Data Analytics Course 7: Data Analysis with R Programming

Module 1

Computer programming: giving instructions to a computer to perform an action or set of actions

R is useful for organizing, cleaning, and analyzing data

Mistakes, errors, and error messages are part of the process!

Programming languages: the words and symbols we use to write instructions for computers to follow.

- Syntax: shows you how to arrange the words and symbols you enter.

Coding is writing instructions to the computer in the syntax of a specific programming language.

Python: a general-purpose language that can be used for all sorts of things, from working with artificial intelligence to creating virtual reality experiences.

Other popular programming languages for data analysis: SAS, Scala, Julia

Benefits of using programming language in data analysis:

- Clarifies the steps of your analysis
- Saves time
- Reproduce and share your work

Spreadsheets, SQL, and R all use filters and functions.

Key question	Spreadsheets	SQL	R
What is it?	A program that uses rows and columns to organize data and allows for analysis and manipulation through formulas, functions, and built-in features	A database programming language used to communicate with databases to conduct an analysis of data	A general purpose programming language used for statistical analysis, visualization, and other data analysis
What is a primary advantage?	Includes a variety of visualization tools and features	Allows users to manipulate and reorganize data as needed to aid analysis	Provides an accessible language to organize, modify, and clean data frames, and create insightful data visualizations
Which datasets does it work best with?	Smaller datasets	Larger datasets	Larger datasets
What is the source of the data?	Entered manually or imported from an external source	Accessed from an external database	Loaded with R when installed, imported from your computer, or loaded from external sources

Where is the data from my analysis usually stored?	In a spreadsheet file on your computer	Inside tables in the accessed database	In an R file on your computer
Do I use formulas and functions?	Yes	Yes	Yes
Can I create visualizations?	Yes	Yes, by using an additional tool like a database management system (DBMS) or a business intelligence (BI) tool	Yes

R: a programming language frequently used for statistical analysis, visualization, and other data analysis.

R is based on another language called S

R is:
- Accessible
- Data-centric
- Open source
- Has active community of users

Open source: code that is freely available and may be modified and shared by the people who use it.

Situations where you might use R:
- Reproducing your analysis
- Processing lots of data
- Creating data visualizations

Programs like RStudio, an interactive development environment (IDE) for programming in R, use the R Console and other tools to make it easier to write and execute R code

In RStudio, the R Console is often referred to as the console pane

Integrated Development Environment: a software application that brings together all the tools you may want to use in a single place.

Console is the place where you give commands to R.
- commands will be forgotten when you log off

Source editor coding allows you to save your script executing code in the editor causes it to automatically appear in the console.

R is case sensitive

R packages: custom solutions to data problems created by R users

RStudio makes it easy to reproduce your work on different datasets. When you input your code, it's simple to just load a new dataset and run your scripts again. You can also create more detailed visualizations using RStudio.

Module 2

Fundamentals of programming using R
Basic Concepts of R

- Functions (R): a body of reusable code used to perform specific tasks in R.
  - Usually followed by one or more arguments in parentheses
  - Argument (R): information that a function in R needs in order to run.
  - To find out more about functions in R type ?functionname() and hit command + return and you will get more info on that function in the Help window.
  - Functions are case sensitive
- Variables (R): a representation of a value in R that an be stored for use later during programming.
  - When naming a variable in R you can use a short phrase.
  - A variable name should start with a letter and can also contain numbers and underscores. Also case sensitive and should be all lower case as often as possible.
  - <- is the assignment operator for variables. It looks like an arrow pointing from the value to the variable.
    - There are others that work too, but it’s always good to stick with one consistently in your code.
- Comments (R): helpful when you want to describe or explain what’s going on in your code.

    - Use them as much as possible so everyone can understand.
    - Use # for comments in R
- Data types: logical, date, date/time, string, numerical
- Vectors (R): a group of data elements of the same type stored in a sequence in R.
    - One way to make vectors is using the combine function: c(values, you_want, inside, your_vector)
        - The combine function combines multiple values into a vector.
    - We can use a stored vector anywhere in our analysis by using its variable name.
        - The values in the vector will automatically be applied to our analysis.
    - Can't have a vector that includes both logicals and numerics.
- Pipe (R): a tool in R for expressing a sequence of multiple operations; represented with "%>%"
    - Used to apply the output of one function into another function.
    - Pipes can make your code easier to read and understand.
    - Pipes make a sequence of code easier to work with and read

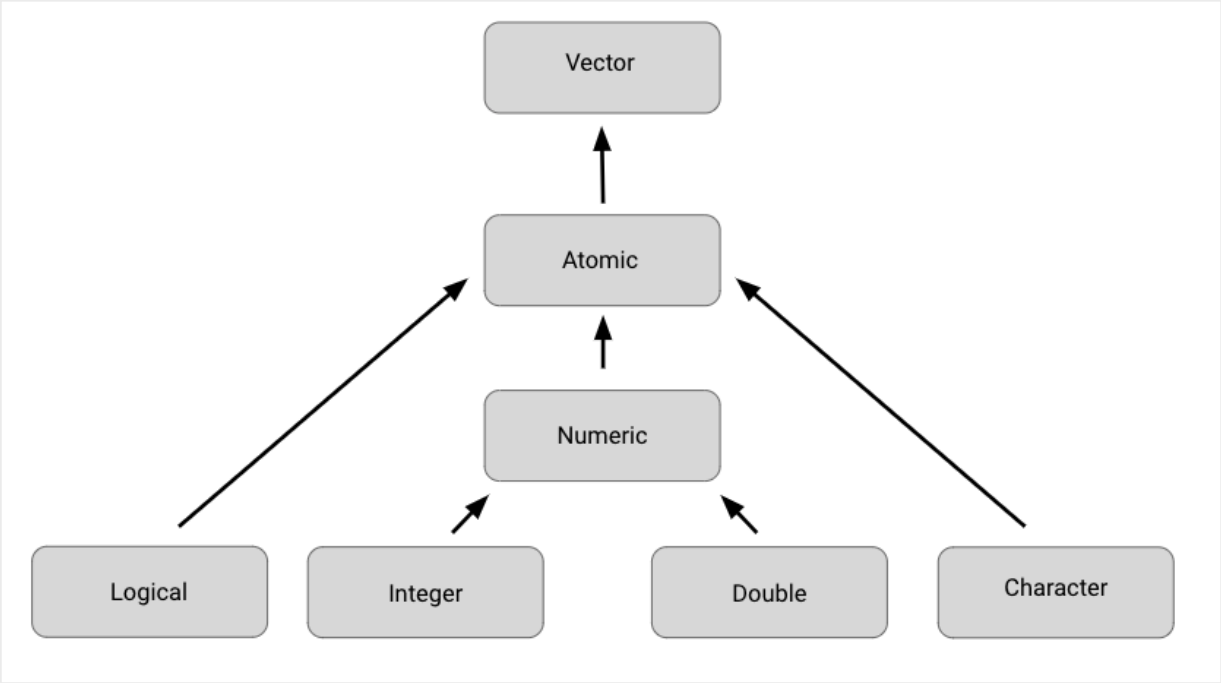
data structure: a format for organizing and storing data

- Most common types for R are vectors, data frames, matrices, and arrays.

Two types of vectors: atomic vectors and lists.

- Atomic vectors
    - Logical: True/False
    - Integer: Positive and negative whole values
        - To create a vector of integers you must place the letter "L" directly after each number.
    - Double: Decimal values
        - Integer and double vectors are known as numeric vectors because they both contain numbers.
    - Character: String/character values
    - Complex: Not as common in data analysis.
    - Raw: Not as common in data analysis.

Hierarchy of relationship between vectors:



Every vector will have two key properties: type and length.

- You can determine the type of vector you are working with using the **typeof()** function. Place code inside these parentheses and run it. It will tell you the type.
  - typeof(c(1L , 3L))
  - #> [1] "integer"
- the **length()** function will tell you the length of an existing vector. Length meaning the number of elements it contains.
  - x <- c(33.5, 57.75, 120.05)
  - length(x)
  - #> [1] 3

- You can also check if a vector is a specific type using **is** function: is.logical(), is.double(), is.integer(), is.character().
  - Returns TRUE
    - x <- c(2L, 5L, 11L)
    - is.integer(x)
    - #> [1] TRUE
  - or FALSE
    - y <- c(TRUE, TRUE, FALSE)
    - is.character(y)
    - #> [1] FALSE
- **names()** function allows you to name elements of a vector. Useful for writing readable code and describing objects in R.
  - All types of vectors can be named.

    - names(vector_name) <- (“a”, “b”, “c”)

atomic vectors can only contain elements of the same type. Lists allow you to store elements of different types.

Lists: their elements can be of any type, can even contain other lists.

- create with **list()** function. Similar to combine function c(), the list() function is just list followed by the values you want in your list inside ().
    - list("a", 1L, 1.5, TRUE)
- Use the **str()** function to find out what types of elements a list contains.
    - str(list("a", 1L, 1.5, TRUE))
- Like vectors, they can be named. You can name the elements of a list when you first create it with the list() function.

Dates and times in R using lubridate package
In R, there are three types of data that refer to an instant in time:

- A date **("2016-08-16")**
    - To get the current date you can run the **today()** function.
- A time within a day **(“20:11:59 UTC")**
- And a date-time. This is a date plus a time **("2018-03-31 18:15:48 UTC")**
    - To get the current date-time you can run the **now()** function.
    - UTC stands for Universal Time Coordinated
- When working with R, there are three ways you are likely to create date-time formats:
    - From a string
    - From an individual date
    - From an existing date/time object
- R creates dates in the standard yyyy-mm-dd format by default.

Converting from strings using lubridate: First, identify the order in which the year, month, and day appear in your dates. Then, arrange the letters y, m, and d in the same order. That gives you the name of the lubridate function that will parse your date.

- ymd(“2021-01-20”)

The **ymd()** function and its variations create dates. To create a date-time from a date, add an underscore and one or more of the letters h, m, and s (hours, minutes, seconds) to the name of the function:
ymd_hms("2021-01-20 20:11:59")

You can use the function **as_date()** to convert a date-time to a date.

Files: used to access and store data and related information.

Data frames: the most common way of storing and analyze data in R.

- A collection of column containing data, similar to a spreadsheet or SQL table.
- Each column has a name that represents a variable and includes one observation per row.
- Data frames summarize data and organize it into a format that is easy to read and use.
  - Can include many different types of data like numeric, logical, or character.
  - Can only have one elements in each cell
  - Each column should be named
  - Each column should consist of elements of the same data type.
- If you need to manually create a data frame in R, you can use the **data.frame()** function.
  - The data.frame() function takes vectors as input.
    - Example: data.frame(x = c(1, 2, 3) , y = c(1.5, 5.5, 7.5))
- Use the extract operator to extract a subset from a data frame. When you use this operator on a data frame, it takes two arguments: the row(s) and column(s) you'd like to extract, separated by a comma.
  - Example if data.frame is named "z": z[2, 1] to extract second row, first column

Working with files

- file.create() to create a blank file
  - Place the name and type of file in the ()
  - Files will usually be .txt, .docx. or .csv.
  - If successfully created will return value TRUE
- file.copy() to copy a file
  - name of file to be copied goes in () along with ", "name_of_destination_folder"
    - Example: file.copy("new_text_file.txt", "destination_folder")
- Delete files with unlink() function
  - Example: unlink("some_.file.csv")

Matrices are a collection of two-dimensional data elements that can only contain one data type.

Operator: a symbol that names the type of operation or calculation to be performed in a formula.

Assignment operators: used to assign values to variables and vectors.

Arithmetic operators: used to complete math calculations

Logical operators: return a logical data type such as TRUE or FALSE

- There are three primary types of logical operators:
  - AND (sometimes represented as & or && in R)
  - OR (sometimes represented as | or || in R)
  - NOT (!)

Conditional statement: a declaration that if a certain condition holds, then a certain event must take place.

- Example: For example, “If the temperature is above freezing, then I will go outside for a walk.” If the first condition is true (the temperature is above freezing), then the second condition will occur (I will go for a walk).
- if()
- else()
- else if()

if()

```
if (condition) {
 expr
}
```

else: The else statement is used in combination with an if statement.

```
if (condition) {
  expr1
} else {
 expr2
}
```

else if: The else if statement comes in between the if statement and the else statement.

```
if (condition1) {
 expr1
} else if (condition2) {
 expr2
} else {
 expr3
}
```

Packages are a key part of working with R

- They contain bundles of code called functions.

CRAN: Comprehensive R Archive Network
install.packages() allows you to download and install packages in R
library(package_name) loads the package
- the library function

One common function you can use to preview the data is the `head()` function
glimpse() works kind of like a schema in BigQuery
- same with str() or colnames()

rename() function allows you to rename columns or variables in your data.
- rename(diamonds, carat_new = carat, cut_new = cut)

summarize(): can generate a wide range of summary statistics for your data.
- summarize(diamonds, mean_carat = mean(carat))
- find mean, max, min, etc.

To build a visualization with `ggplot2` you layer plot elements together with a `+` symbol.

Packages (R): unites of reproducible R code
- Reusable R functions
- Documentation about the functions
- Sample datasets
- Tests for checking your code

R automatically includes Base R

CRAN (Comprehensive R Archive Network): an online archive with R packages, source code, manuals, and documentation.

Tidyverse (R): a system of packages in R with a common design philosophy for data manipulation, exploration, and visualization.

Conflicts happen when packages have functions with the same names as other functions.

8 core tidyverse packages
- ggplot2
- tibble
- tidyr
- readr
- purrr
- dplyr
- stringr

- forcats

You'll use them in almost every analysis.

Essential packages for workflow of data analysts:
- ggplot2: for data visualization; create a variety of data viz by applying different visual properties to the data variables in R.
- dplyr: offers a consistent set of functions that help you complete some common data manipulation tasks.
- tidyr: a package used for data cleaning to make tidy data.
- readr: used for importing data; most common is read_csv combined with column specification (often figured out automatically)

tibble works with data frames
purrr: works with functions and vectors
stringr: functions that make it easier to work with strings
forcats: provides tools that solve commons problems with factors.

Factors: store categorical data in R where the data values are limited and usually based on a finite group like country or year.

Pipes
Nested: in programming, describes code that performs a particular function and is contained within code that performs a broader function.

Pipes can be used to basically code the phrase "and then"

Nested functions read from the inside out
- i.e. arrange(filter(ToothGrowth, dose == 0.5), len) would filter first then arrange

Keyboard shortcut for pipe operators:
- ctrl+shift+m for PC and Chromebook
- cmd+shift+m for Mac

When using pipes:
- Add the pipe operator at the end of each line of the piped operation except the last one.
- Check your code after you've programmed your pipe.
- Revisit piped operation to check for parts of your code to fix.

Pipes or piping, and the functions that are part of the piping process, are building blocks for putting together analyses in R.

Module 3

Data frame: a collection of columns

- Columns should be named
- Data stored can be many different types, like numeric, factor, or character
- Each column should contain the same number of data items

Tibbles are like streamlined data frames

- Never change the data types of the inputs
- Never change the names of your variables
- Never create row names

Data frames and tibbles are the building blocks of analysis in R.

Tidy data (R): a way of standardizing the organization of data within R.

- Standards:
  - Variables are organized into columns
  - Observations are organized into rows
  - Each value must have its own cell

head() function gives us the first six rows of a data frame as a preview
str() is the structure function and it highlights the structure of a data frame.

- Gives high level info like the column names and the type of data contained in those columns.

colnames() can be used if we only want the column names instead.
mutate() function lets us make changes to our data frame. It's part of the dplyr package in the tidyverse.

There are three common sources for data in R:
- A`package` with data that can be accessed by loading that `package`
- An external file like a spreadsheet or CSV that can be imported into `R`
- Data that has been generated from scratch using `R` code

TIbbles are slightly different from standard data frames.

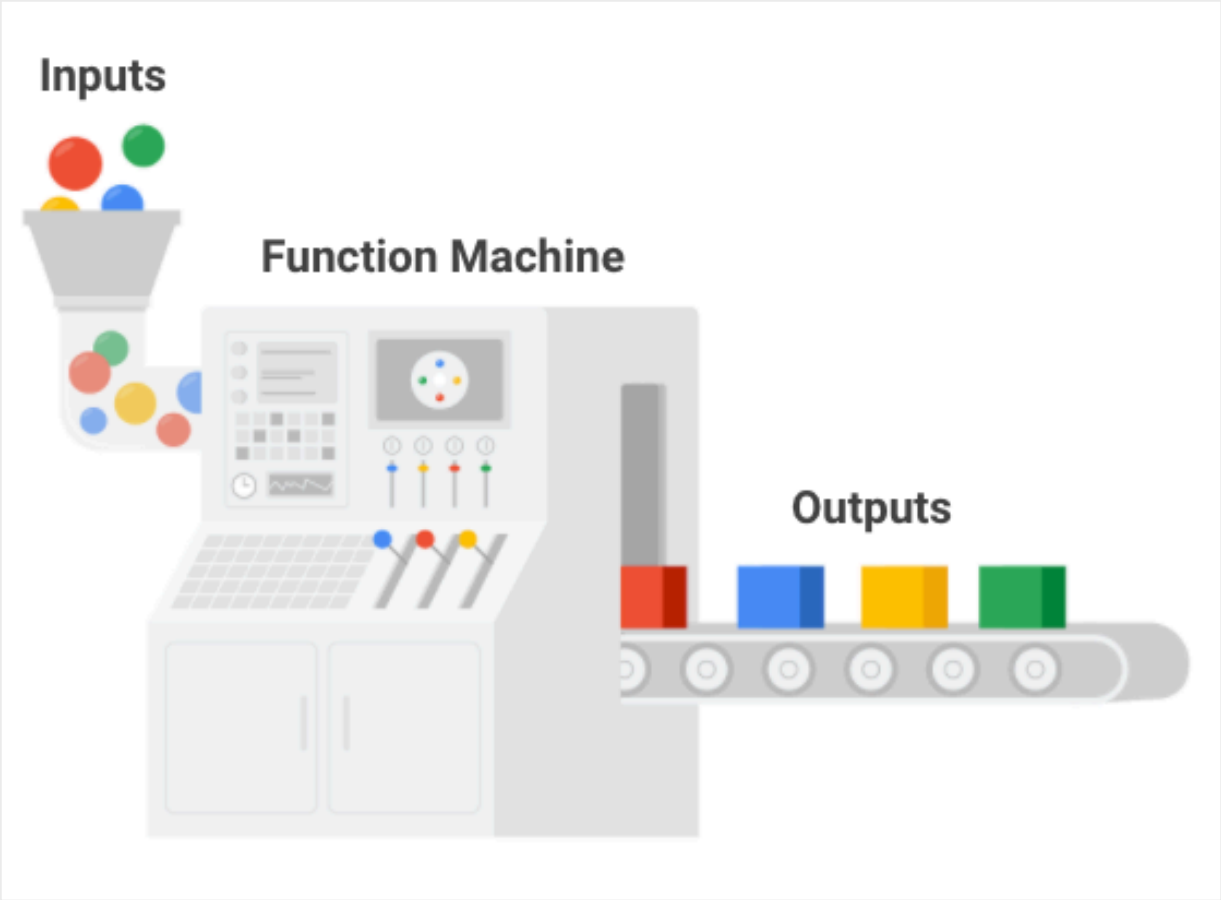
- Data frame has collection of columns like spreadsheet or SQL table. Tibbles are like streamlined data frames that automatically set to pull up only the first 10 rows of a dataset and only as many columns as can fit on the screen.
  - Tibbles are really useful when you're working with large sets of data.
- Unlike data frames, tibbles never change the names of your variables or the data types of your inputs.

- You can make more changes with data frames, but tibbles are easier to use.

view() function lets you add a data frame to the data viewer in RStudio.

Use as_tibble() to view a dataset as a tibble
- Makes large dataset easier to view and print.

Data import basics



Use the data() function to load a data set in R.
- If you leave the area between () blank, R will display a list of the available datasets.

The readr package in R is a great tool for reading rectangular data. Rectangular data is data that fits nicely inside a rectangle of rows and columns, with each column referring to a single variable and each row referring to a single observation.
- Files that store rectangular data:
  - .csv: (comma separated values): a .csv file is a plain text file that contains a list of data. They mostly use commas to separate (or

delimit) data, but sometimes they use other characters, like semicolons.
    - .tsv: (tab separated values): a .tsv file stores a data table in which the columns of data are separated by tabs. For example, a database table or spreadsheet data.
    - .fwf: (fixed width files): a .fwf file has a specific format that allows for the saving of textual data in an organized fashion.
    - .log: a .log file is a computer-generated file that records events from operating systems and other software programs.
- Base R has functions for reading files, but same functions in readr are typically much faster.
- readr functions:
    - read_csv(): comma-separated values (.csv) files
    - read_tsv(): tab-separated values files
    - read_delim(): general delimited files
    - read_fwf(): fixed-width files
    - read_table(): tabular files where columns are separated by white-space
    - read_log(): web log files
        - When you run one of these functions, R prints out a column specification that gives the name and type of each column. It also prints a tibble.
- readxl will read spreadsheets from Excel like readr does to csv and so on.
    - read_excel() is the function

skim_without_charts() gives us a pretty comprehensive summary of a dataset

glimpse()

head()

select() is useful for selecting just a subset of data from a large dataset.

rename() makes it easy to change column names
- syntax:

rename_with() can change column names to be more consistent
- rename_with(table_name, tolower) would make all column names lower case
    - in janitor package

clean_names(): ensure that there are only characters, numbers, and underscores

in names.

File-naming conventions

- should be accurate, consistent, and easy to read
- DO:
  - Keep your filenames to a reasonable length
  - Use underscores and hyphens for readability
  - Start or end your filename with a letter or number
  - Use a standard date format when applicable; example: YYYY-MM-DD
  - Use filenames for related files that work well with default ordering; example: in chronological order, or logical order using numbers first
- DON'T:
  - Use unnecessary additional characters in filenames
  - Use spaces or "illegal" characters; examples: &, %, #, <, or >
  - Start or end your filename with a symbol
  - Use incomplete or inconsistent date formats; example: M-D-YY
  - Use filenames for related files that do not work well with default ordering; examples: a random system of numbers or date formats, or using letters first

Four main types of operators in R:

- Arithmetic
- Relational
- Logical
- Assignment

Arithmetic: let you perform basic math operations like addition, subtraction, multiplication, and division.

Operator	Description	Example Code	Result/ Output
+	Addition	x + y	[1] 7
-	Subtraction	x - y	[1] -3
*	Multiplication	x * y	[1] 10
/	Division	x / y	[1] 0.4
%%	Modulus (returns the remainder after division)	y %% x	[1] 1
%/%	Integer division (returns an integer value after division)	y%/% x	[1] 2
^	Exponent	y ^ x	[1]25

Relational: also known as comparators, allow you to compare values.

Operator	Description	Example Code	Result/Output
<	Less than	x < y	[1] TRUE
>	Greater than	x > y	[1] FALSE
<=	Less than or equal to	x < = 2	[1] TRUE
>=	Greater than or equal to	y >= 10	[1] FALSE
==	Equal to	y == 5	[1] TRUE
!=	Not equal to	x != 2	[1] FALSE

Note that the [1] that appears before each output is used to represent how output is displayed in RStudio.

Logical: allow you to combine logical values.

Operator	Description
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR
!	Logical NOT

Assignment: let you assign values to variables.

Operator	Description	Example Code (after the sample code below, typing x will generate the output in the next column)	Result/ Output
<-	Leftwards assignment	x <- 2	[1] 2
<<-	Leftwards assignment	x <<- 7	[1] 7
=	Leftwards assignment	x = 9	[1] 9
->	Rightwards assignment	11 -> x	[1] 11
->>	Rightwards assignment	21 ->> x	[1] 21

arrange(): used to order data frame by a certain column

- arrage(column_name) does this in ascending order
- arrange(- column_name) does this in descending order

groupby(): Group by is usually combined with other functions. For example, we might want to group by a certain column and then perform an operation on those groups.

```
penguins %>%
  group_by(island) %>%
  drop_na() %>% # Drops any rows with missing data from the table
  summarize(mean_bill_length_mm = mean(bill_length_mm))
```

filter():

```
penguins %>%
  filter(species == "Adelie")
```

separate() and unite() can pull information in a column into two columns or make one column into two columns.

- syntax: separate(employee, name, into = c('first_name', 'last_name'), sep = ' ')
- syntax: unite(employee, 'name', first_name, last_name, sep = ' ')

pivot_longer and pivot_wider functions

Anscombe's quartet: four datasets that have nearly identical summary statistics

summarize(sd)standard deviation can help us understand the spread of values in a dataset and show us how far the value is from the mean

summarize(cor) correlation shows us how strong the relationship between two variables is.

In R, we can actually quantify bias by comparing the actual outcome of our data with the predicted outcome.

- The bias function finds the average amount that the actual outcome is greater than the predicted outcome. It's included in the sim design package.
- If model is unbiased, result should be pretty close to zero.
- If model is biased, result might be biased.
- Negative number means predicted outcome is greater than actual outcome
- Positive number means predicted outcome is less than actual outcome

sample() to inject a randomization element into our R programming.

- Can help address issues of bias.

Example of creating a new data from after cleaning, organizing, or manipulating data

```
hotel_bookings_v2 <-
  arrange(hotel_bookings, desc(lead_time))

max(hotel_bookings$lead_time)
min(hotel_bookings$lead_time)
mean(hotel_bookings$lead_time)

```{r group and summarize}
hotel_summary <-
  hotel_bookings %>%
  group_by(hotel) %>%
  summarise(average_lead_time=mean(lead_time),
            min_lead_time=min(lead_time),
            max_lead_time=max(lead_time))
```

Module 4

ggplot2 is most popular viz tool for analysts in RStudio

- created in 2005
- grammar of graphics

Benefits of ggplot2

- Can create different types of plots (scatter plots, bar charts, line
```

diagrams, more)
- Customize the look and feel of plots
  - Can change colors, layout, and dimensions and add text elements like titles, captions, and labels.
- Create high quality visuals
- Combine data manipulation and visualization

Aesthetics: a visual property of an object in your plot.
Geoms: the geometric object used to represent your data.
- points, bars, lines, etc.

Facets: let you display smaller groups, or subsets, of your data.
Labels and annotations: let you customize your plot

```
ggplot(data=penguins) +
  geom_point(mapping = aes(x = flipper_length_mm, y = body_mass_g))
```

Functions and arguments from ggplot2
Written in layers using + sign to add more layers

Aesthetic: a visual property of an object in your plot.
- Position, color, shape, or size.

Mapping: matching up a specific variable in your dataset with a specific aesthetic.

Basic steps to using ggplot function
1. Start with the ggplot() function and choose a dataset to work with
2. Add a geom_function to display your data
3. Map the variables you want to plot in the arguments of the aes() function

Create ggplot template

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

+ sign always goes at the end of a line of code

?<function_name> will give help for function that you have trouble with

R is case sensitive

Scatterplots are useful for showing the relationship between two numeric variables.

Mapping means matching up a specific variable in your dataset with a specific aesthetic.

All the code inside of the aes function tells R how to map aesthetics to variables.

You can map multiple aesthetics to the same variable

If we want to change appearance without regard to specific variables, code that outside of aes function

```
ggplot(data=penguins) +
  geom_point(mapping = aes(x = flipper_length_mm,
                           y = body_mass_g),
             color = "purple", shape = "triangle")
```

Most common aesthetics for points:

- X
- Y
- Color
- Shape
- Size
- Alpha

Geom: the geometric object used to represent your data.

- points, bars, lines, etc.
- geom_point: scatterplots
- geom_bar: bar charts
- geom_line: line graph

jitter: helps with over plotting where points overlay each other

When you use geom_bar R automatically counts the number of times the x variable appears in the data and shows it in the y-axis.

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```

Smoothing enables the detection of a data trend even when you can't easily notice a trend from the plotted data points.
Example code:

```
ggplot(data, aes(x=distance,
y= dep_delay)) +
    geom_point() +
    geom_smooth()
```

Loess smoothing: best for smoothing less than 1,000 points

```
ggplot(data, aes(x=, y=))+
  geom_point() +
  geom_smooth(method="loess"
```

Gam smoothing: (Generalized Additive Model) useful for large number of points.

```
ggplot(data, aes(x=, y=)) +
  geom_point() +
  geom_smooth(method="gam",
formula = y ~s(x))
```

Facets: let you display smaller groups, or subsets, of your data.

- Functions:
  - facet_wrap(): facet your plot by a single variable
  - facet_grid(): facet your plot by two variables. Will split the plot into facets vertically by the values of the first variable and horizontally by the values of the second variable.

Tilde operator is used to define the relationship between dependent variable and independent variables in a statistical model formula.

Facets help us focus on important parts of our data that we might not notice in a single plot. If your is visual is too busy, for example, if it's got too many variables or levels within variables, faceting can be a good option.

Filtering before plotting example

```
data %>%
   filter(variable1 == "DS") %>%
   ggplot(aes(x = weight, y = variable2, colour = variable1)) +
   geom_point(alpha = 0.3,  position = position_jitter()) + stat_smooth(method =
"lm")
```

Annotate: to add notes to a document or diagram to explain or comment upon it.

labs() function:

- titles: title = “Title Goes Here”
- subtitles: subtitle = “Subtitle goes here”
- captions: let us know the source of our data, auto displayed on bottom right of plot

annotate() function lets us add important info in the table itself

```
annotate(“text”, x = 22, y = 3500, label = “The Gentoos are the largest”)
```

Store a variable in R: variable name <- code

- then later you can just recall the variable and keep adding to it

Export option in plots tab of RStudio lets us save our plots
or you can use the ggsave() function

- ggsave() automatically saves the last plot you displayed
- ggsave("Three Penguin Species.png")

Module 5

R Markdown: a file formate for making dynamic documents with R.

- works as a code notebook: has code chunks, notes, documentation, reporting, etc.
- Great tool for documenting your analysis at any stage

Markdown is a syntax for formatting plain text files.

R Notebook: lets users run your code and who the graphs and charts that visualize the code.

Can convert to lots of different formats too.

- HTML, PDF, and Word documents
- Slide presentation
- Dashboard

HTML: the set of markup symbols or codes used to created a webpage.

Other notebook options:

- Jupyter
- Kaggle
- Google Colab

YAML: a language for data that translates it so it's readable.

- YAML originally stood for "yet another markup language"
- YAML Header section is called out using three dashes on first and last lines
- Basically for metadata, automatically included when you create a new file

Text is a way to comment on and explain your code and analysis and any visualizations you're including.

- Text can be formatted to include links, ordered lists, equations, and more.
- # are used for headers, the more #, the smaller the header
- Links go inside <> angled brackets.
- Bolded words or phrases go inside double asterisks **like this**
- Italicized words or phrases get one asterisk on both sides *like this*

- Embed a link like this: click here[link](http://rmarkdown.rstudio.com>).
- Bullet list get one asterisk on each line * like this
- Add graphic with ![Description of graphic](link to graphic)
  - ![Caption](https://png.pngtree.com/png-vector/20190419/ourmid/pngtree-vector-down-arrow-icon-png-image_956433.jpg)

Code chunk: code in an Rmd file

Delimiter: a character that indicates the beginning or end of a data item.

Code chunk delimiters: ```{r} to begin and ``` to end it
or go to Code menu and select Insert Chunk
or command + option + i

R Markdown renders files to specific presentation formats when you use the following output settings:

- beamer_presentation – for PDF presentations with beamer
- ioslides_presentation – for HTML presentations with ioslides
- slidy_presentation – for HTML presentations with Slidy
- powerpoint_presentation – for PowerPoint presentations
- revealjs : : revealjs_presentation – for HTML presentations with reveal.js (a framework for creating HTML presentations that requires the reveal.js package)

Output formats in R Markdown

You can save this reading for future reference. Feel free to download a PDF version of this reading below:
View PDF File

This reading will explore the different types of output formats you can produce with R Markdown.

Setting the output of an R Markdown document

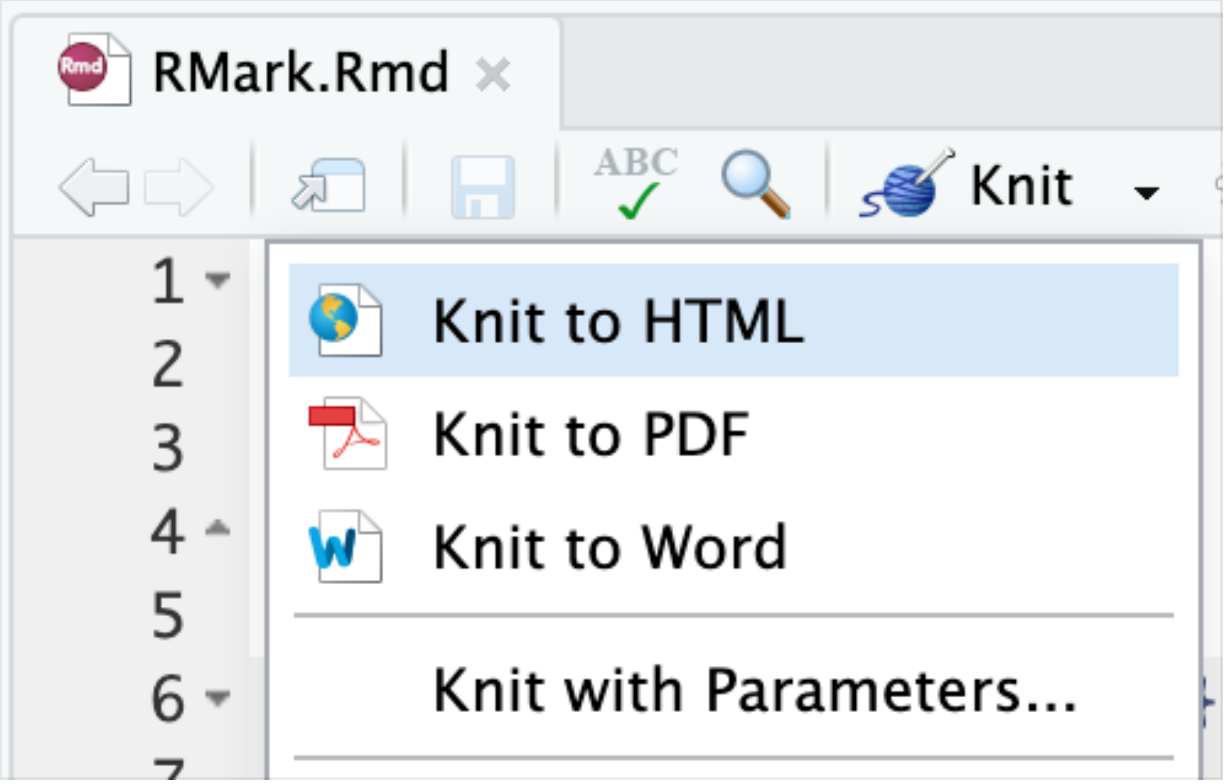
When working in RStudio, you can set the output of a document in R Markdown by changing the YAML header.
For example, the following code creates an HTML document:

```
---
title: "Demo"
output: html_document
---
```

And the following code creates a PDF document:

```
---
title: "Demo"
output: pdf_document
---
```

The Knit button in the RStudio source editor renders a file to the first format listed in its output field (HTML is the default). You can render a file to additional formats by clicking the dropdown menu next to the knit button.



Available document outputs

In addition to the default HTML output (html_document), you can create other types of documents in R Markdown using the following output settings:

- pdf_document – This creates a PDF file with LaTeX (an open source document layout system). If you don't already have LaTeX, RStudio will automatically prompt you to install it.
- word_document – This creates a Microsoft Word document (.docx).
- odt_document – This creates an OpenDocument Text document (.odt).
- rtf_document – This creates a Rich Text Format document (.rtf).
- md_document – This creates a Markdown document (which strictly conforms to the original Markdown specification)
- github_document – This creates a GitHub document which is a customized version of a Markdown document designed for sharing on GitHub.

For a detailed guide to creating different types of R Markdown documents, check out the Documents chapter in R Markdown: The Definitive Guide.

Notebooks
A notebook (html_notebook) is a variation on an HTML document (html_document). Overall, the output formats are similar; the main difference between them is that the rendered output of a notebook always includes an embedded copy of the source code.
Notebooks and HTML documents also have different purposes. HTML documents are good for communicating with stakeholders. Notebooks are better for collaborating with other data analysts or data scientists. To learn more, check out the section on Notebooks in the R Markdown documentation.

Presentations
You can also use R Markdown to produce presentations. Automatically inserting the results of your R code into a presentation can save you lots of time.
R Markdown renders files to specific presentation formats when you use the following output settings:

- beamer_presentation – for PDF presentations with beamer
- ioslides_presentation – for HTML presentations with ioslides
- slidy_presentation – for HTML presentations with Slidy
- powerpoint_presentation – for PowerPoint presentations
- revealjs : : revealjs_presentation – for HTML presentations with reveal.js (a framework for creating HTML presentations that requires the reveal.js package)

To learn more, check out the section on Slide Presentations in the R Markdown documentation.

Dashboards
Dashboards are a useful way to quickly communicate a lot of information. The flexdashboard package lets you publish a group of related data visualizations as a dashboard. Flexdashboard also provides tools for creating sidebars, tabsets, value boxes, and gauges. To learn more, visit the flexdashboard for R page and the Dashboards section in the R Markdown documentation.

Shiny
Shiny is an R package that lets you build interactive web apps using R code. You can embed your apps in R Markdown documents or host them on a webpage.
To call Shiny code from an R Markdown document, add runtime: shiny to the YAML header:

```
---
title: "Shiny Web App"
output: html_document
```

runtime: shiny

To learn more about Shiny and how to use R code to add interactive components to an R Markdown document, check out the Shiny tutorial from RStudio.