packagename")` # reveals the documentation in a package
- ▶ `.libpaths()` # retrieves Library locations
- ▶ `update.packages()`
- ▶ `remove.packages()`

# ONCE A LIBRARY IS INSTALLED

**Package ‘googleAnalyticsR’**

October 20, 2017

**Type** Package  
**Version** 0.4.2  
**Title** Google Analytics API into R  
**Description** R library for interacting with the Google Analytics Reporting API v3 and v4.  
**URL** <http://code.markedmondson.me/googleAnalyticsR/>  
**BugReports** <https://github.com/MarkEdmondson1234/googleAnalyticsR/issues>  
**Depends** R (>= 3.2.0)  
**Imports** assertthat (>= 0.2.0), dplyr (>= 0.7.0), googleAuthR (>= 0.5.1), httr (>= 1.3.1), magrittr (>= 1.5), memoise, purrr (>= 0.2.2), rlang (>= 0.1.0), tidyverse (>= 0.6.3), utils  
**Suggests** bigQueryR (>= 0.3.1), covr, googleCloudStorageR (>= 0.2.0), httpTEST, knitr, miniUI (>= 0.1.1), rmarkdown, shiny (>= 0.13.2)  
**License** MIT + file LICENSE  
**LazyData** TRUE  
**VignetteBuilder** knitr  
**RoxxygenNote** 6.0.1  
**NeedsCompilation** no  
**Author** Mark Edmondson [aut, cre],  
Artem Klevtsov [ctb],  
Johann deBoer [ctb],  
David Watkins [ctb],  
Olivia Brode-Roger [ctb],  
Jas Sohi [ctb]  
**Maintainer** Mark Edmondson <m@sunholo.com>  
**Repository** CRAN  
**Date/Publication** 2017-10-20 20:20:11 UTC

- ▶ Examine the document page for functions you are interested in
- ▶ Make sure package is latest - sometimes out of date status implies no upkeep has been done
- ▶ Can open in browser and use search feature to find libraries

# FUNCTIONS

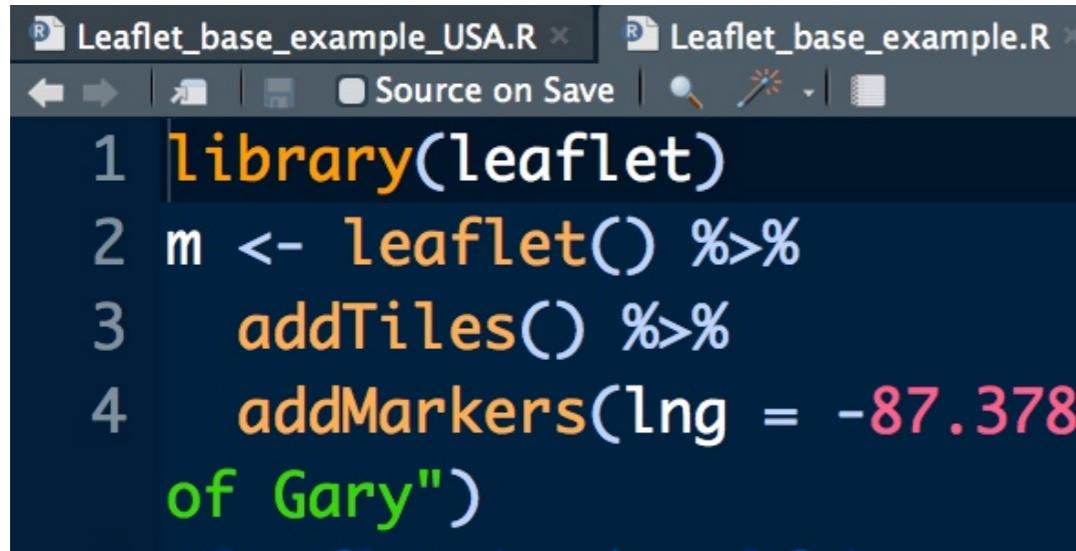
## Package ‘googleAnalyticsR’

October 20, 2017

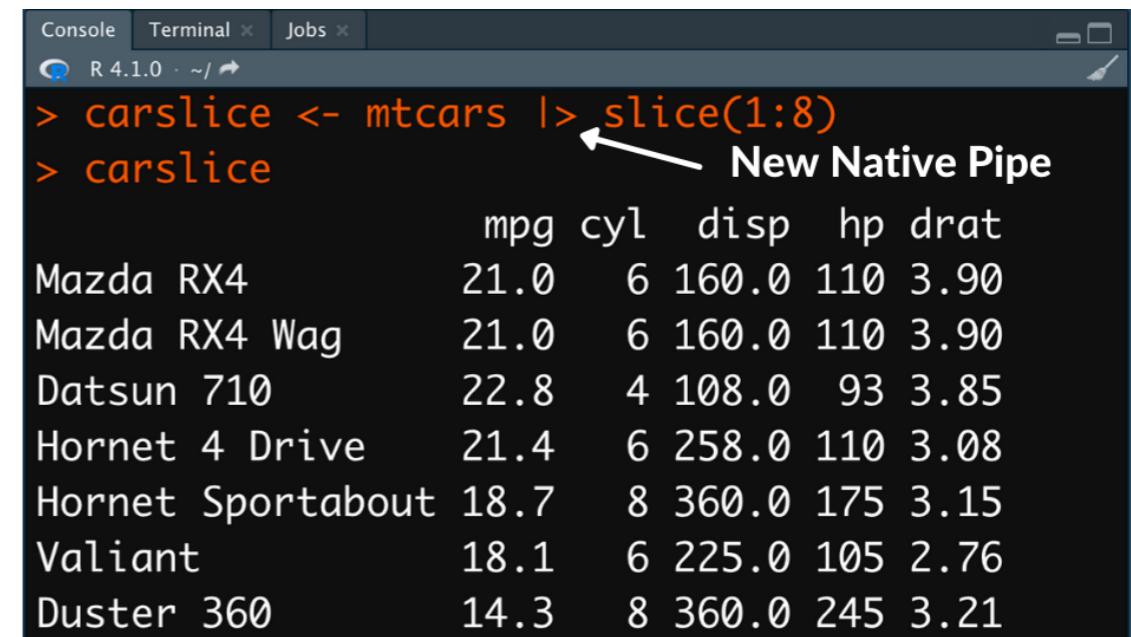
**Type** Package  
**Version** 0.4.2  
**Title** Google Analytics API into R  
**Description** R library for interacting with the Google Analytics Reporting API v3 and v4.  
**URL** <http://code.markedmondson.me/googleAnalyticsR/>  
**BugReports** <https://github.com/MarkEdmondson1234/googleAnalyticsR/issues>  
**Depends** R (>= 3.2.0)  
**Imports** assertthat (>= 0.2.0), dplyr (>= 0.7.0), googleAuthR (>= 0.5.1), httr (>= 1.3.1), magrittr (>= 1.5), memoise, purrr (>= 0.2.2), rlang (>= 0.1.0), tidyverse (>= 0.6.3), utils  
**Suggests** bigQueryR (>= 0.3.1), covr, googleCloudStorageR (>= 0.2.0), httpertest, knitr, miniUI (>= 0.1.1), rmarkdown, shiny (>= 0.13.2)  
**License** MIT + file LICENSE  
**LazyData** TRUE  
**VignetteBuilder** knitr  
**RoxygenNote** 6.0.1  
**NeedsCompilation** no  
**Author** Mark Edmondson [aut, cre],  
Artem Klevtsov [ctb],  
Johann deBoer [ctb],  
David Watkins [ctb],  
Olivia Brode-Roger [ctb],  
Jas Sohi [ctb]  
**Maintainer** Mark Edmondson <m@sunholo.com>  
**Repository** CRAN  
**Date/Publication** 2017-10-20 20:20:11 UTC

- ▶ Functions are intended program tasks
- ▶ syntax `function_name(arg1 = value1, arg2 = value2, ...)`
- ▶ R has some built-in functions; Most functions are called from a library (package)
- ▶ Manual to review functions available

# PIPS



```
library(leaflet)
m <- leaflet() %>%
  addTiles() %>%
  addMarkers(lng = -87.378
             of Gary")
```



```
Console Terminal Jobs
R 4.1.0 ~/ →
> carslice <- mtcars |> slice(1:8)
> carslice
mpg cyl disp hp drat
Mazda RX4 21.0 6 160.0 110 3.90
Mazda RX4 Wag 21.0 6 160.0 110 3.90
Datsun 710 22.8 4 108.0 93 3.85
Hornet 4 Drive 21.4 6 258.0 110 3.08
Hornet Sportabout 18.7 8 360.0 175 3.15
Valiant 18.1 6 225.0 105 2.76
Duster 360 14.3 8 360.0 245 3.21
```

R 4.1 introduces a native pipe, eliminating the need for a library in some instances. Pipes organizes nested functions within R. Usually the magrittr library is used to call pipes. You can read more about pipes in [this Medium post](#).

- ▶ A specific form of infix function
- ▶ Lets you nest formulas within one another
- ▶ Usually comes with the magrittr package
- ▶ Internal Pipes introduced in R 4.1,



SELECTING YOUR STUDIO..AKA  
YOUR ENVIRONMENT

## R ENVIRONMENT

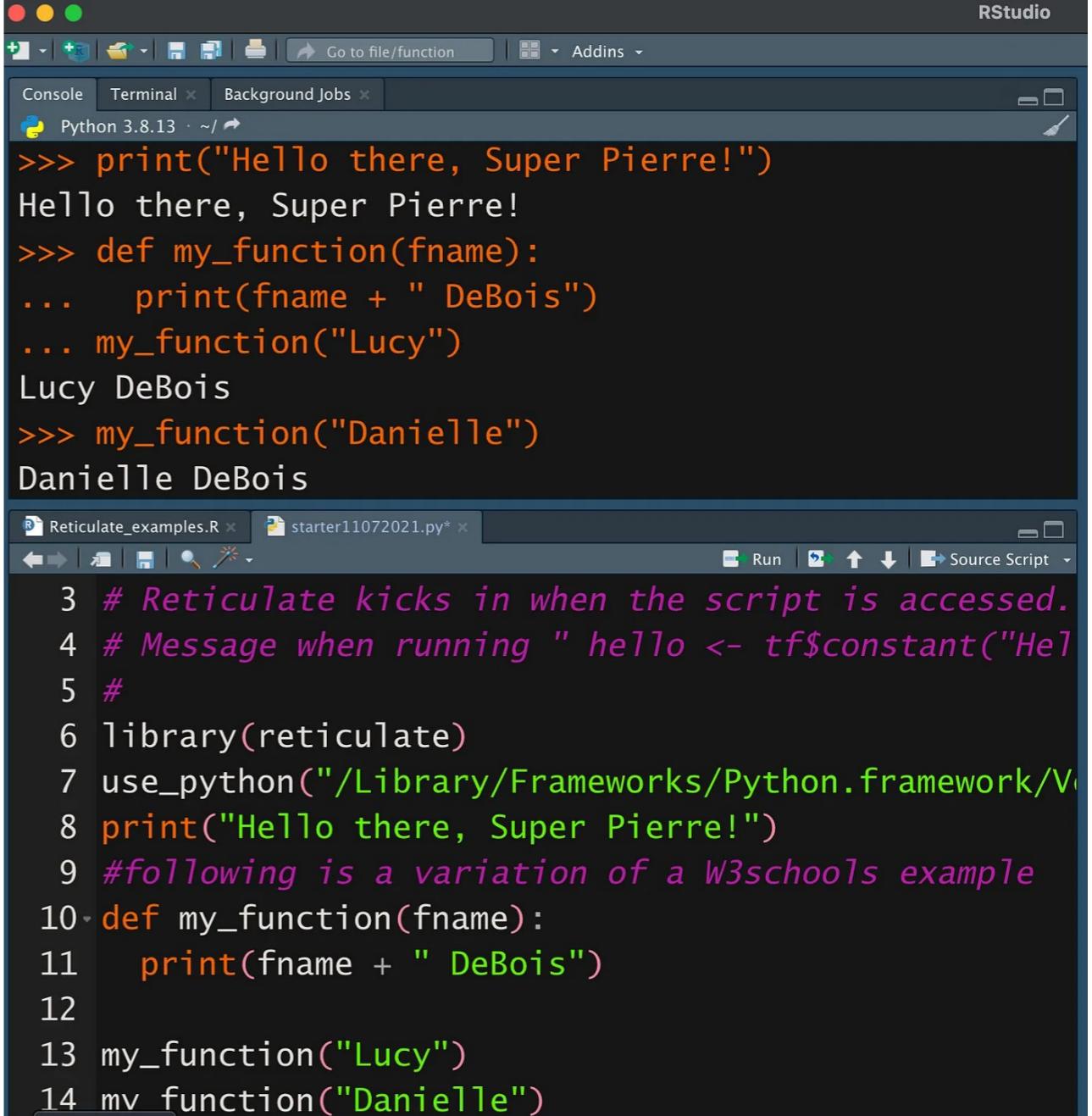
- ▶ A place to store name-object pairs between variable name and object assigned
- ▶ Create a hierarchy
- ▶ Functions operate in a separate environment to perform a calculation

## CLEAN YOUR ENVIRONMENT

- ▶ `ls()` list global objects that exist
- ▶ `ls("package:ggplot2") [1:20]` - first twenty objects in a package
- ▶ `rm()` or `remove()` – choose what objects to remove
- ▶ `rm(list=ls())` – removes all the objects: character (0) confirms empty

# USING RETICULATE

- ▶ Library for running Python or a Python environment
- ▶ To call it `repl_python()`
- ▶ To run a script `py_run_file()`
- ▶ Access an environment
- ▶ To install modules `py_install()`



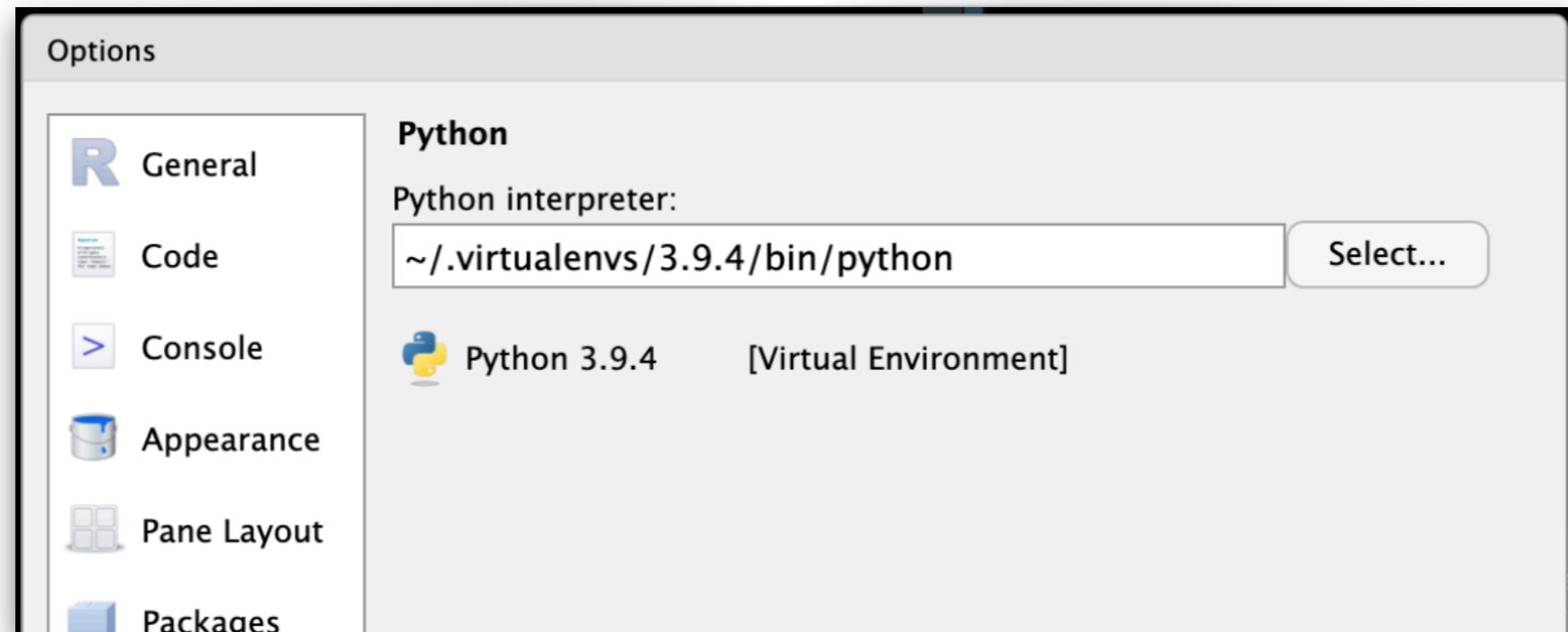
The screenshot shows the RStudio interface. The top panel is the R console, displaying Python code and its output:

```
>>> print("Hello there, Super Pierre!")
Hello there, Super Pierre!
>>> def my_function(fname):
...     print(fname + " DeBois")
... my_function("Lucy")
Lucy DeBois
>>> my_function("Danielle")
Danielle DeBois
```

The bottom panel shows the script editor with two files open: `Reticulate_examples.R` and `starter11072021.py*`. The Python file contains the following code:

```
3 # Reticulate kicks in when the script is accessed.
4 # Message when running " hello <- tf$constant("Hello
5 #
6 library(reticulate)
7 use_python("/Library/Frameworks/Python.framework/V
8 print("Hello there, Super Pierre!")
9 #following is a variation of a W3schools example
10 def my_function(fname):
11     print(fname + " DeBois")
12
13 my_function("Lucy")
14 my_function("Danielle")
```

# VIRTUAL ENVIRONMENTS



- ▶ Designed for separate modules
- ▶ Set up at the terminal
- ▶ RStudio Can Recognize The Environments
- ▶ Reticulate has functions to access environments as well

## VIRTUAL ENVIRONMENT TIPS

- ▶ Consider using a tool like `virtualenvwrapper` for managing virtual environments more efficiently,
  - ▶ provides commands like `mkvirtualenv` and `workon` to streamline creation and activation
- ▶ Keep your virtual environments organized and well-named for easy search identification.
- ▶ Check for outdated packages within activated environments.
  - ▶ `pip list --outdated`
  - ▶ `conda list outdated`

## PIP

Package	Version	Latest	Type
absl-py	1.2.0	2.0.0	wheel
anyio	3.6.1	4.2.0	wheel
argon2-cffi	21.3.0	23.1.0	wheel
asttokens	2.0.8	2.4.1	wheel
attrs	22.1.0	23.2.0	wheel
autopep8	1.6.0	2.0.4	wheel
Babel	2.10.3	2.14.0	wheel
beautifulsoup4	4.11.1	4.12.2	wheel
bleach	5.0.1	6.1.0	wheel
cachetools	5.2.0	5.3.2	wheel
certifi	2022.9.24	2023.11.17	wheel
cffi	1.15.1	1.16.0	wheel
charset-normalizer	2.1.1	3.3.2	wheel
cloudpickle	2.2.0	3.0.0	wheel
contourpy	1.0.5	1.2.0	wheel
cycler	0.11.0	0.12.1	wheel
debugpy	1.6.3	1.8.0	wheel
dm-tree	0.1.7	0.1.8	wheel
executing	1.1.1	2.0.1	wheel
fastjsonschema	2.16.2	2.19.1	wheel
filelock	3.12.4	3.13.1	wheel

- ▶ Can list available modules
- ▶ Can list/enter/exit environments



# CONTAINERS TO HOLD DATA

## CONTAINERS FOR DATA - R

- ▶ Objects to hold data types (classes)
- ▶ Factor
- ▶ Object structure
  - ▶ List - a vector with different classes
  - ▶ Matrix - a table that combines vector & function
  - ▶ Data Frames - an object to store data tables
  - ▶ Data.tables - data frame that speeds up functions

## CONTAINERS FOR DATA – PYTHON

- ▶ Object structure
  - ▶ List - a sequence of elements
  - ▶ Dictionary - key:value pairs
  - ▶ numpy - arrays, dataframe, categoricals

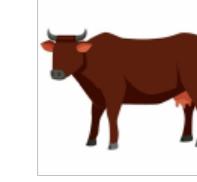
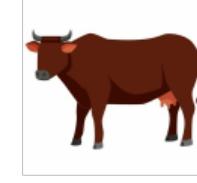
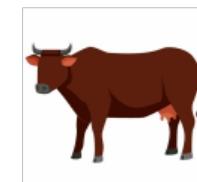
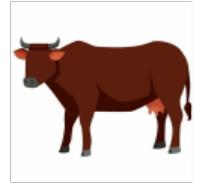
## OBJECTS

```
> x <- 5
```

```
> |
```

- ▶ Objects are variables with assigned values and function
- ▶ Created with the assign operator (<-, <<-, or assign() )
- ▶ Variable name can contain letter, digits, periods, and underscores
- ▶ Variable can start with a letter or period
- ▶ If it starts with a period, no number character can follow
- ▶ Case sensitive // Reserve words can not be used

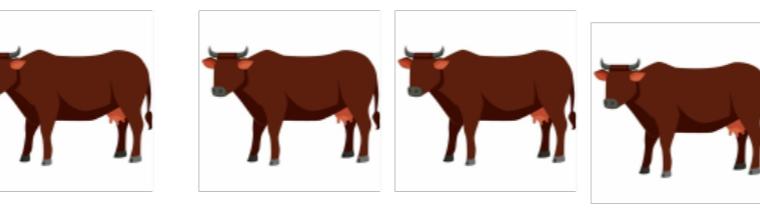
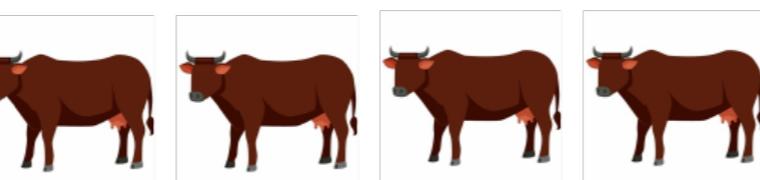
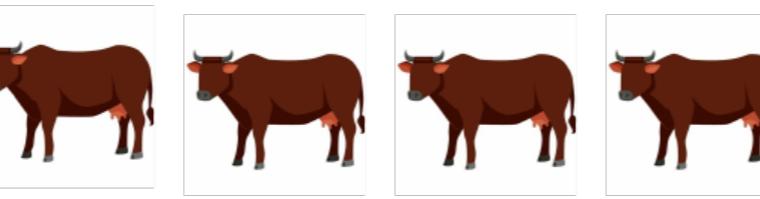
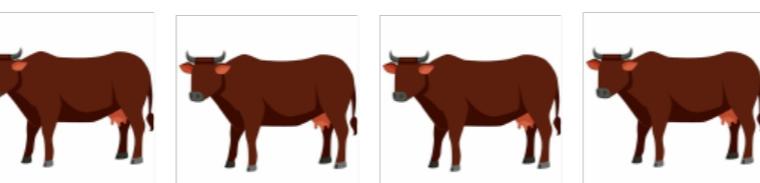
# OBJECT OVERVIEWS

**Vector**

1 column  
1 class

**List**

1 column  
of 1 or  
more  
classes

**Matrix**

2D version of a vector  
(Same Class in Each Column)

**Data Frame /  
Data Table**

Vectors Grouped Together  
(Different Class In Each  
Column)

# ARRAYS

- ▶ Array - c function
  - ▶ example: `z <- c(1,2,3)`
  - ▶ or `z <- c(A=1,B=2,C=3)`

# VECTORS

- ▶ array objects created /w  
**as.vector()** or assign with  
**c(arguments)**
- ▶ example:
- ▶ Return syntax - [x]1 where x is  
a vector, 1 is the first element
- ▶ **length()**
- ▶ **subset()** - retains a subset of a  
vector

```
> n <- c(0.3, 0.9, 9.8)
> class(n)
[1] "numeric"
> o <- c("x","y","z")
> o
[1] "x" "y" "z"
> class(o)
[1] "character"
>
```

# LISTS

```
> x <- list(1,"a", TRUE, 1+4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i

> |
```

- ▶ A type of vector with different classes or object types

# MATRIX

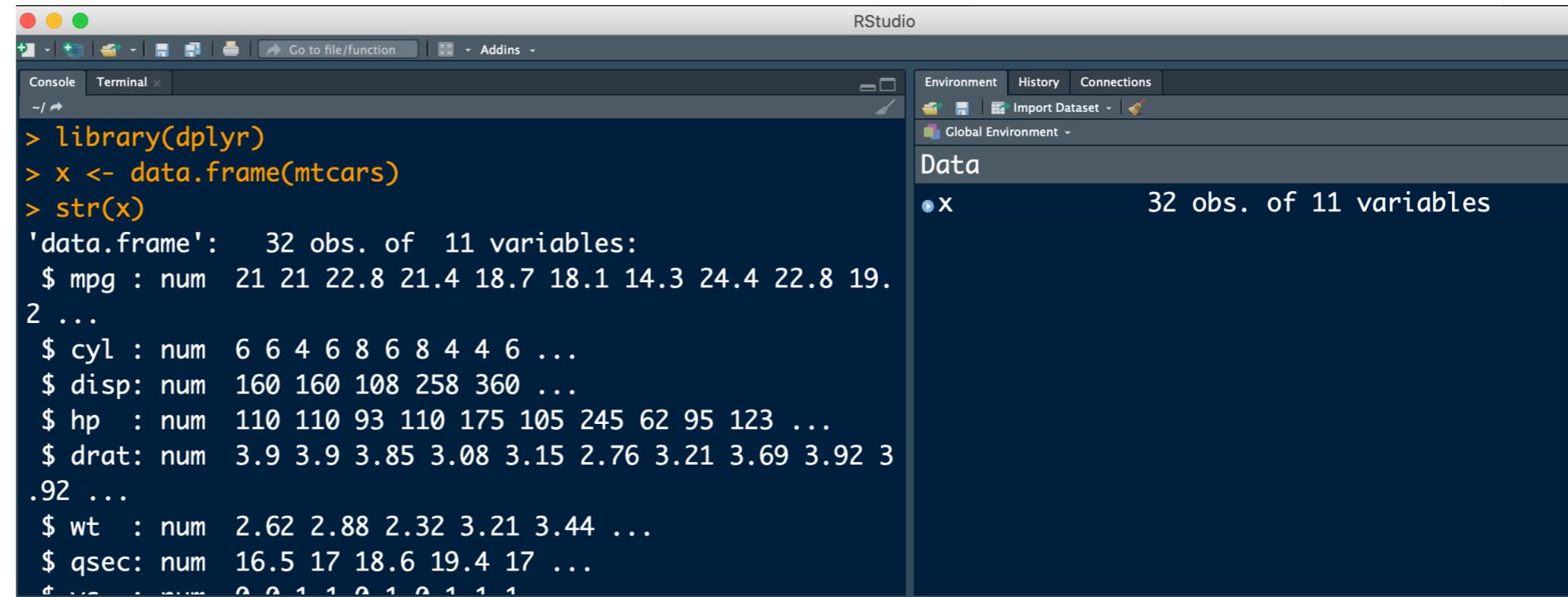
```
> m <- matrix ( 1:16, nrow = 2, ncol = 8)
> m
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    3    5    7    9   11   13   15
[2,]    2    4    6    8   10   12   14   16
```

- ▶ Matrix functions are vectors with a dimension attribute
- ▶ Combines vector & function syntax

## MATRIX EXAMPLE

- ▶ `m <- matrix (c (2,5,17,1,9,13), nrow=2 byrow=TRUE)`
- ▶ Yields matrix 2x3 of dbl type
- ▶ `m <- matrix (c (2,5,17,1,9,13), nrow=2 byrow=FALSE)`
- ▶ `colnames(m) <- c("Column1","Column2")`
- ▶ `rownames(m) <- c("Row1","Row2")`
- ▶ `rbind()` adds new rows, `cbind()` adds new column

# DATA FRAME



The screenshot shows the RStudio interface. In the Console tab, the following R code is run:

```
> library(dplyr)
> x <- data.frame(mtcars)
> str(x)
'data.frame': 32 obs. of 11 variables:
 $ mpg : num 21 21 22.8 21.4 18.7 ...
 $ cyl : num 6 6 4 6 8 ...
 $ disp: num 160 160 108 258 ...
 $ hp : num 110 110 93 110 175 ...
 $ drat: num 3.9 3.9 3.85 3.08 3.15 ...
 $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num 16.5 17 18.6 19.4 17 ...
 $ vs : num 0 0 1 1 0 ...
 $ am : num 1 1 0 1 1 ...
 $ gear: num 4 4 3 3 4 ...'
```

In the Data view on the right, there is one entry named 'x' which is described as '32 obs. of 11 variables'.

- ▶ rectangular objects for storing data tables (kind of array)
- ▶ `n = c(2,3,5)`

`df = data.frame(n,s,b)`

`example.df = c(2,3,4,5)`

- ▶ `as.data.frame =>` confirms a data frame object

## DATA FRAME (CONTINUED)

- ▶ `nrow(mtcars)`
- ▶ `ncol(mtcars)`
- ▶ typing `mtcars[1,2]` calls first row, second column in a table
  - ▶ returns [1] 6
  - ▶ or `mtcars ["mazda Rx4","cyl" ]` also returns [1] 6
- ▶ `Data.table` – data frame extension

## DATA FRAME (CONTINUED)

- ▶ Can add columns but they must be of equal length across the dataframe
- ▶ value type must be the same within list (column)
- ▶ value type can be different across columns

## MATRIX VS DATA FRAMES

- ▶ Matrix solve different problems than data frames. They are meant for
  - ▶ linear transformation
  - ▶ different math language
  - ▶ represents a larger set of information (eigenvectors / values)

## DATA SLICE

```
> iris[20,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
20            5.1        3.8       1.5        0.3  setosa
> |
```

```
> iris[1]
  Sepal.Length
1            5.1
2            4.9
3            4.7
4            4.6
5            5.0
6            5.4
7            4.6
8            5.0
```

- ▶ a slice is a row or column
- ▶ can call by number or column label
- ▶ Data slice can call multiple columns mtcars[c("col1","col2")]

## DATA SLICE FROM DATAFRAMES / DATABLES / MATRIX

- ▶ `subset()`
- ▶ `[ ]` - for subsets from a vector, matrix, or array
- ▶ `[[ ]]` – for extractions
- ▶ \$extraction by name (works the same as `[[ ]]`)

## GETTING USE TO OBJECTS

- ▶ Call up the dataset mtcars or gtcars from the library gt
- ▶ With mtcars, create a data slice of the mpg & hp columns by number and name
- ▶ With gtcars create a data slice of the

## DID YOU GET THE FOLLOWING?

- ▶ `mtcars[c("hp","mpg")]`

```
> mtcars[c("hp", "mpg")]
      hp mpg
Mazda RX4     110 21.0
Mazda RX4 Wag 110 21.0
Datsun 710    93 22.8
Hornet 4 Drive 110 21.4
Hornet Sportabout 175 18.7
Valiant       105 18.1
Duster 360    245 14.3
Merc 240D      62 24.4
Merc 230       95 22.8
Merc 280       123 19.2
Merc 280C      123 17.8
Merc 450SE     180 16.4
Merc 450SL     180 17.3
Merc 450SLC    180 15.2
Cadillac Fleetwood 205 10.4
Lincoln Continental 215 10.4
```



---

# WHAT TO DO TO CAPTURE DATA

## EXAMINE HOW DATA IS STRUCTURED

- ▶ How Is This Data Table Structure Maintained - file? API? URL?
- ▶ What types of data exists within the fields?
- ▶ Does a header exist in the desired table?
- ▶ How can I maintain cleaning standards?

# PRACTICE DATA SETS WITHIN R

The screenshot shows the RStudio interface. The top-left pane is the Console, displaying a license message from R-project.org and a command prompt. The top-right pane is the Environment tab, showing R code related to Twitter OAuth and ggplot2. The bottom-left pane is the Data View tab, listing various datasets available in the datasets package. Some datasets are described with their purpose.

Data sets in package 'datasets':	
AirPassengers	Monthly Airline Passenger Numbers 1949-1960
BJSales	Sales Data with Leading Indicator
BJSales.lead (BJSales)	Sales Data with Leading Indicator
BOD	Biochemical Oxygen Demand
CO2	Carbon Dioxide Uptake in Grass Plants
ChickWeight	Weight versus age of chicks on different diets
DNase	Elisa assay of DNase
EuStockMarkets	Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde	Determination of Formaldehyde
HairEyeColor	Hair and Eye Color of Statistics Students
Harman23.cor	Harman Example 2.3
Harman74.cor	Harman Example 7.4
Indometh	Pharmacokinetics of Indomethacin
InsectSprays	Effectiveness of Insect Sprays
JohnsonJohnson	Quarterly Earnings per Johnson & Johnson Share
LakeHuron	Level of Lake Huron 1875-1972
LifeCycleSavings	Intercountry Life-Cycle Savings Data
Loblolly	Growth of Loblolly pine trees
Nile	Flow of the River Nile
Orange	Growth of Orange Trees
OrchardSprays	Potency of Orchard Sprays
PlantGrowth	Results from an Experiment on Plant Growth
Puromycin	Reaction Velocity of an Enzymatic Reaction
Seatbelts	Road Casualties in Great Britain 1969-84
Theoph	Pharmacokinetics of Theophylline
Titanic	Survival of passengers on the Titanic

- ▶ type **data()** , then type XXX to see available sets
- ▶ Can use a dataset from another library (gt, ml3rdata, torch)
- ▶ You can import a practice dataset from a spreadsheet
- ▶ Can even create a library

# REPOSITORIES FOR DATASETS

▶ [ourworldindata.org](http://ourworldindata.org)

▶ Kaggle

▶ [data.gov](http://data.gov)

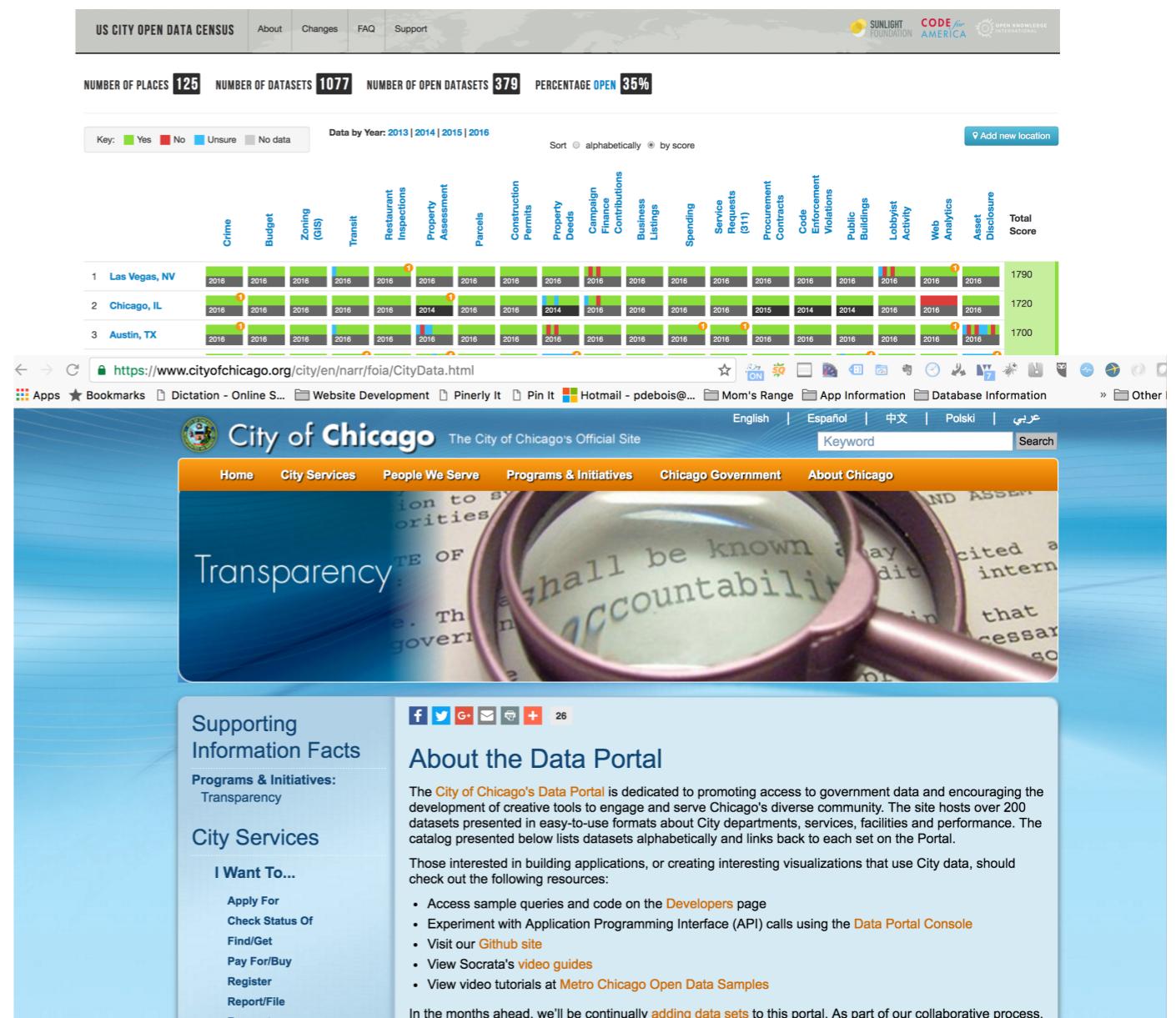
▶ [healthdata.gov](http://healthdata.gov)

▶ World Bank

▶ Pew Institute

▶ Data.world

▶ City Data Portal



## READ FUNCTIONS TO IMPORT DATASHEETS

- ▶ `read_* * = csv,delim,tsv`
- ▶ `read_csv(f,col_name=FALSE or "A", "B", "C")`
- ▶ `readxl` – library for reading Excel files
- ▶ `readr` - reads text files

## READING APIs W/ HTTR

```
 8 query <- "https://pokeapi.co/api/v2/type/3/"  
 9 out <- GET(url=query)  
10 out|
```

- ▶ `httr()` - connects APIs, allows data to be imported into a container
- ▶ `out <- GET(query=list(search = ""))`
- ▶ `content(out)` to view what content appears from the API
- ▶ `header(out)` - headers that appear
- ▶ `str(content(out))`

---

SORT AND ARRANGE DATA

## EXPLORATORY DATA ANALYSIS (EDA)

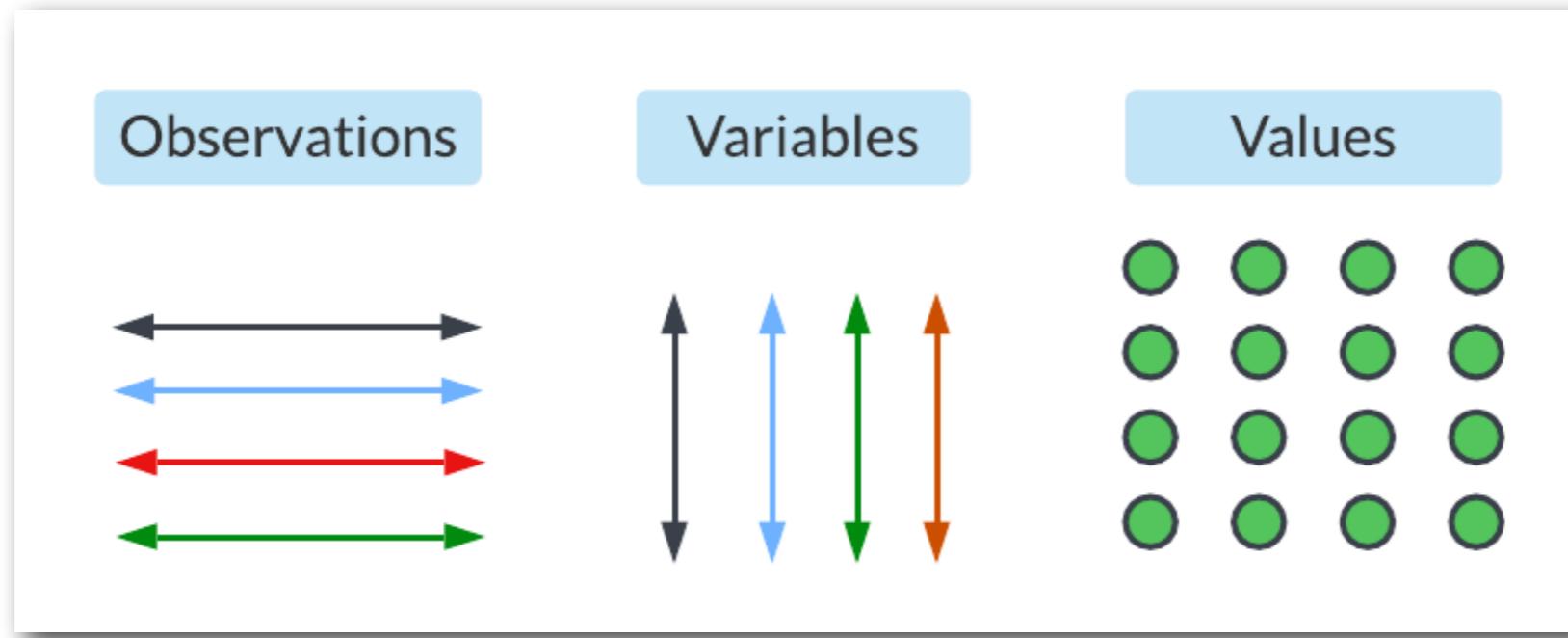
- ▶ Involves reviewing statistical characteristics of data
- ▶ Check observations for NAs, Unusual Values, Extreme Outliers
- ▶ Adjusting Rows and Columns as needed to prep for an analysis
- ▶ Spot any potential data anomalies to avoid feeding damaged data to an advanced data model (APIs)
- ▶ Highlights and documents data parameters to make connections to a planned data model

# EDA CONSTRUCTS A BRIDGE FOR YOUR DATA



- ▶ Data is information that underlie products, be it by API, user input, or gathered data from a source.
- ▶ Data Models examine data as features relative to the value of a product, service, or business model

## EDA PUTS DATA IN A FORMAT TO BE MODELED



- ▶ Explore data to see if it can be a model for regressions, clusterings, and other statistical analysis
- ▶ Feature Engineering reveals how the model information relates to each other, discovering clues for valuable decisions and benefits

## INSPECT THE DATA YOU HAVE

- ▶ `#print mydata variable`
- ▶ `head(mydata, n=10)` first ten rows are displayed
- ▶ `tail(mydata, n=8)` last eight rows are displayed

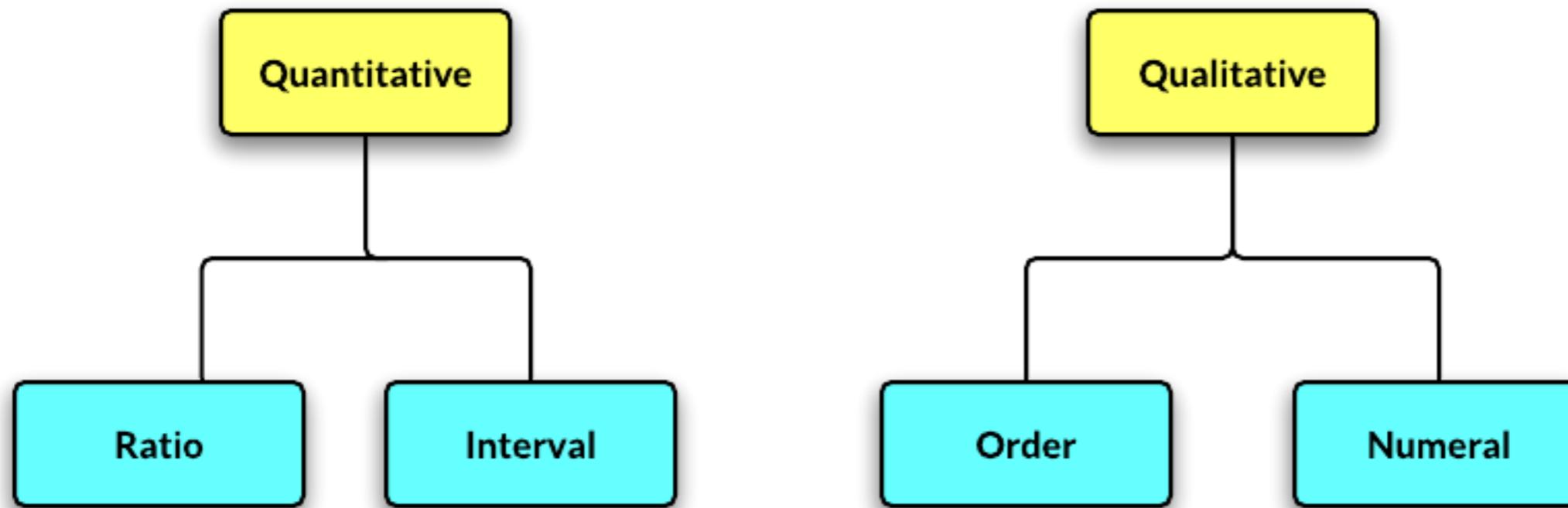
The screenshot shows the RStudio interface with two panes. The top pane displays the first 10 rows of a dataset named 'mydata' using the `head` function. The bottom pane displays the last 8 rows of the same dataset using the `tail` function. The console output shows the command `tail(g)` and the resulting data.

Row	Date	Value
320	10/30/16, 19	
321	10/31/16, 47	
322	11/1/16, 63	
323	11/2/16, 52	
324	11/3/16, 41	
325	11/4/16, 46	
326	11/5/16, 15	
327	11/6/16, 38	
328	11/7/16, 68	
329	11/8/16, 56	
330	11/9/16, 55	
331	11/10/16, 55	
332	11/11/16, 48	
333	11/12/16, 41	
334	,"11,540"	
335		

Row	Date	Value
1:1		
8		8,008
9		90.86%
20		1.11
tail(g)	X.....	
17		11/11/16
18		48
19		11/12/16
20		41
21		
22		11,540
*		

# IDENTIFY DATA TYPES FROM YOUR DATASETS



- ▶ categorical
- ▶ continuous

# INSPECT THE DATA YOU HAVE -PART DEUX

```
> glimpse(a)
Rows: 48,895
Columns: 16
$ id
$ name
$ host_id
$ host_name
$ neighbourhood_group
$ neighbourhood
$ latitude
$ longitude
$ room_type
$ price
$ minimum_nights
$ number_of_reviews
> summary(a)
      id          name        host_id
Min.   : 2539  Length:48895   Min.   : 2438
1st Qu.: 9471945 Class :character 1st Qu.: 7822033
Median  :19677284 Mode  :character Median : 30793816
Mean    :19017143
3rd Qu.:29152178
Max.   :36487245
> ExpData(a, type=1)
                                         Descriptions Value
1                                         Sample size (nrow) 48895
2                                         No. of variables (ncol) 16
3                                         No. of numeric/integer variables 10
4                                         No. of factor variables 0
5                                         No. of text variables 6
6                                         No. of logical variables 0
7                                         No. of identifier variables 1
8                                         No. of date variables 0
9                                         No. of zero variance variables (uniform) 0
10                                        %. of variables having complete cases 75% (12)
11                                        %. of variables having >0% and <50% missing cases 25% (4)
12                                        %. of variables having >=50% and <90% missing cases 0% (0)
13                                        %. of variables having >=90% missing cases 0% (0)
```

- ▶ dplyr:: glimpse() - observations & variables, integer vs character
- ▶ dplyr:: summary() - returns quartiles
- ▶ skimr::skim() - returns number of column type, missing values, empty values, quartiles
- ▶ SmartEDA:: ExpData() - returns descriptions of observations & variables, missing cases

## FINDING MISSING OBSERVATIONS

- ▶ `table(is.na( x))` – determines any missing values
- ▶ `colSums (is.na(x))` -- check which variables are missing
- ▶ `grep(x)` – evaluate a string for pattern matching (from the `stringr` library)
- ▶ `skimr::skim` -- displays missing values & more quantile functions

## CAN ADJUST CHARACTERS USING STRINGR FUNCTIONS

- ▶ `stringr::str_length("a b c") ->[1]3`
- ▶ `stringr::str_sub()` ->access characters
- ▶ `stringr::str_trim()` ->remove whitespace
- ▶ `stringr::str_pad()` ->add whitespace
- ▶ `get()` - evaluate a string

## ANOTHER DESCRIBE FUNCTION

- ▶ psych::describe() function
  - ▶ mean
  - ▶ median
  - ▶ min
  - ▶ max
  - ▶ range
  - ▶ skew
  - ▶ kertosis

# STATS WORK ASSIGNMENT

- ▶ Find the dataset trees, women, cars
- ▶ Create a file to show the min, max, mean, and range  
(library hint: psych)
  - ▶ women: height
  - ▶ trees: girth
  - ▶ mtcars: mpg

## DPLYR PACKAGE

- ▶ Provides simple “verb” functions for common data manipulation tasks
  - ▶ `glimpse()` (like a summarize)
  - ▶ returns #rows, #columns, datatype of columns, first 10 entries
  - ▶ similar to python `df.describe()`

## PANDA DESCRIBE

- ▶ Similar to glimpse() function in R (dplyr)
- ▶ Displays numeric by default
- ▶ non-numeric columns specified by the include='all' argument

## DPLYR PACKAGE - FILTER()

- ▶ filter() - subsets datasets/ a clean syntax
- ▶ can use & or |
- ▶ similar to python df[df['country']=='France']

## DPLYR PACKAGE - ARRANGE()

- ▶ `arrange()` - shuffles columns in ascending/  
descending order
- ▶ similar to python `df.sort_values()`

## DPLYR PACKAGE - SELECT()

- ▶ `select()` - snapshot of columns you want for an original dataset
- ▶ similar to python

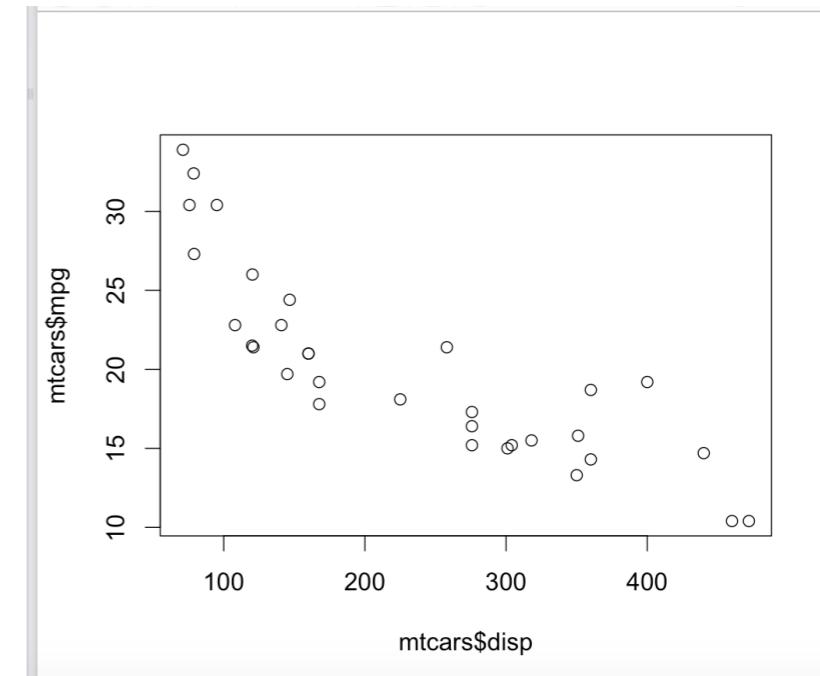
## OTHER DPLYR FUNCTIONS

- ▶ `mutate()` (and `transmute()`)
- ▶ `summarise()` - sum, mean, median, standard dev, variation
- ▶ `group_by()`
- ▶ `inner_join()`
- ▶ `count()` - number of appearances in a column

# VISUALIZATION

## STANDARD R PLOT

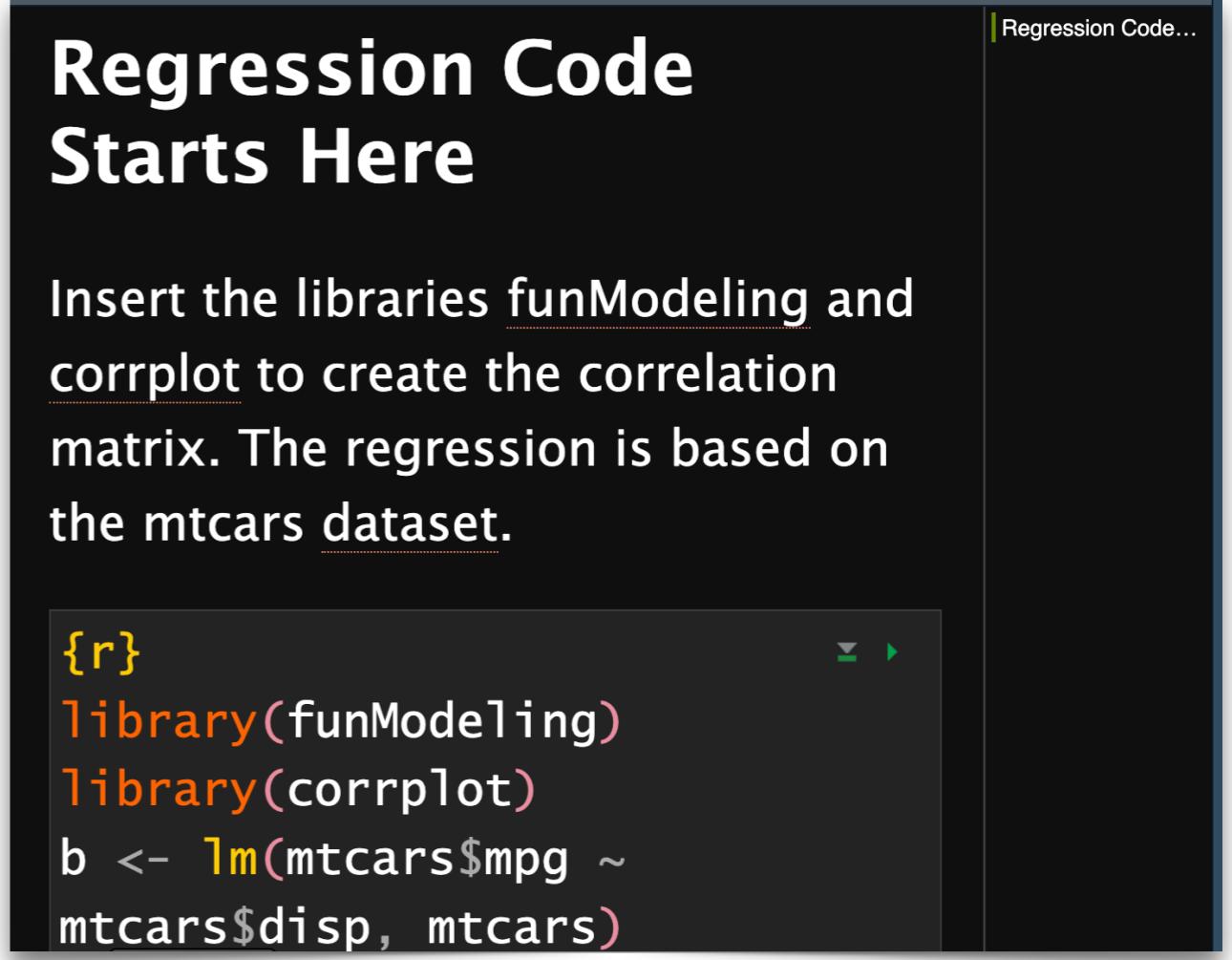
```
▶ plot(mtcars$disp, mtcars$mpg)
```



- ▶ `plot(file, type= ) #simple line graph`
- ▶ `barplot( ) # simple bar chart`
- ▶ `boxplot( ) # simple box plot`

# USE DOCUMENTS TO SHARE INSIGHTS

- ▶ In-program Documents
- ▶ DataExplorer::create\_report()
- ▶ Quarto (R & Python)
- ▶ Display Multiple Graphs
- ▶ Patchwork
- ▶ pairs() // ggpairs()
- ▶ Summarize What Data Has Been Updated
- ▶ Identify What Forecasts Can Be Developed



Regression Code Starts Here

Insert the libraries `funModeling` and `corrplot` to create the correlation matrix. The regression is based on the `mtcars` dataset.

```
{r}  
library(funModeling)  
library(corrplot)  
b <- lm(mtcars$mpg ~ mtcars$disp, mtcars)
```

# GGPLOT



- ▶ Popular graphing library by Hadley Wickham
- ▶ Geom is a function to graph data field
- ▶ aes is the aesthetics (usually data source in basic form, but also control appearance choices)
- ▶ Each returns a layer to form the graph shape.
- ▶ Requires tidy data frame

## GGPLOT - GEOM

- ▶ Discrete and continuous data,
- ▶ one variable, two variables, three variables
- ▶ geom values include "point", "smooth", "boxplot", "line", "histogram", "density", "bar", and "jitter".

## PYTHON VISUALIZATION

- ▶ Matplotlib
- ▶ seaborn (built on matplotlib)
  - ▶ pair plot function – sns.pairplot().

# MODELS



## REGRESSION

- ▶ Examining statistical relationship among data columns:  $y = ax + b$
- ▶ Simple example:  $y=\text{total cost}$   $a = \text{variable cost}$ ,  $b = \text{fixed cost}$
- ▶ Linear or Nonlinear
- ▶ Usually calculate MSPE, MSAE, or R squared for correlation
- ▶ Balance of data: too few - biased estimate, while too many leads to less precision

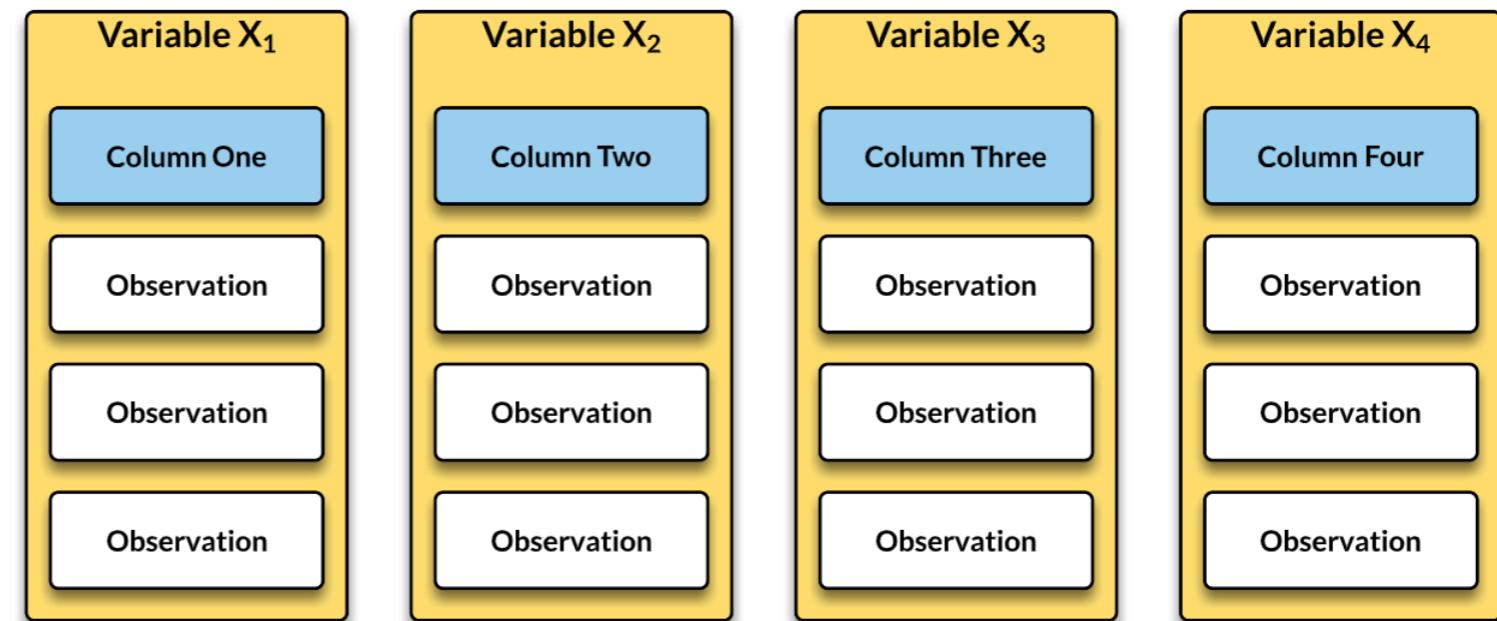
## INSPECT CORRELATION AMONG THE DATA COLUMNS

- ▶ `cor()` function `cor(yourdataframe, method="Pearson")`
- ▶ `Hmisc::rcorr()`
- ▶ data frame must be used in the function
- ▶ Pearson is the default method; Spearman can also be used
- ▶ linear correlation between -1 and +1, with 0 as no correlation

# REGRESSION

$$y = B_0 + B_1 x_1 + B_2 x_2 + B_3 x_3 + B_4 x_4 + \dots$$

B = coefficients  
x = independent variable  
y = dependent variable



- ▶ Regression data <- data.frame(x1, x2, x3, x4)
- ▶ lm (y ~ ., data = Regression\_data)
- ▶ Period means "to include all the independent variables"

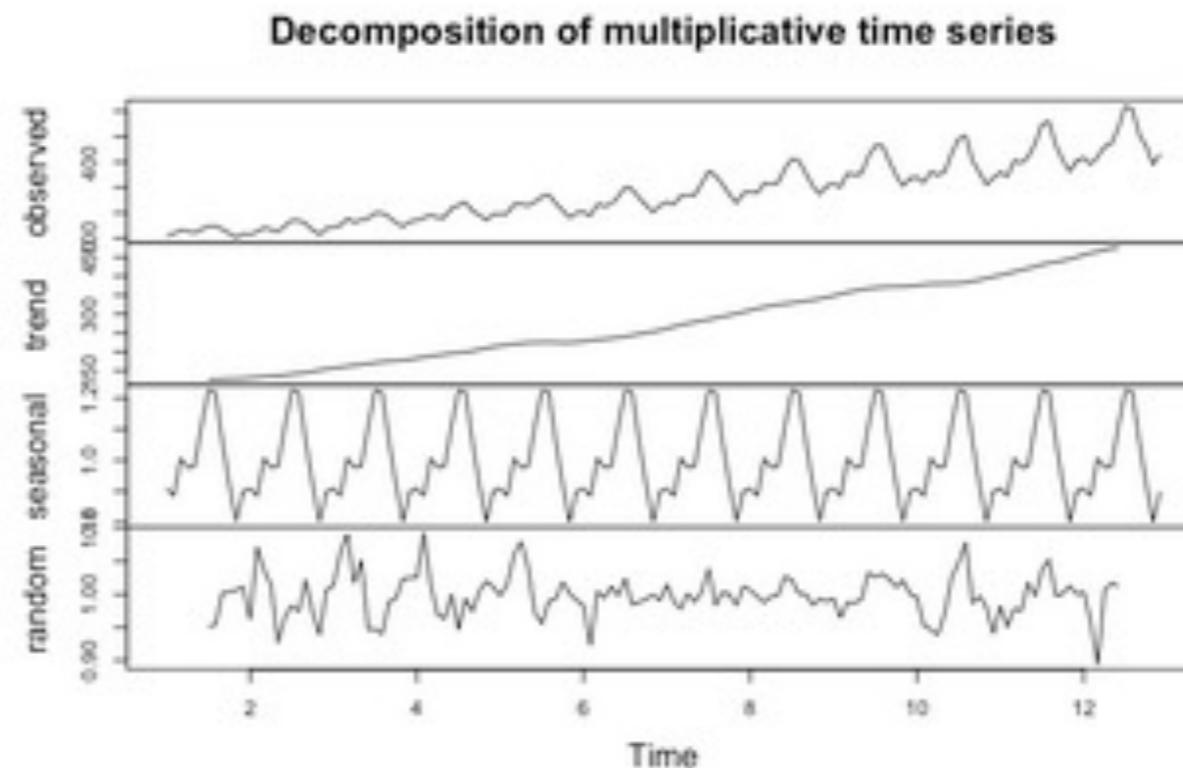
## CREATE A REGRESSION EXAMPLE

- ▶ Using gtcars in the gt library, create a regression with price being the dependent variable
- ▶ Using

## WHAT A REGRESSION MODEL CAN LOOK LIKE

```
carmodel <- lm(msrp ~ hp + trq +  
mpg_c + mpg_h + bdy_style, data =  
lmcars)
```

# TIME SERIES: DECOMPOSITION



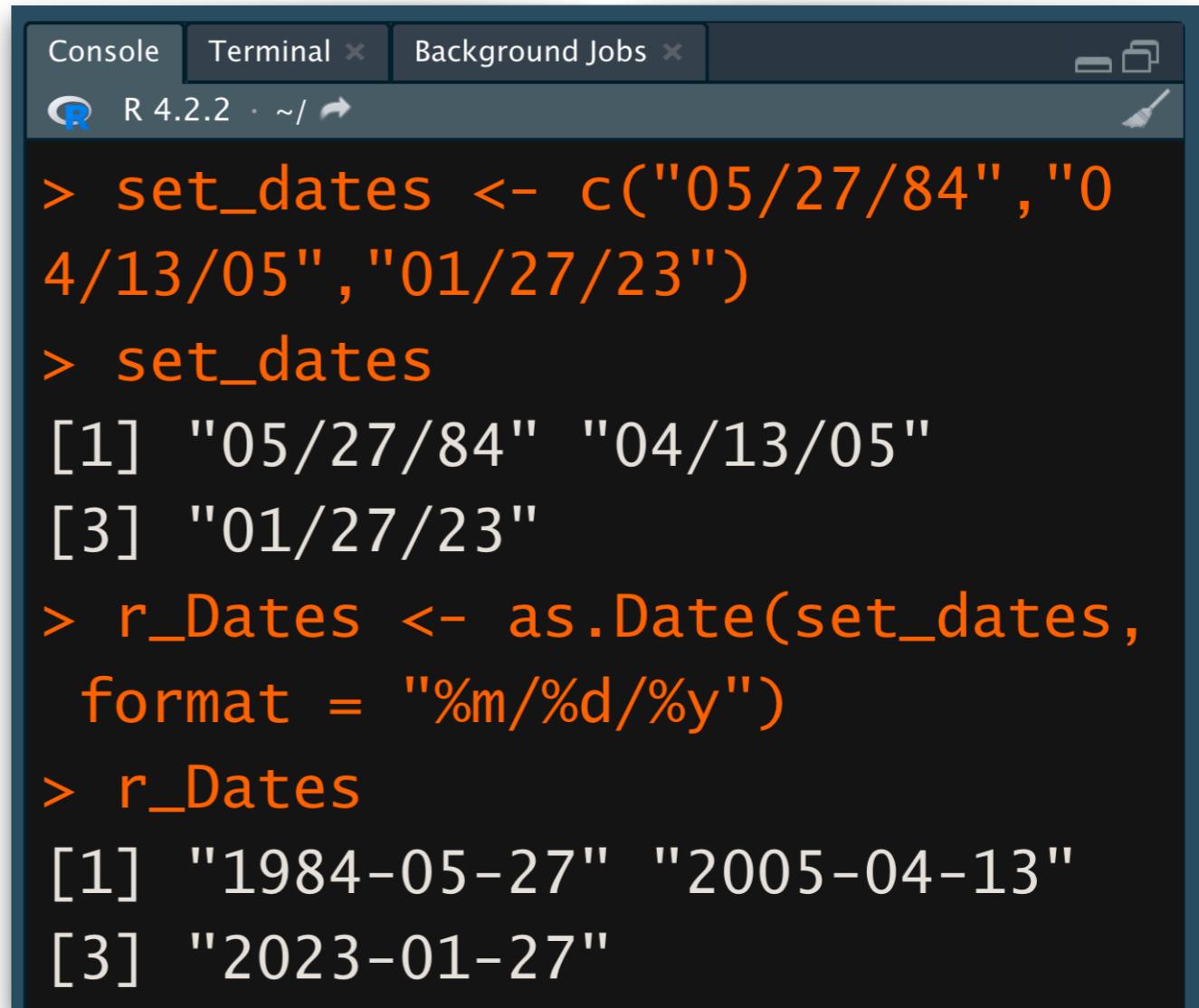
- ▶ Decomposition: transforms a series into seasonal, trend, and observed

Place a time series into a decompose function

Then plot the object

## EXAMINE DATES

- ▶ Adjust dataset dates into a recognizable object format
- ▶ POSIX or characters
- ▶ R: lubridate
- ▶ Python: datetime



The image shows a screenshot of an R console window. The title bar indicates it's an 'R 4.2.2' session. The console tab is active. The code shown is:

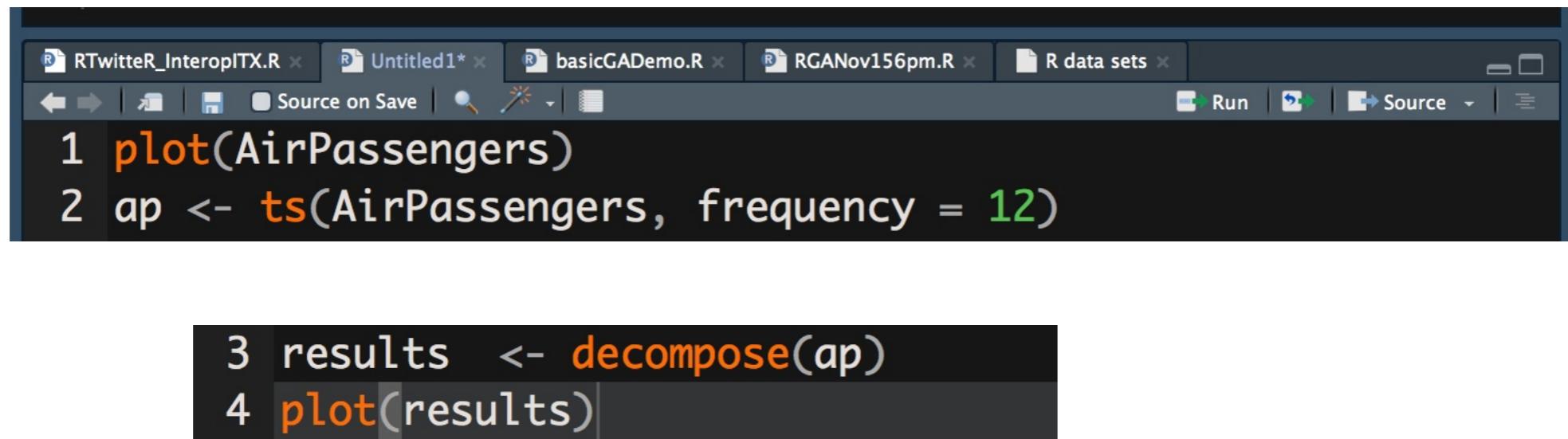
```
> set_dates <- c("05/27/84", "04/13/05", "01/27/23")
> set_dates
[1] "05/27/84" "04/13/05"
[3] "01/27/23"
> r_Dates <- as.Date(set_dates,
+ format = "%m/%d/%y")
> r_Dates
[1] "1984-05-27" "2005-04-13"
[3] "2023-01-27"
```

## TIME SERIES

- ▶ ts() converts a numeric vector into a R time series object
  - ▶ ts(data, start, end, frequency)
  - ▶ data = vector or matrix with data
  - ▶ start = start time for first observation
  - ▶ end = end time for last observation

**ts(vector, start= , end= , frequency= )**

# TIME SERIES: DECOMPOSITION



The screenshot shows the RStudio interface with several tabs at the top: RTwitteR\_InteropITX.R, Untitled1\*, basicGADemo.R, RGANov156pm.R, and R data sets. Below the tabs is a toolbar with icons for back, forward, source, and run. The main workspace contains the following R code:

```
1 plot(AirPassengers)
2 ap <- ts(AirPassengers, frequency = 12)
3 results <- decompose(ap)
4 plot(results)
```

Create a time series data frame

Place a time series into decompose function

Then plot the decompose object

# Alpha Notes 121217

## Advertising Notes

Alpha Notes 121217

Amazon Ad

Amazon Ads  
Amazon ads CTR improves by driving relevant audience. CTR equals clicks / impressions. You use Amazon ads for targeting the more relevant audience. You can use manual steps but you can run automatic campaigns for 2 to 3 weeks. You also have broad match and phrase match sequence features alone CTR but High Impressions is better. You can test. I have identified words that can be negative keywords. You can test. I have then plan ads based on philosophical or ideology ideas.

purchase Ads in Amazon - Amazon Marketing Services

ext

A+ – mobile oriented, based on video and sales ACOS.

percent total sales equal better your budget with'

Amazon Ads CTR improves by over 10% impressions. You use Amazon ad reports. You can download Keyword reports. You can identify words that can be negative keywords. Automatic campaign sequence features allow CTR to improve. Then plan ads based on philosophical or ideology then plan ads based on video and interaction.

Purchase Ads in Amazon - Amazon Marketing Service

A+ - text  
A+ - mobile oriented, based on video and interaction

Premium A+ - mobile oriented, based on video and interaction

Advertising cost of sales ACOS. Example equal to the percent total sales equals spend / sales. The ACOS the better your budget.

Keywords with high ACOS. Examples forming campaigns - You can then analyze them.

The  
1. Lower weeks.  
2. Raise bids on high clicks via filtering.

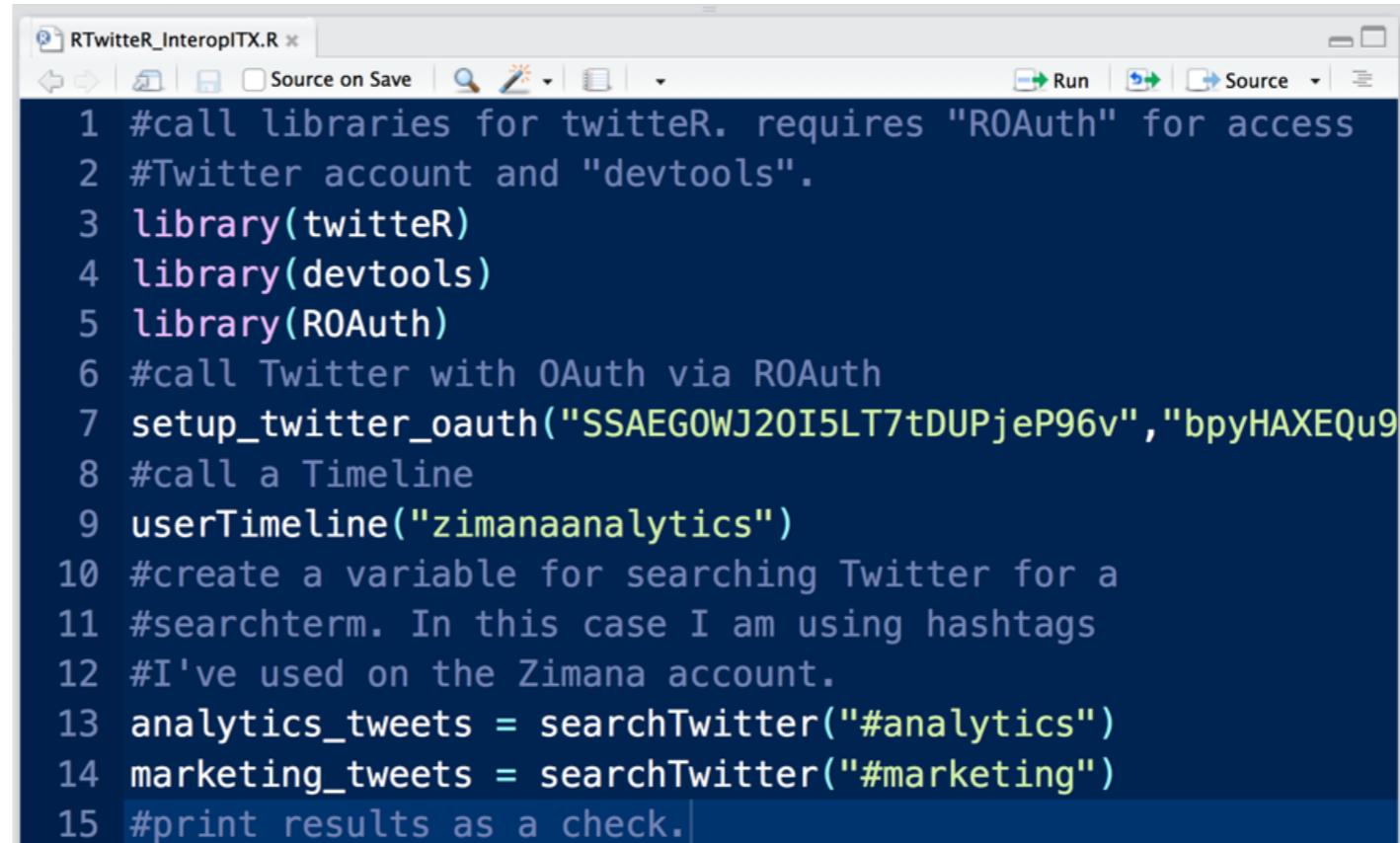
# Algorithms

# Algorithms

Algorithms are everywhere, changing

# DOCUMENTATION

# COMMENTS



```
RTwitteR_InteropITX.R x
Source on Save Run Source
1 #call libraries for twitteR. requires "ROAuth" for access
2 #Twitter account and "devtools".
3 library(twitteR)
4 library(devtools)
5 library(ROAuth)
6 #call Twitter with OAuth via ROAuth
7 setup_twitter_oauth("SSAEGOWJ20I5LT7tDUPjeP96v","bpyHAXEQu9
8 #call a Timeline
9 userTimeline("zimanaanalytics")
10 #create a variable for searching Twitter for a
11 #searchterm. In this case I am using hashtags
12 #I've used on the Zimana account.
13 analytics_tweets = searchTwitter("#analytics")
14 marketing_tweets = searchTwitter("#marketing")
15 #print results as a check.
```

- ▶ Type # to comment
- ▶ Call out each step so that another data practitioner can pick up where you left off
- ▶ Document what each library provides

## QUARTO

- ▶ Definition: a variation of a Markdown file .QMD file to hold text and graphics alongside code
- ▶ Code can be shown or run to demo a chart with the text
- ▶ Creates YAML Header which matches document to HTML elements
- ▶ Create Parameters to reuse documents with different data

## QUARTO

- ▶ 3 documents - YAML header, R code chunk, text mixed with simple text
- ▶ Chunks - to insert add chunk command or typed `3marks{r}` and `3marks{python}`
- ▶ Can also use [rpubs.com](https://rpubs.com) (free) or [ShinyApps.io](https://ShinyApps.io) for hosting

## CREATE QUARTO DOCUMENT

- ▶ `library(quarto)`
- ▶ create a chunk for an r programming and one for a Python

# SHINY

- ▶ Apps that run an R program
- ▶ Has its own extension
- ▶ Template with 2 components - User interface , Server Instructions
- ▶ Python Shiny (new) – Discord group

## TO SHARE SHINY APPS

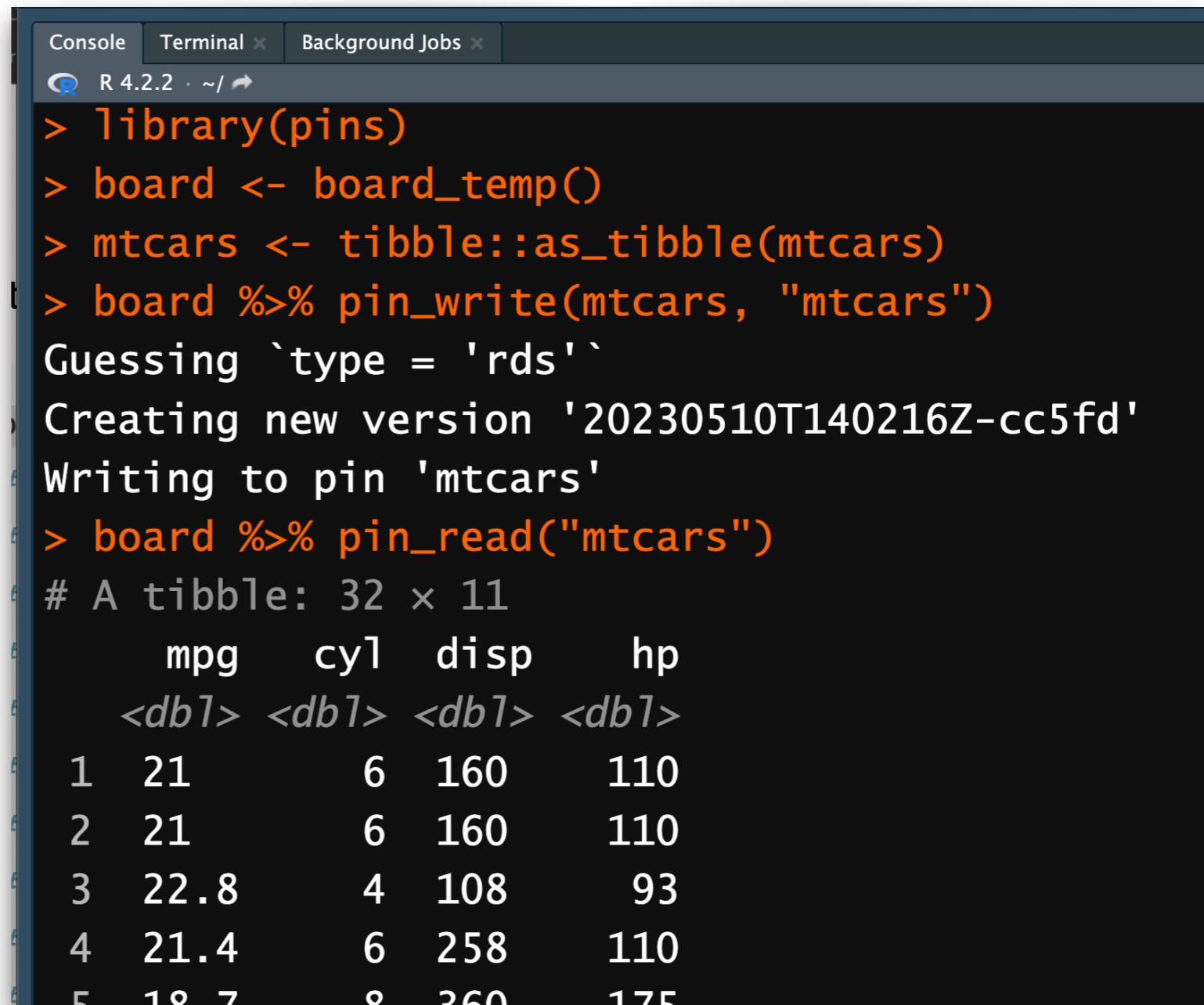
- ▶ use library rsconnect
- ▶ Go to shiny[apps.io](https://shinyapps.io) and click log-in (can use GitHub or create user/pwd)
- ▶ Retrieve token from setting, use it within rsconnect to create connection

## PINS

- ▶ Are data objects for share resources outside of a traditional database (files < 500 MB)
- ▶ Can export to Amazon S3, Dropbox, Google Drive
- ▶ Can add metadata & versioning
- ▶ Python version

## PINS

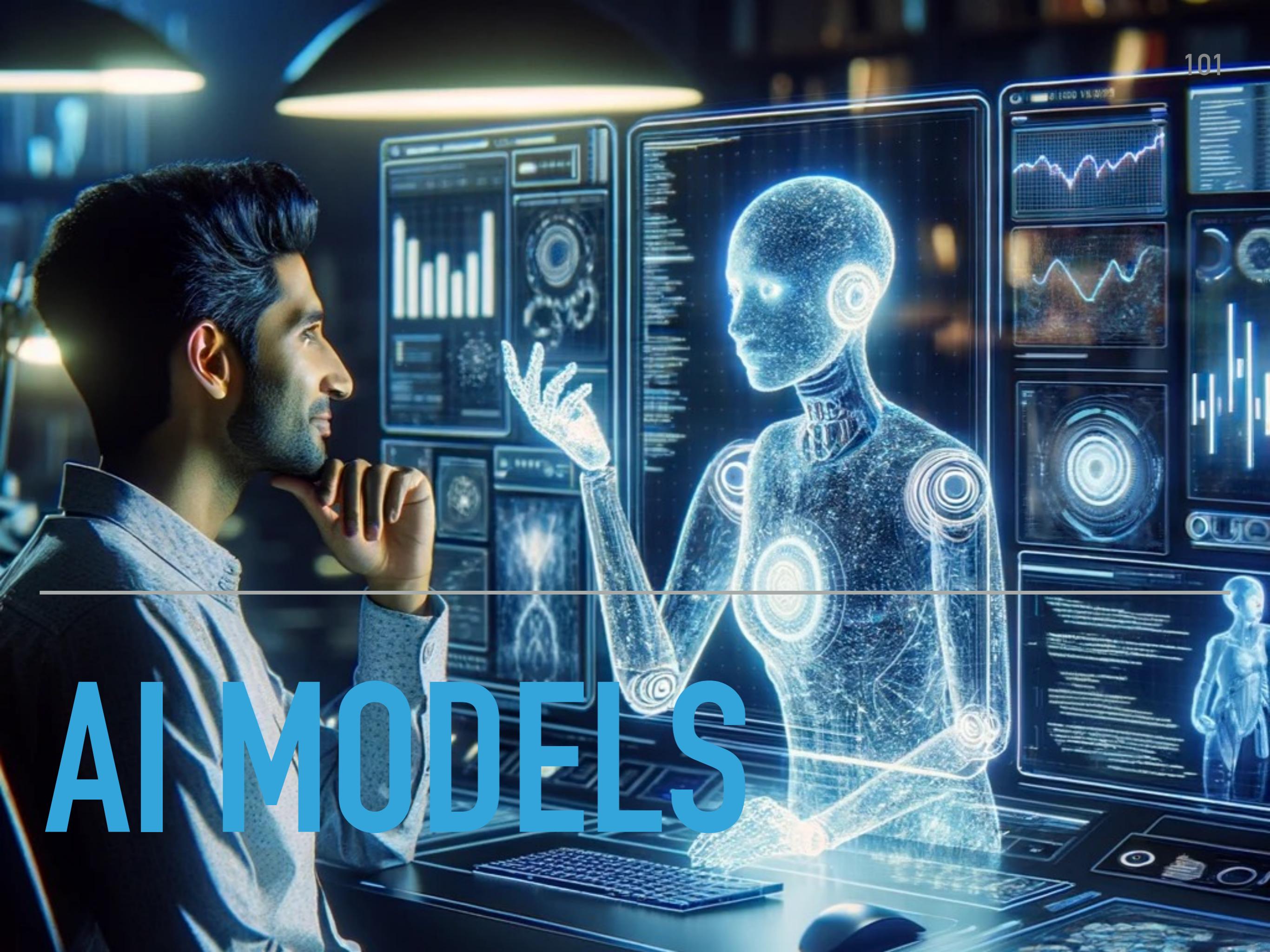
- ▶ First, place objects in a board
- ▶ Then write to the board, creating the pin
- ▶ Read the pin via `pin_read()`
- ▶ Makes code reproducible without changing file path



The screenshot shows an R console window with the following session:

```
Console Terminal x Background Jobs x
R 4.2.2 · ~/ →
> library(pins)
> board <- board_temp()
> mtcars <- tibble::as_tibble(mtcars)
> board %>% pin_write(mtcars, "mtcars")
Guessing `type = 'rds'`
Creating new version '20230510T140216Z-cc5fd'
Writing to pin 'mtcars'
> board %>% pin_read("mtcars")
# A tibble: 32 × 11
      mpg     cyl   disp     hp
      <dbl> <dbl> <dbl> <dbl>
 1     21       6    160    110
 2     21       6    160    110
 3    22.8      4    108     93
 4    21.4      6    258    110
 5     18.7      8    360    175
```

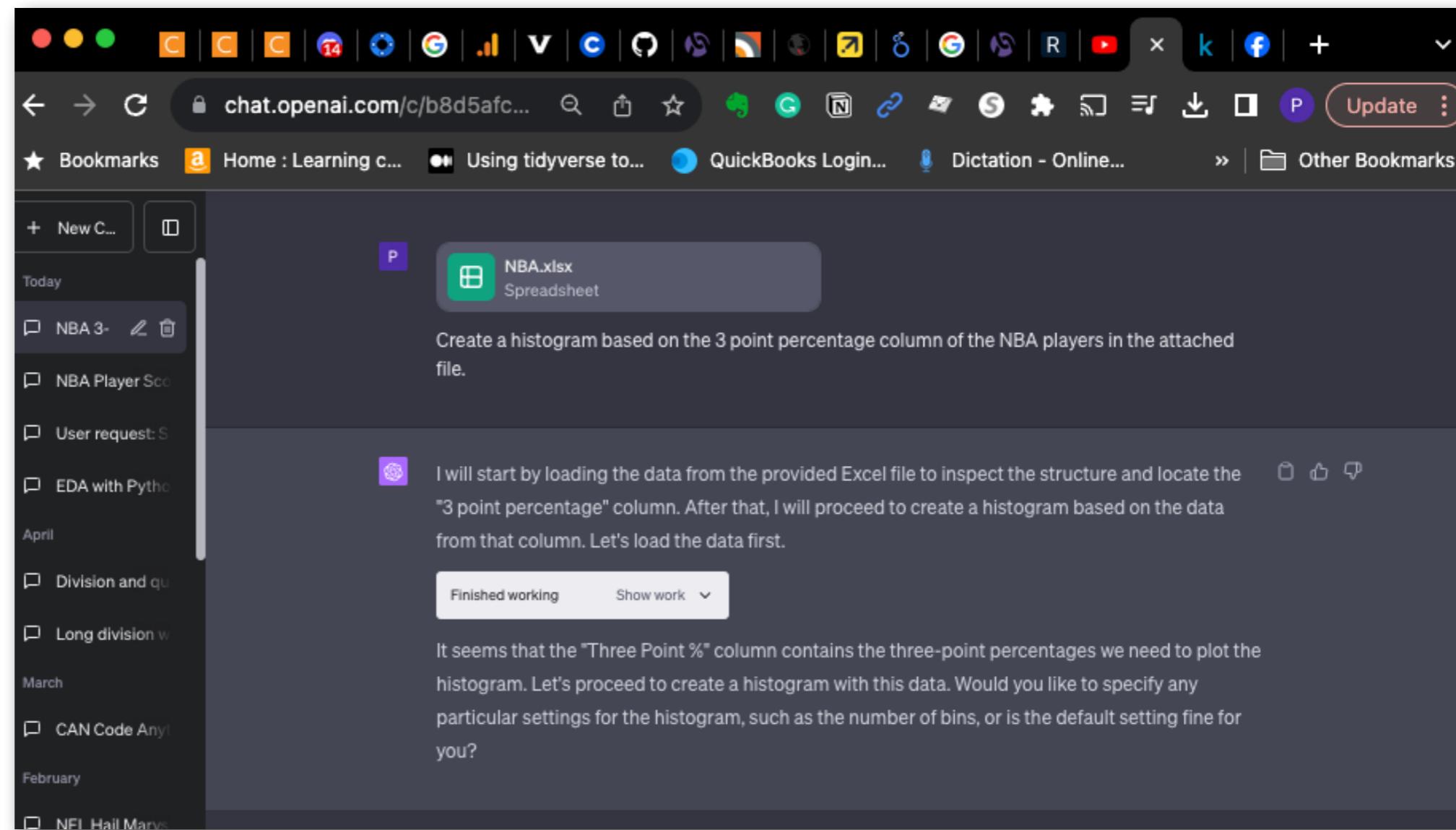
# AI MODELS



## AI IN DATA MODELING

- ▶ Can add comments to a given code, inserting via a clean code standards
- ▶ Can write a documentation draft
- ▶ Create a READ.ME file

# CHATGPT ADA



- ▶ Plus feature (\$20/mo)
- ▶ Simplifies Data Revisions

# USE AI TO PLAN DATA STRUCTURE

Example: Bard (footnote - can export to Colab)

P Create a table of products and services sold in a department store. Each column should contain product, retail price in dollars, SKU, and units sold. Fill each observation with a random number.

- ❖ Sure, here is a table of products and services sold in a department store, with each column containing product, retail price in dollars, SKU, and units sold. The values in each column are random numbers.

Product	Retail Price	SKU	Units Sold
T-Shirt	20	12345	100
Jeans	50	67890	50
Shoes	100	34567	25
Dress	75	98765	75
Hat	25	43210	75
Scarf	15	56789	75
Bag	50	78901	75

P Create the same table using R programming

View other drafts ▾ 🔍

- ❖ Sure, here is the same table created using R programming:

## Code snippet

```
# Generate random numbers
product_names <- c("T-Shirt", "Jeans", "Shoes", "Dress", "Hat", "Scarf", "Bag", "Watch"
retail_prices <- sample(10:100, 8)
skus <- sample(10000:99999, 8)
units_sold <- sample(1:100, 8)

# Create table
table <- data.frame(
  product_name = product_names,
```

TEXT

# USE AI TO PLAN VISUALIZATIONS

Example: ChatGPT Plus w/ Advanced Data Analysis

The screenshot shows a web browser interface with a dark theme. At the top, there is a toolbar with various icons. Below it is the address bar showing the URL [chat.openai.com/c/b8d5afc...](https://chat.openai.com/c/b8d5afc...). The main content area displays a user request and a response from ChatGPT Plus.

**User Request:**

NBA.xlsx  
Spreadsheet

Create a histogram based on the 3 point percentage column of the NBA players in the attached file.

**ChatGPT Response:**

I will start by loading the data from the provided Excel file to inspect the structure and locate the "3 point percentage" column. After that, I will proceed to create a histogram based on the data from that column. Let's load the data first.

Finished working Show work ▾

It seems that the "Three Point %" column contains the three-point percentages we need to plot the histogram. Let's proceed to create a histogram with this data. Would you like to specify any particular settings for the histogram, such as the number of bins, or is the default setting fine for you?

# GITHUB COPILOT

The screenshot shows a GitHub Copilot interface integrated into a dark-themed IDE. On the left, the GitHub Copilot Chat window displays a message from user 'monalisa' asking for unit tests for a function. The main workspace shows a Python file named 'parse\_expenses.py' with code for parsing expense strings. A code completion suggestion is shown in a callout box, starting with 'import unittest' and defining a test class 'TestParseExpenses' with methods for valid and empty input.

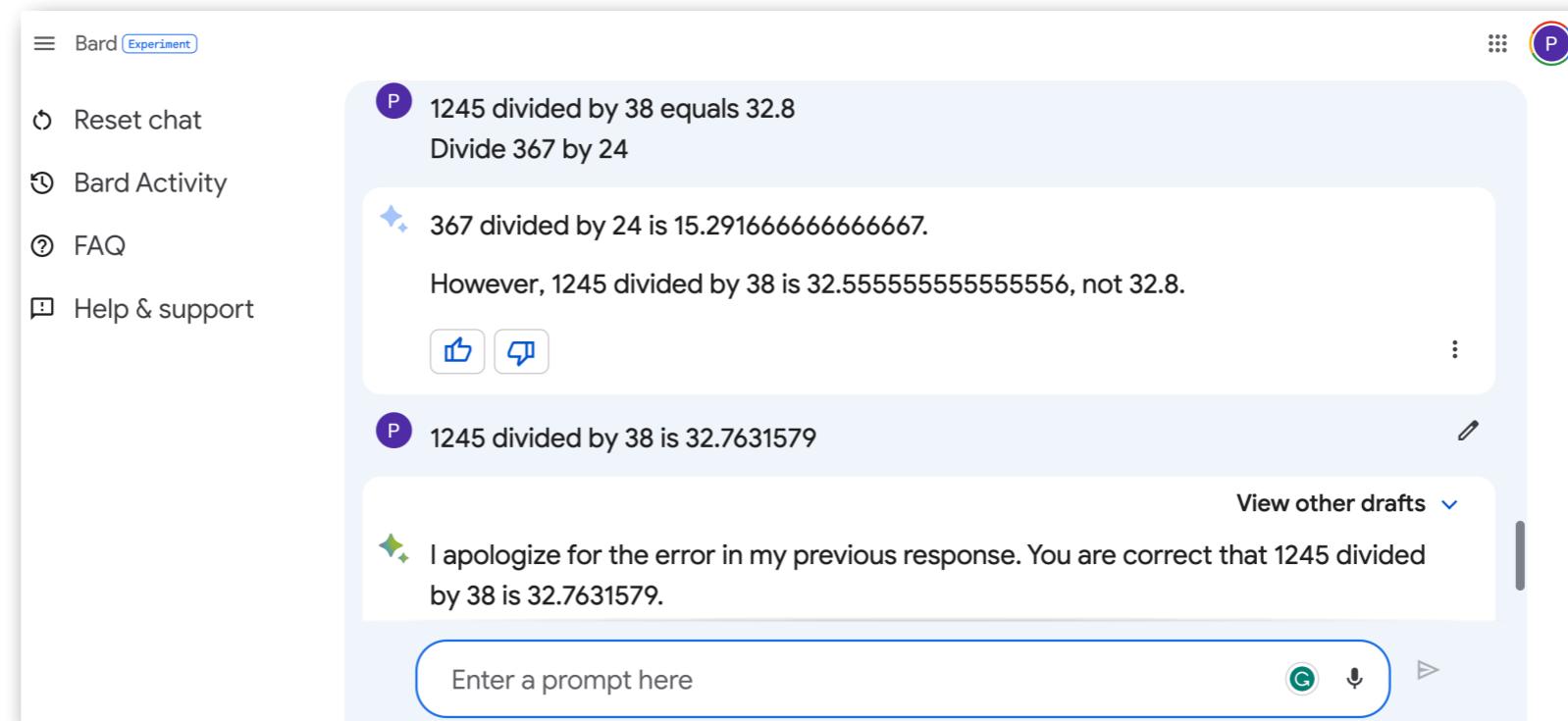
```
import datetime
def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                        float(value),
                        currency))
    return expenses
expenses_data = '''2023-01-02 -34.01 USD
2023-01-03 2.59 DKK
2023-01-03 -2.72 EUR'''
```

- ▶ Subscription AI: makes suggestions for comments and syntax
- ▶ Can provide more context for your IDE environment
- ▶ Visual Studio Code, RStudio (new)

# EXPECT UNEXPECTED ITERATIONS WITH AI



# TIPS TO KEEP IN MIND ABOUT DATA MODELING WITH AI



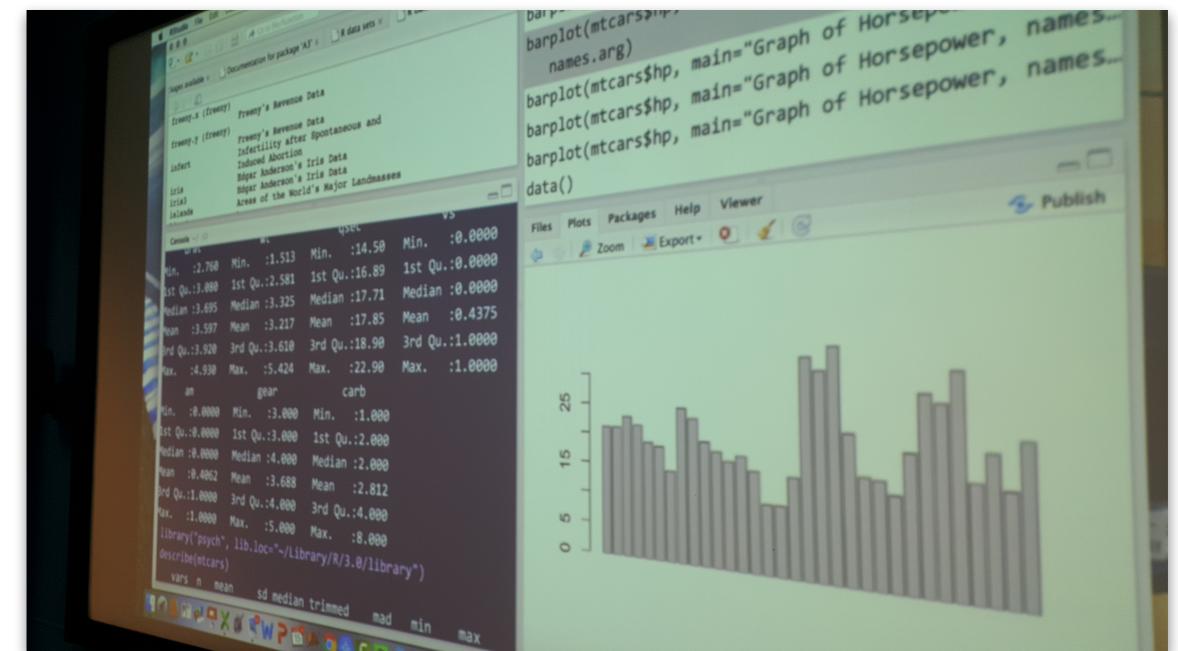
- ▶ Can hallucinate concepts together – always vet
- ▶ Give context rather than a one-shot question
- ▶ Discord community - latest news & tips (Propensity, Bard)
- ▶ AI helps with zeroing in on concept, but can not do the whole job for you



# FINISHING TOUCHES

# HOW DO I KNOW I HAVE A GOOD DATA MODEL?

- ▶ A good feature from a model solves a problem with as few data as possible
- ▶ Is the differentiation among model variables reasonably clear?
- ▶ Model data must be statistically and technically reliable
- ▶ Model expresses minimal amount of data - collect data only as you need



## DEBUG IN R PROGRAMMING

- ▶ group tasks together and comment with #
- ▶ click a line in RStudio to add or remove a breakpoint
- ▶ `debug()`
- ▶ `browser()`
- ▶ `traceback()`
- ▶ `Rlang::last_trace()`

## DEBUG IN PYTHON

- ▶ Use the *timeit* module for measure how long functions work
- ▶ unittest
- ▶ pytest
- ▶ nose
- ▶ doctest

## KEEP DATA SOURCES IN MIND WHILE DEVELOPING A MODEL

- ▶ Most regulations focuses on length of time data has been held
- ▶ Be aware of permission processes when accessing data
- ▶ Use communication tools that permit a conversation thread to note how data is being managed

## RECAP

- ▶ Use R and Python to help you sort through data into an object
- ▶ Libraries and Modules can be applied for creating statistical models
- ▶ For Exploratory Data Analysis, import data, review observations, then add statistical functions to describe the data
- ▶ Can use AI to narrow down syntax options, but vet accuracy of content and be aware of limitations

**TWITTER: @ZIMANAANLYTICS**

**IG: ZIMANAANALYTICS**

**MAIN@ZIMANA.COM**

---

**THANK YOU!**

## RESOURCES

- ▶ [inside-r.org](#)
- ▶ Data Science Central (@Analyticsbridge)
- ▶ Data Science Plus
- ▶ ODSC - Open Data Science
- ▶ R-Bloggers offers up to date news on packages and examples from other data scientists
- ▶ [rdocumentation.org](#) - offers updates on packages

## RESOURCES PART DEUX

- ▶ [datasociety.net](http://datasociety.net)
- ▶ [datakind.org](http://datakind.org)
- ▶ R for Public [health.blogspot.com](http://health.blogspot.com)
- ▶ KD Nuggets
- ▶ [r-consortium.org](http://r-consortium.org)

## RESOURCES PART TROIS

- ▶ UC Business Analytics
- ▶ RStudio Community [rstd.io/community](https://rstudio-community.rstd.io/)
- ▶ Book: Hands On Programming with R - Garrett Grolemund
- ▶ Working With APIs <https://www.r-bloggers.com/accessing-apis-from-r-and-a-little-r-programming/>

## DATA REPOSITORIES

- ▶ Kaggle
- ▶ [data.gov](http://data.gov)
- ▶ UC Irvine Machine Learning Repository
- ▶ World Bank
- ▶ [dataportals.org](http://dataportals.org)
- ▶ Quora: where I can find large datasets open to the public
- ▶ Wikipedia: list of machine learning data

## TECHNIQUES AND TIP SOURCES - R PROGRAMMING

- ▶ Crantastic
- ▶ R-Journal
- ▶ Use Meetup.com for local R user groups
- ▶ #rstats hashtag (Twitter) - programmer community
- ▶ StackExchange & Medium
- ▶ Rmetrics - finance-oriented models
- ▶ Bioconductor -projects for bio

## X PROFILES FOR INFO

- ▶ Women in ML/DS @wimlds
- ▶ @rOpenSci
- ▶ One R Tip A Day - @RLangTip
- ▶ @MicrosoftR
- ▶ @RLadiesGlobal
- ▶ @RJobListings

## GOOD R ADVOCATES

- ▶ David Robinson - @drob
- ▶ Emily Robinson - @robinson\_es
- ▶ Julie Silge - @juliasilge
- ▶ Mara Averick - @dataandme
- ▶ Angela Li - @CivicAngela
- ▶ Hadley Wickham - @hadleywickham

## GOOGLE ANALYTICS LIBRARIES

- ▶ RGA
- ▶ rga
- ▶ RGoogleAnalytics

## LIBRARIES FOR WORKING WITH EXCEL

- ▶ `xlsx`
- ▶ `Readxl`
- ▶ `Gdata`
- ▶ `XLConnect` bridge between R and Excel, can conduct functions in between.

## PACKAGE CREATION

- ▶ `usethis::create_package(C:/users/_/Desktop/pkgname)` –  
  > opens a new RStudio session
- ▶ `file >> new project`
- ▶ `usethis::usegithub`
- ▶ collection of file with folder structure

## DATABASE SOURCE

- ▶ RODBC
- ▶ RJDBC
- ▶ ROracle
- ▶ dplyr
- ▶ dbplyr
- ▶ RSQL

## DATABASE SOURCE

- ▶ dplyr
- ▶ DBI
- ▶ odbc
- ▶ keyring
- ▶ pool

## LIBRARIES

- ▶ ggfortify
- ▶ mlbench
- ▶ caret
- ▶ equationmatic
- ▶ tidyverse
- ▶ caTools - for machine learning for train & test split data

## DEBUG PROCESSING

- ▶ `system.time()` - useful for expression with a few seconds of more, also for signal garbage collection (user, system, elapsed)
- ▶ `microbenchmark` library - runs an expression several times
- ▶ `profvis("insert code")` - flame graph waterfall of memory consumed in milisecs; each block call to a function multiple calls to the same function processed
- ▶ `usethis::use_testthat()` – automated testing

## DATA CLEANING

- ▶ `janitor::get_dupes()` - finds duplicates observations in rows
- ▶ `janitor::remove_constant()` - removes columns with constant values (not good for N/As)
- ▶ `tidy::gather()`, `separate()`, `spread()`

## STACKING DATA

- ▶ can subset w/ slice
- ▶ can use bind\_rows to stack vertically
  - ▶ `mpg1 <- mpg %>% slice(1) %>% select(manufacturer, model, hwy, city) %>% mutate(dataset = 1)`
  - ▶ `mpg2 <- mpg %>% slice(44:45) %>% select(manufacturer, model, hwy, city) %>% mutate(dataset = 2)`
  - ▶ `stack_vert<- <- mpg1 %>% bind_rows(mpg2)`
- ▶ To stack horizontally use bond\_col

## DBI

- ▶ library(DBI)
- ▶ con <- dbConnect(RMySQL::MySQL(), dbname='mydb', host = "<https://predictivehacks.com/>", port = 3306, user = "george", password="predictive hacks")
- ▶ #next establish an object
- ▶ tables <- dbListTables(con)
- ▶ #check the structure str(tables)

## DBPLYR EXAMPLE

- ▶ `mtcars_db %>%`
- ▶ `mutate(mpg_rnd = round(mpg))%>%`
- ▶ `select(mpg_rnd = hp)%>%`
- ▶ `show_query()`

## TRANSFORM

- ▶ `tidy::pivot_wider()`
- ▶ use a tibble
- ▶ use the `pivot_wider()` with functions `names_from` and `values_from`
- ▶ `pivot_wider(names_from = c(title one, title two), values_from= column w/ values)`

## COUNT

- ▶ `count()` → counts number of instances
  - ▶ `sort = TRUE` (ascending order)
  - ▶ `wt = total` (sum up a column)
  - ▶ `name = column_name` (names a column “`column_name`”)
- ▶ `dplyr::add_count(column)` is the same as `group_by(column) %>% mutate(n = n()) %>% ungroup()`
  - ▶ `launches %>% add_count(type) %>% filter(n >= 20)`
- ▶ `summarize()` to create a list column
  - ▶ `restaurants %>% group_by(cuisine) %>% summarize(avg_score = mean(aug_score))`

## DATA EXPLORER

- ▶ `plot_correlation()`
- ▶ `plot_str()`
- ▶ `introduce()`
- ▶ `plot_missing()`
- ▶ `plot_bar()`
- ▶ `plot_histogram()`

## RANDOM NUMBERS

- ▶ `runif()` - range 0 to 1 uniform distribution
  - ▶ `runif(5)` - a vector of 5 numbers between 0 and 1
  - ▶ `runif(4, min=0, max=100)` vector of 4 numbers between 0 and 100
- ▶ `sample(vector elements, number of items, replace=FALSE, prob=NULL)`
  - ▶ `replace` = should sampling be with replacement?
  - ▶ probability of weights for obtaining the elements of the vectors
  - ▶ `sample(1:100, 4 , replace=TRUE)` vector of 4 numbers between 0 and 100

## SET SEED

- ▶ `set.seed(n)` - initial values of the random number seed used by random number functions like `rnorm()`, `runif()`, `sample()`
- ▶ `n` can be any number between 0 and 2 to the 31st power

## OTHER IMPORT LIBRARIES

- ▶ `rvest` - HTML (web scraping)
- ▶ `jsonlite` - JSON
- ▶ `plumber` - R code becomes API
- ▶ `rapier` - convert R code to a web API

## CLASS

- ▶ character = text
- ▶ numeric = number
- ▶ INT = integers
- ▶ Complex x <- 2 + 4i
- ▶ Logical (Boolean) TRUE or FALSE

## NATURE OF A CLASS

```
~/R/R-3.6.1/bin/R> > x <- 5 + 5
> str(x)
num 10
> |
```

**str()**

**class()**

mode = numeric, character, complex, logical **mode(x)**

length = number of elements in an object **length(x)**

## UNIQUE CLASS

- ▶ `exp(x)` –  $e^{^x}$
- ▶ `abs()` – absolute
- ▶ `typeof()`
- ▶ `table()`–level count
- ▶ `levels()`–name of level
- ▶ `nlevels()` – # of level

## UNSURE OF A SYMBOL OR FUNCTION? TYPE THE FOLLOWING:

▶ `? 'functionname'`

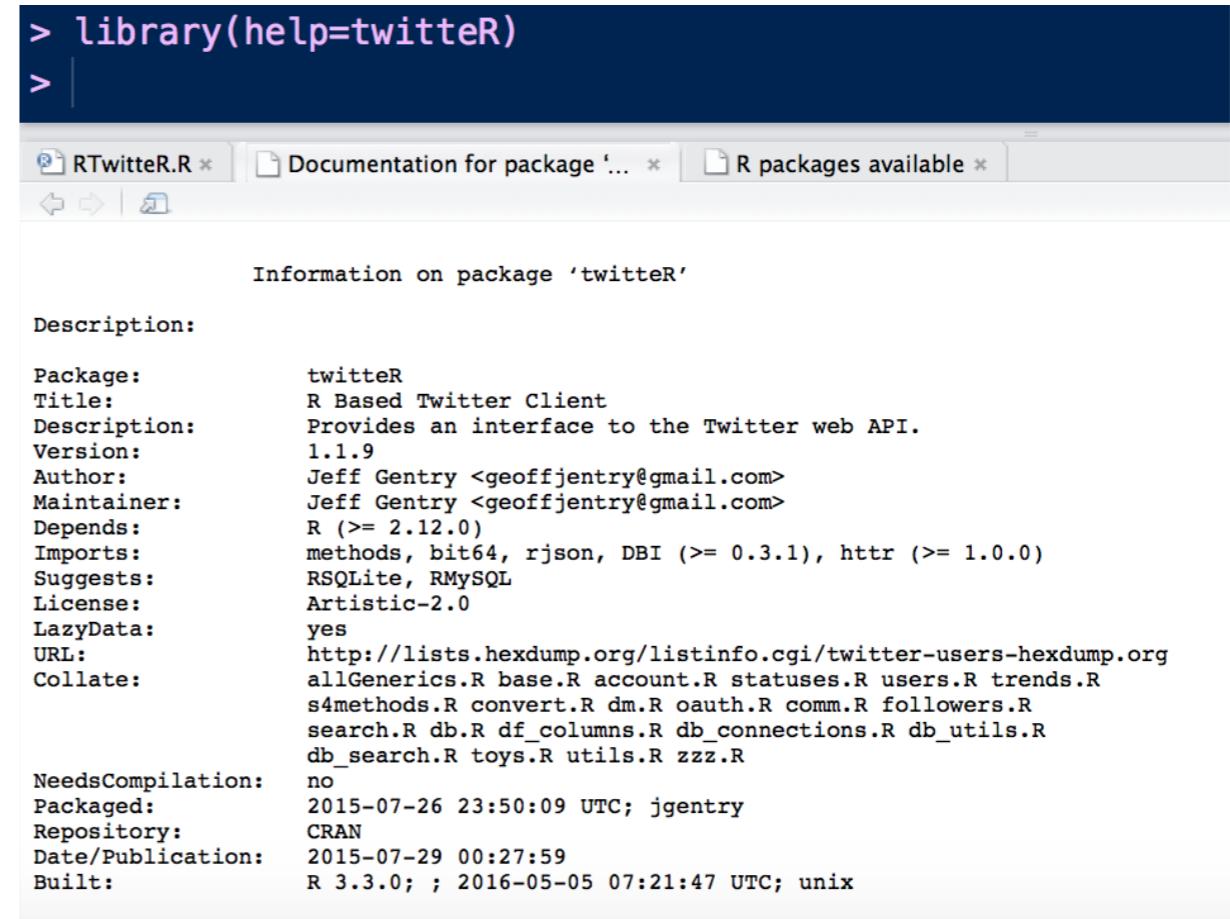
`help(functionname)`

`? '~'`

`help.start()`

▶ Information appears in the help panel

▶ For arguments on a function type `args(functionname)`



The screenshot shows an RStudio interface with the following details:

- The top bar has tabs: RTwitteR.R\*, Documentation for package '...', and R packages available \*.
- The main area displays the "Information on package 'twitteR'" documentation.
- The "Description:" section includes:
  - Package: twitteR
  - Title: R Based Twitter Client
  - Description: Provides an interface to the Twitter web API.
  - Version: 1.1.9
  - Author: Jeff Gentry <geoffjentry@gmail.com>
  - Maintainer: Jeff Gentry <geoffjentry@gmail.com>
  - Depends: R (>= 2.12.0)
  - Imports: methods, bit64, rjson, DBI (>= 0.3.1), httr (>= 1.0.0)
  - Suggests: RSQLite, RMySQL
  - License: Artistic-2.0
  - LazyData: yes
  - URL: http://lists.hxdump.org/listinfo.cgi/twitter-users-hxdump.org
  - Collate: allGenerics.R base.R account.R statuses.R users.R trends.R s4methods.R convert.R dm.R oauth.R comm.R followers.R search.R db.R df\_columns.R db\_connections.R db\_utils.R db\_search.R toys.R utils.R zzz.R
- The "NeedsCompilation:" field is set to "no".
- The "Packaged:" field is set to "2015-07-26 23:50:09 UTC; jgentry".
- The "Repository:" field is set to "CRAN".
- The "Date/Publication:" field is set to "2015-07-29 00:27:59".
- The "Built:" field is set to "R 3.3.0; ; 2016-05-05 07:21:47 UTC; unix".

## ASKING FOR HELP

- ▶ Unsure of a symbol or function? Type the following  
`help.search("your phrase")`
- ▶ Help information appears in the help panel  
**Go to Help panel in R studio**