INSY 6500
Information Systems for Operations

Introduction to R

Fall 2019

Jeffrey S. Smith

Industrial and Systems Engineering Department
Auburn University
jsmith@auburn.edu

What is R?

https://www.r-project.org/about.html

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Our R-related Resources

- Course R projects
  - GitHub: code\R\In-Class Work.Rprj
  - …
- R For Data Science Site - http://r4ds.had.co.nz/
- The R Project Site - https://www.r-project.org/
- CRAN – https://cran.r-project.org/
- Tutorialspoint - https://www.tutorialspoint.com/r/index.htm

… Many, many other tutorials and sites with code samples, etc.

Features of R

- **R** is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- **R** has an effective data handling and storage facility,
- **R** provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- **R** provides a large, coherent and integrated collection of tools for data analysis.
- **R** provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

https://www.tutorialspoint.com/r/r_overview.htm

Outline of Our R Coverage

1. RStudio
2. Introduction to R (Ch 1 + Tutorialspoint + misc. material from Chs 14, 15, 16, 19, 20)
3. Data Visualization (Ch 2)
4. Data Transformation (Ch 5)
5. Exploratory Data Analysis (Ch 7)
6. Tibbles (Ch 10)
7. Data Import (Ch 11)
8. Tidy Data (Ch 12)
9. Relational Data (Ch 13)

RStudio

- Information about RStudio and Related topics:
  - Using the IDE - https://support.rstudio.com/hc/en-us/sections/200107586-Using-the-RStudio-IDE
  - Projects - https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects
  - R Notebooks - https://blog.rstudio.com/2016/10/05/r-notebooks
  - R Markdown - https://rmarkdown.rstudio.com/

R Variables, Objects, Data Types

- `basic_R_programming.R`
- R *variables* are like Python variables in that they point to an object and have no inherent *type.*
- Frequently used object types:
  - Vectors
  - Lists
  - Matrices
  - Arrays
  - Factors
  - Data Frames
- Frequently used data types:
  - Logical (TRUE/FALSE)
  - Numeric (R switches as necessary)
    - Double-precision float
    - Integer
  - Integer
  - Complex
  - Character
  - Raw (stored as hex)

Functions in R

```
function_name <- function(arg_1, arg_2 ...) {
     Function body
}
```

- **Function Name** – This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** – An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** – The function body contains a collection of statements that defines what the function does.
- **Return Value** – The return value of a function is the last expression in the function body to be evaluated.

https://www.tutorialspoint.com/r/r_functions.htm

Lexical Scoping

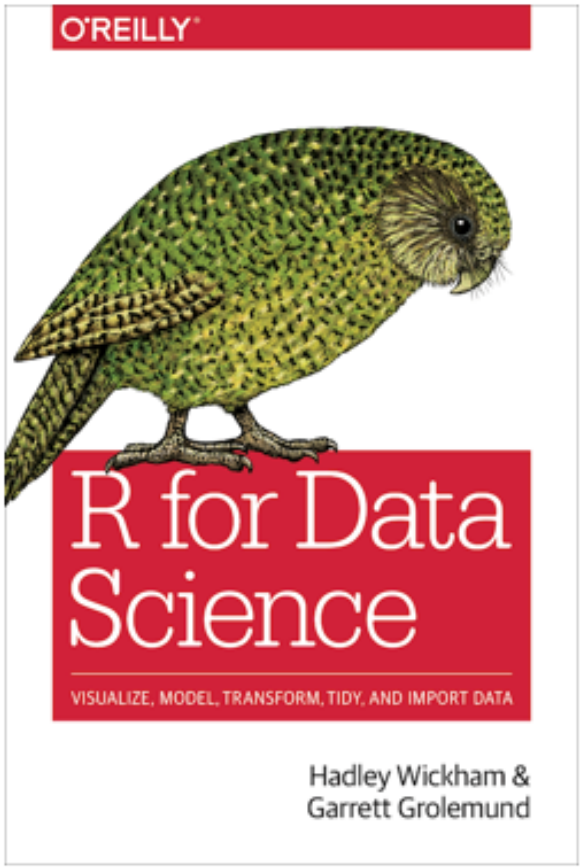
```r
#
# Lexical Scoping Example
#

# Note that a is defined inside the function, y is used,
# but not defined or passed as an arguement, z is defined
# and returned.
my_fun <- function(x) {
  a = "dog"
  z = x+y
  print(ls())
  return(z)
}


# note that the a and z variables defined in the function
# aren't in the environment.
y = 207
b = my_fun(13)
print(ls())
```

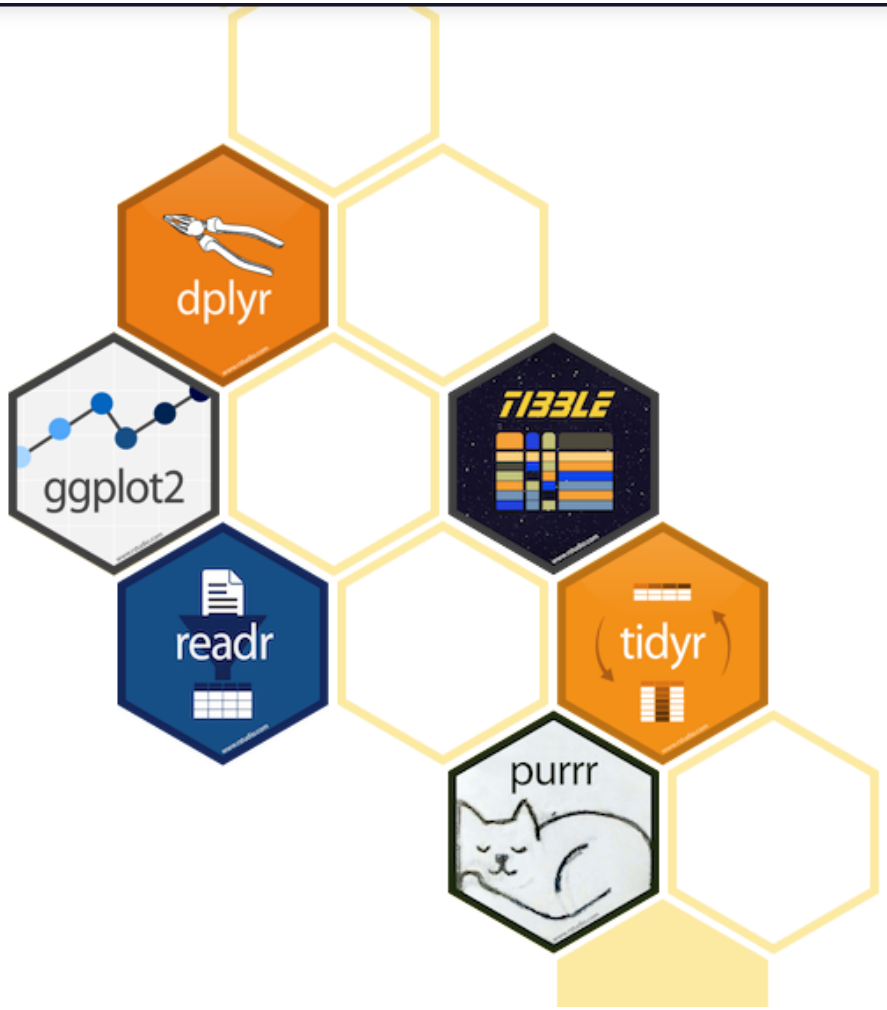
- What would the output look like?

lexical_scoping_example.R

The Tidyverse

- From our R installation, we need to install a couple of *packages* that we'll be using in *R for Data Science*:
  - install.packages("tidyverse")
  - install.packages(c("nycflights13", "gapminder", "Lahman"))
- What is the Tidyverse?
  - https://www.tidyverse.org/
- Rules for Tidy Data
  1. Each variable must have its own column.
  2. Each observation must have its own row.
  3. Each value must have its own cell.





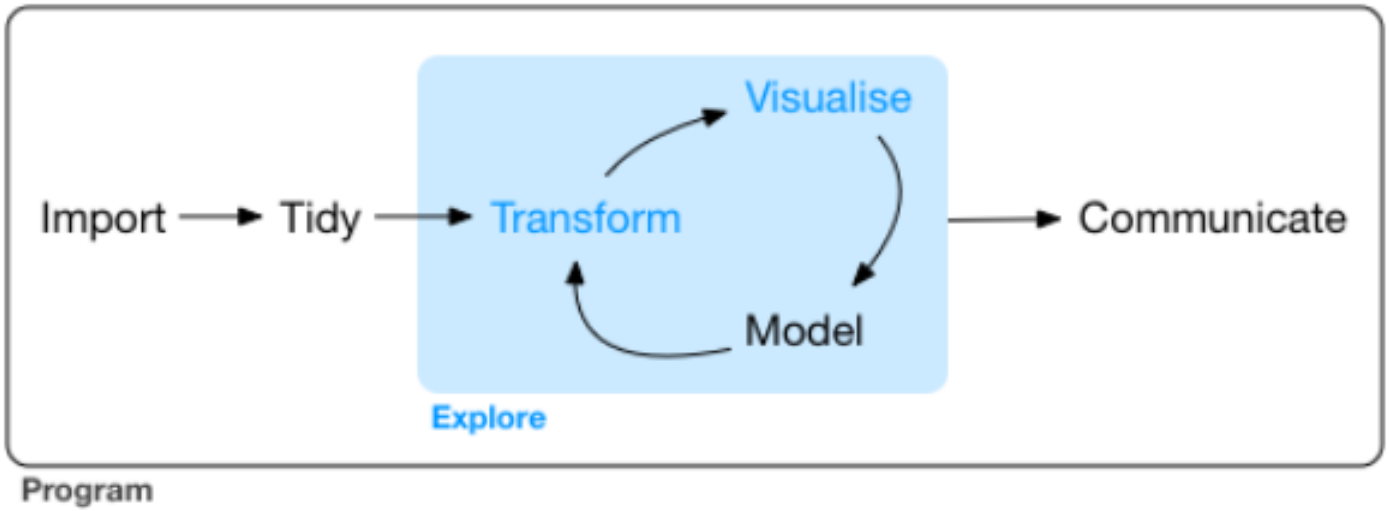
R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Grolemund and Wickham's view of Data Exploration



http://r4ds.had.co.nz/explore-intro.html

Data Visualization (Ch. 3)

- `visualization_chapter.R`
- ggplot2 – Implements the "Grammar of Graphics" (http://vita.had.co.nz/papers/layered-grammar.pdf)
- Example + Preliminary *graphing template:*

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))

ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

- https://www.rstudio.com/resources/cheatsheets/

Stats

1. **`geom_bar()`** begins with the **`diamonds`** data set

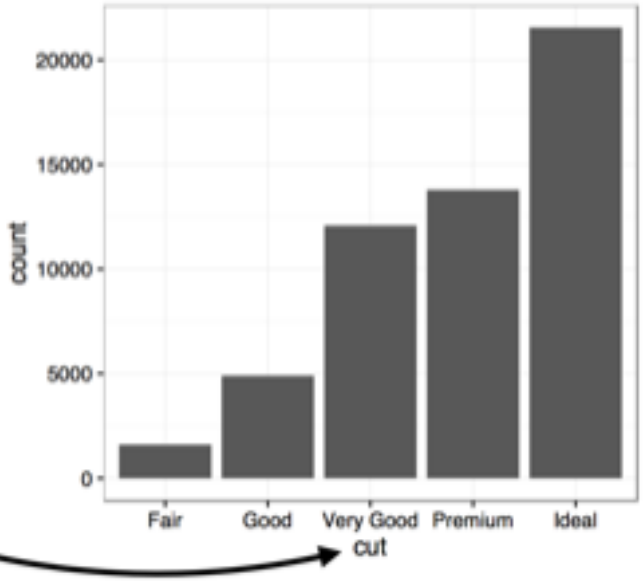
carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

stat_count()

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.



http://r4ds.had.co.nz/data-visualisation.html

Layered Grammar of Graphics

```
ggplot(data = <DATA>) +
     <GEOM_FUNCTION>(
          mapping = aes(<MAPPINGS>),
          stat = <STAT>,
          position = <POSITION> ) +
     <COORDINATE_FUNCTION> +
     <FACET_FUNCTION>
```

http://r4ds.had.co.nz/data-visualisation.html

Data Transformation (Ch. 5)

- dplyr basics
  - Pick observations by value – filter()
  - Reorder the rows – arrange()
  - Pick variables by names – select()
  - Create new variables with functions of existing variables – mutate()
  - Collapse many values down to a single summary – summarize()
- group_by()
- All verbs (functions) work similarly:
  1. A data frame is the first argument
  2. Subsequent arguments describe what to do with the data frame, unquoted variable names
  3. The result is a new data frame

Pipes

- Suppose that you want to some analysis on flight delays by destination. The following would be a logical process:
  1. Group flights by destination
  2. Summarize the pertinent values – count, distance, delay
  3. Filter to remove outliers
- Earlier we said that the transformations result in new dataframes – so the process can generate lots of intermediate dataframes that we're not interested in using other than as input for the "next step in the process."
- The *pipe* eliminates the need for saving the intermediate dataframes.

http://r4ds.had.co.nz/transform.html#grouped-mutates-and-filters

Pipes

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
   count = n(),
   dist = mean(distance, na.rm = TRUE),
   delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
```

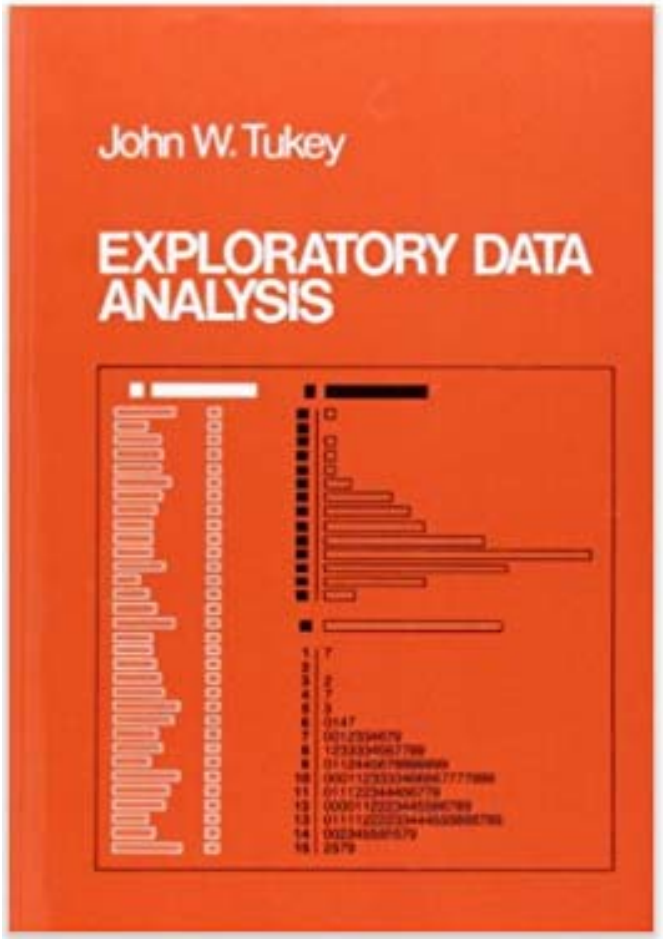
```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

Example – Meals Data

- Consider the tips dataset that we used in the Python section of the class.
- 894 records with the fields:
  - payer
  - meal
  - party_size
  - day
  - cost
  - tip

- Suppose that we want to:
  1. Compute the tip percentage for each record;
  2. Determine the average tip, average meal price, median tip percentage, total number of people:
     a) overall
     b) by day
     c) by meal
     d) by payer

Exploratory Data Analysis (EDA)



"John W. Tukey wrote the book Exploratory Data Analysis in 1977. Tukey held that too much emphasis in statistics was placed on statistical hypothesis testing (confirmatory data analysis); more emphasis needed to be placed on using data to suggest hypotheses to test. In particular, he held that confusing the two types of analyses and employing them on the same set of data can lead to systematic bias owing to the issues inherent in testing hypotheses suggested by the data."

https://en.wikipedia.org/wiki/Exploratory_data_analysis

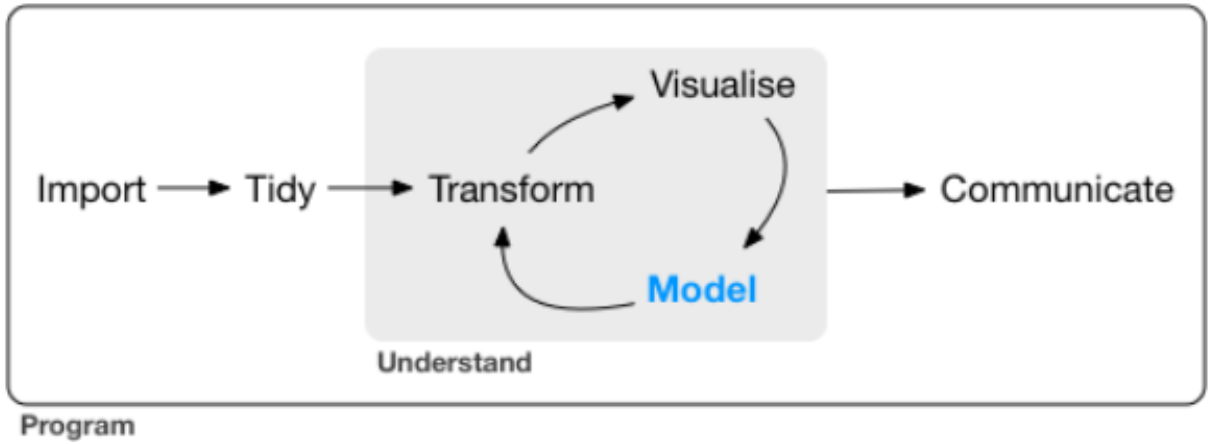
Exploratory Data Analysis (EDA)

- Chapter 7 - http://r4ds.had.co.nz/exploratory-data-analysis.html
- EDA is an iterative cycle where you:
  1. Generate questions about your data;
  2. Search for answers by visualizing, transforming, and modelling your data;
  3. Use what you learn to refine your questions and/or generate new questions.
- "There is no rule about which questions you should ask to guide your research. However, two types of questions will always be useful for making discoveries within your data. You can loosely word these questions as:
  - What type of variation occurs within my variables?
  - What type of covariation occurs between my variables?"

Our Dataset

- Real Estate Transactions from Auburn for 2017
- 1,326 Records with fields:
  - MLSID (char)
  - Address (char)
  - Subdivision (char)
  - Price (Int)
  - SqFt (Int)
  - Bedrooms (int)
  - BathsFull (int)
  - BathsHalf(int)
  - DaysOnMarket (int)
  - Agent (char)
  - Firm (char)
  - Baths (int)
  - PType (char)
  - NBed (char)
  - NBath (char)

Model (Chs. 22, 23, 24)



- Hypothesis generation vs. Hypothesis confirmation – *Exploration* vs. *Inference*
- To do inference correctly:
  1. Each observation can either be used for exploration or confirmation, but not both.
  2. You can use an observation as many times as you like for exploration, but you can only use it once for confirmation. As soon as you use an observation twice, you've switched from confirmation to exploration.

http://r4ds.had.co.nz/model-intro.html