Course 7 - Data Analysis with R Programming

Week 1

Languages	R	Python
Common features	- Open-source - Data stored in data frames - Formulas and functions readily available - Community for code development and support	- Open-source - Data stored in data frames - Formulas and functions readily available - Community for code development and support
Unique advantages	- Data manipulation, data visualization, and statistics packages - "Scalpel" approach to data: *find packages to do what you want with the data*	- Easy syntax for machine learning needs - Integrates with cloud platforms like Google Cloud, Amazon Web Services, and Azure
Unique challenges	- Inconsistent naming conventions make it harder for beginners to select the right functions - Methods for handling variables may be a little complex for beginners to understand	- Many more decisions for beginners to make about data input/output, structure, variables, packages, and objects - "Swiss army knife" approach to data: *figure out a way to do what you want with the data*

- **Programming** - giving instructions to a computer to perform an action
- **Programming language -** words and symbols we use to write instructions
- **Syntax** - own set of rules for how these words and symbols should be used
- Coding is writing instructions to the computer in the syntax of a specific programming language
- Programming clarifies the steps of analysis, saves time, reproduce and share your work (code must be consistently clear, one line of code can apply multiple filters, code is automatically stored and can be reproduced years later)
- Data analyst - collects, transforms, and organizes data to draw conclusions, make predictions, and drive informed decision-making
    - ***Python*** - general purpose language, used for working with AI, creating virtual reality
    - **R** - offers convenient statistical features, useful for creating advanced viz
- Web designer - styling and layout of web pages - text, graphics, video
    - ***HTML5*** *(Hypertext Markup Language version 5)* - provides structure for web pages, used to connect to hosting platforms
    - ***CSS*** *(Cascading Styling Sheets)* - for web page design, controls graphic elements, page presentation on multiple devices (mobile, large screens)
- Mobile app developer - uses programming to create applications used on laptops, mobile phones, and tablets
    - ***Swift*** - for Apple platforms - open source scripting language - macOS, iOS, etc. goal is to make applications run faster
    - ***Javascript*** - developing online apps, web browsers; official language of Android

    - ***C#*** *(C-sharp)* - an object-oriented programming language that is widely used to create mobile apps in the .NET open source developer platform
        - Xamarin - mobile app platform owned by Microsoft - extends .NET platform for developers to create cross-platform mobile apps for IOS and Android
- Web app developer - designs and develops network applications used across the w*eb*
    - Java
    - Python
    - ***Ruby*** - general-purpose, object-oriented programming language used for web application development
        - Diff from *Ruby on Rails* - an open source web application framework that runs using Ruby.
    - ***PHP*** - a scripting language particularly suited for web application development; based on Perl, another programming language
- Game Developer - application developer who specializes in video game creation
    - C#
    - ***C++*** - an extension of the C programming language that is also used to create console games, like those for Xbox

- Tips for learning programming languages:
    - Define a practice project and use the language to help you complete it. This makes the learning process more practical and engaging.
    - Keep previous concepts and coding principles in mind. Many of these are transferable between programming languages. So, after you have learned one language, learning a second or third programming language tends to be much easier.
    - Create and keep good notes and cheat sheets in whatever format (handwritten or typed) that works best for you.
    - Create an online filing system for information that you can easily access while you work in various programming environments.

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 is based on another programming language named S, was used internally at Bell Labs - a scientific research facility
- R refers to the first names of its two authors from New Zealand
- R is now the preferred language of scientists, statisticians and data analysts
- It is accessible, data-centric, open-source and has an active community of users
- **Open source** - code that is freely available and may be modified and shared by the people who use it
- R can quickly process lots of data and reproduce and share every step of an analysis - also create data viz

- R is like the engine of a car, RStudio is like the accelerator, steering wheel and dashboard
- **Integrated Development Environment (IDE)** - a software application that brings together all the tools you may want to use in a single place
- RStudio is a type of IDE - R console is one part of it, it also includes an editor for writing code, tools for managing data and creating visuals

- Rstudio - 4 panes
    - Lower left pane - R console where you give commands to R
    - Upper left - source editor pane - use when working with R Scripts
        - you can write code using the console or source editor - typing commands directly into console will be forgotten when you close current session; saving script in editor, you can access work again
        - Source editor and console work together, executing code in editor, code will automatically appear in console
    - Upper RIght - Environment pane - find all the data you have currently loaded and can easily organise and save it
        - If you import data from a spreadsheet, it'll be visible here
        - History tab - all your previous commands are saved here, most recent line of code is at bottom of list; copy any to console line by double-clicking
    - Lower right - pane that has tabs for files, plots, packages, help
        - Files - gives access to your file directory and shows the contents of the current working folder
        - Plots - if we create a plot, the result appears here eg. scatter plot
        - **Packages** - units of reproducible R code - custom solutions to data problems developed by R users
            - Has combination for code, reusable R functions, descriptive docs about the functions, tests for check code, sample datasets
        - **Tidyverse** - library of packages in R with a common design philosophy for data manipulation, exploration, and visualization
            - Can upgrade, install, manage your library under Packages
            - Packages loaded in current session have a check mark

- imagine you are analyzing sales data for every city across an entire country. That is a lot of data from a lot of different groups - each city has its own group of data
- Here are a few ways RStudio could help in this situation:
    - takes a specific analysis step and perform it for each group using basic code. In this example, could calculate the yearly average sales data for every city
    - allows for flexible data visualization, can visualize differences across the cities using plotting features like facets
    - automatically creates an output of summary stats—or even your visualized plots—for each group.

Week 2 - R Fundamentals

- **Functions (R) -** a body of reusable code used to perform specific tasks in R
    - **Argument** - info that a function in R needs in order to run
- **Variable(R)/Objects** - a representation of a value in R that can be stored for use later during programming
    - Should start with a letter and can contain number or underscores
    - **<- assignment operator**, it assigns the value to the variable
- **Comments - # -** to explain code or make it more readable
- **Data type** - numeric, text, logical, date, date time
- **Vector** - a group of data elements of the same type stored in a sequence in R
    - **c(**x,y,z**)** - values you want in your vector
    - Assign a variable to it and you can return the values anywhere in the analysis

- **Pipes** - make a sequence of code easier to work with and read
    - Expresses a sequence of multiple operations, represented with **%>% - cmd+shift+m**
    - Used to apply the output of one function into another function
    - Makes programming more efficient and less cluttered as opposed to typing everything out or usings nests
- When using pipes:
    - Add the operator at the end of each line of the piped operation except the last one
    - Check your code after you've programmed the pipe
    - Revisit piped operations to check for parts of your code to fix

- **Nested** - code that performs a particular function and is contained within code that performs a broader function (and then)
        - Eg. call up data (and then) group the data (and then) summarize with mean function
- **Nested function** - a function that is completely contained within another function
        - Read from inside

- Eg. typing out:
    - filtered_tg <- filter(ToothGrowth, dose == 0.5)
      arrange(filtered_tg,len)

- Eg. nested:
    - arrange(filter(ToothGrowth, dose == 0.5), len)

- Eg. Pipes: (R automatically indents) call the dataset and then filter and then sort
    - filtered_toothgrowth <- ToothGrowth %>%
      filter(dose==0.5) %>%
      arrange(len)

- Eg. adding more to Pipe without changing the code - find avg tooth length(len) for each of the two supplements in the study when dose = 0.5
    - filtered_toothgrowth <- ToothGrowth %>%
      filter(dose==0.5) %>%
      group_by(supp) %>% *groups results with the 2 supps.*
      summarize (mean_len = mean(len,na.rm = T), .group = "drop")
      arrange(len)
    - *Na.rm = True - the function skips over any NA values*

- **Data structure** - a format for organizing and storing data
    - most common types in R - Vectors, data frames, matrices, arrays\
- **Vectors, 2 types:**

1. **Atomic vectors** - 6 primary types (logical, integer, double, character, complex, raw - last two not used in data analysis

Type	Description	Example
Logical	True/False	`TRUE`
Integer	Positive and negative whole values	`3`
Double	Decimal values	`101.175`
Character	String/character values	`"Coding"`

    - Creating vectors - c() - **combine function**
        - Numeric - c(2.5, 48.5,101.5)
        - Integers - must place L after each number - c(1L, 5L,15L)
        - Characters or logicals - c("Sara", "Lisa", "Anna"); C(True, False,False)
    - Properties of vectors - type and length
        - Can determine type with **typeof() function** - eg. typeof(c(1L,3L)) will yield "integer"
        - Determine number of elements using **length() function** - eg. x<-c(33.5,57.75,120.05), length (x), will yield 3
        - **Is function - is.logical(), is. double(), is. integer()** - will return true or false - eg. x<-c(2L,5L,11L0 is.integer(x0, will yield True
    - Naming vectors - use **names() function**:
        - names(x) <- c("a", "b", "c"); when you run the code R will show that the first element of the vector is named a, second b, third c

2. **Lists** - different from atomic vectors - they can be any type - dates, data frames, vectors, etc.
    - **List() function -** list("a", 1L, 1.5 TRUE)
        - Lists can contain other lists - list(list(list(1,3,5)))
    - Structure of lists - **str(function)**

        - Eg. str(list("a", 1L, 1.5, TRUE)) will yield:
          #> List of 4
          #> $: chr "a"
          #> $: int 1
          #> $: num 1.5
          #> $: logi TRUE
        - The **$ symbol** reflects the nested structure of the list
    - Naming lists - you can name the elements of a list when you first create it with the list() function:
        - list('Chicago' = 1, 'New York' = 2, 'Los Angeles' = 3)
- Use **lubridate** package to work with dates and times in R
- In R, there are three types of data that refer to an instant in time:
    - A date ("2016-08-16")
        - Default is yyyy-mm-dd
    - 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")
        - UTC - Universal Time Coordinated - the primary standard by which the world regulates clocks and time
- **today() function** - gives you year, month and day
- **now() function** - gives you current date and time to the nearest second
- 3 ways to create date-time formats:
    1. From a string - **ymd() function, mdy(), dmy() -** input date and it will return the yyyy-mm-dd format
    2. From an individual date - add an underscore and one or more of the letters h,h,s to the name - **ymd_hms()** or mdy_hm()
    3. From an existing date/time object - convert date-time to date - **as_date() function**
- **Data frame -** collection of columns - similar to a spreadsheet or SQL table, most common way of storing and analyzing data in R
    - Columns should be named
    - Can include many diff types of data (numeric, logical, character)
    - Elements in the same column should be of the same type
    - Use **data.frame() function** to manually create one: use vectors as input
        - Eg. data.frame(x=c(1,2,3), y=c(1.5,5.5,7.5)) - x and y are the names of the column, the elements are in the parentheses

```
  x y
1 1 1.5
2 2 5.5
3 3 7.5
```

-

- **R documentation** - a tool that helps you easily find and brows docs of almost all R packages on CRAN

- **Dir.create () function** to create a new folder, directory to hold your files, place name of folder in parentheses
    - Eg. dir.create("destination folder")
- **file.create()** - creates a blank file
    - Eg. file.create("new_text_file.txt., docx, or .csv.")
    - Widely accepted naming conventions:
        - Use underscores and hyphens for readability
        - Start or end with letter or number
        - Use standard date when applicable yyyy-mm-dd
        - only lower case, numbers, underscores, periods
        - Chronological or logical order
        - clear, concise, meaningful, reasonable length
- If the file is successfully created, R will return TRUE

- **file.copy()** to copy a file - name to be copied, name of destination folder
    - Eg. file.copy ("new_text_file.txt", "destination_folder")
- **unlink()** function to delete files

- **matrix/matrices** - a 2D collection of data elements - it has both rows and columns (vector is 1D)
    - Like vectors, can only contain a single data type
    - Use **matric() function** to create -first add vector which contains the values you want to place in the matrix, next add at least one matrix dimension
    - Specify number of columns or rows by using **nrow = or ncol =**
        - Eg. matrix(c(3:8), nrow =2) - creates a 2x3 that contain numbers 3-8

```
     [,1] [,2] [,3]

[1,]    3    5    7

[2,]    4    6    8
```

        -
        - Specify number of columns - matrix(c(3:8), ncol = 2)

```
     [,1] [,2]

[1,]    3    6

[2,]    4    7

[3,]    5    8
```

        -

- **Operators** - a symbol that names the type of operation ro calculation to be performed in a formula

1. **Arithmetic operators** - used to complete math calculations

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

2. **Relational Operators** - allows you to compare values

```
x <- 2

y <- 5
```

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

3. **Logical operators** - returns a logical data type such as TRUE or FALSE (boolean)
   - 3 primary types - AND, OR, NOT
   - **AND (& or &&)** - returns true only if both values are TRUE
     - **& (element-wise logical AND) -** difference comes when dealing with vectors - this examines all elements by comparing the first element of the first vector with the first element of the second vector etc

```
x <- c(3, 5, 7)

y <- c(2, 4, 6)
```

Then run the code with a single ampersand (&). The output is boolean (TRUE or FALSE).

```
x < 5 & y < 5
```

-

```
[1]  TRUE FALSE FALSE
```

- **&& (Logical AND)** - only examines first element of each vector

```
x < 5 && y < 5
```

```
[1] TRUE
```

- **OR (I or II) -** returns true if at least one value is TRUE
  - Element-wise logical OR vs logical OR - same diff as above
- **NOT (!)** - negates the logical value it applies to !TRUE would get FALSE
  - Zero is considered FALSE, non-zero numbers are taken as TRUE

For example, apply the NOT operator to your variable `(x <- 10)`:

```
!(x < 15)
```

```
[1] FALSE
```

The NOT operation evaluates to `FALSE` because it takes the opposite logical value of the statement `x < 15`, which is TRUE (10 is less than 15).

4. **Assignment operators** - lets you assign values to variables, best practice in R is single arrow left assignment

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

- **Conditional statements -** declaration hat if a certain condition holds, then a certain event must take place
- **if()** - sets a condition and if the condition evaluates to TRUE, the R code associated with the if statement is executed
  - Put condition in () **parentheses**, put code that has to be executed in {} **curly braces**

For example, let's create a variable *x* equal to 4.

```
x <- 4
```

Next, let's create a conditional statement: if *x* is greater than 0, then R will print out the string `"x is a positive number".`

```
if (x > 0) {
  print("x is a positive number")
}
```

- If False, a blank line will appear

- **else()** - used in combination with if, executed when the condition of the if statement is not True

```
x <- -7

if (x > 0) {

 print("x is a positive number")

} else {

 print ("x is either a negative number or zero")

}

[1] "x is either a negative number or zero"
```

-

- **Else if()** - comes in between the if and the else statement
  - If (condition1) {expr1} else if (condition2) {expr2} else {expr3{
  - When you have 3 conditions

```
x <- -1

if (x < 0) {

 print("x is a negative number")

} else if (x == 0) {

 print("x is zero")

} else {

 print("x is a positive number")

}
```

-

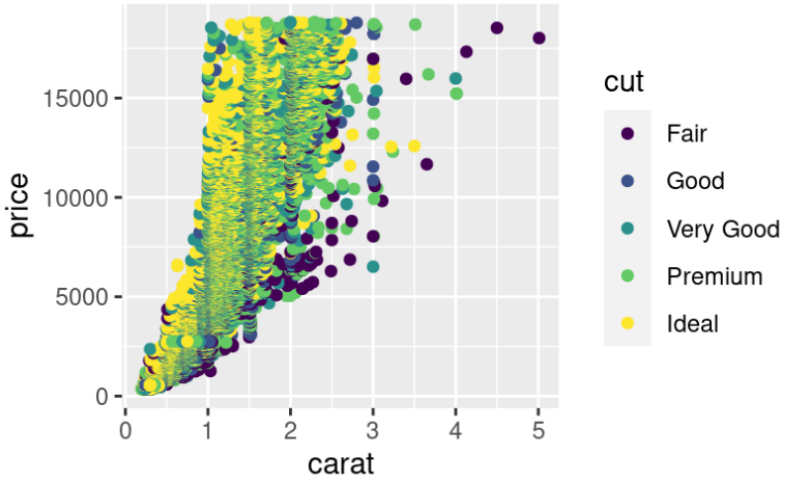
-

- **tidyverse()** - execute this code to install the package, has a collection of packages that can help you perform a wide variety of analysis tasks
- **library(**tidyverse**)** function to load the package
- **head(**name of dataset) function displays the column names and the first several rows of data
- **str(), glimpse()** - return summaries of each column in your data arranged horizontally
- **colnames()** - returns a list of column names from your dataset
    - Will show number in brackets, which helps you count the number of columns in your dataset : (depth is the 5th column, price is the 7th)

```
[1] "carat"   "cut"     "color"   "clarity"
[5] "depth"   "table"   "price"   "x"
[9] "y"       "z"
```

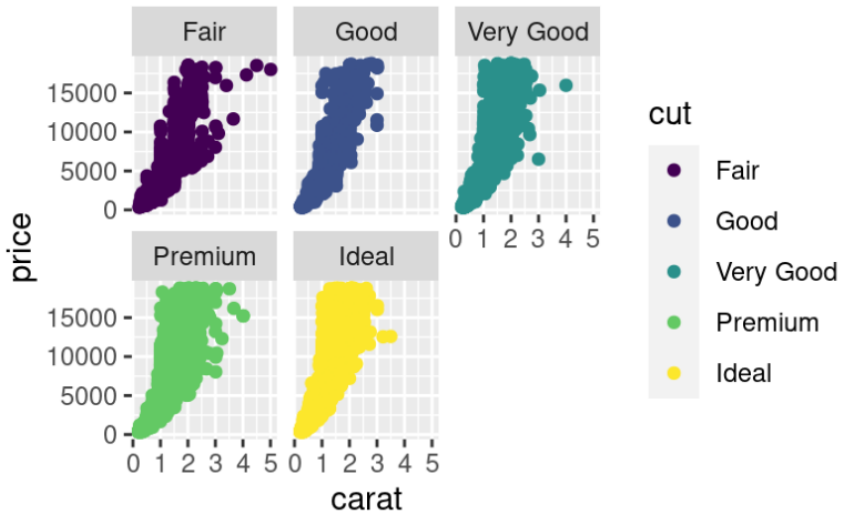
-
- **rename(**dataname, new_name = old_name, new_name2 = old_name2)
- **summarise()** - generate wide range of summary statistics for your data
    - Eg. summarise (diamonds, mean_carat = mean(carat))
- **Ggplot2** package - automatically loaded when you install and laid tidyverse,creates visualization
    - Eg. ggplot(data = diamonds, aes(x = carat, y = price)) + geom_point()
    - **geom_point()** - creates scatter plot
    - Can change the color of each point so that it represents another variable:

```
ggplot(data = diamonds, aes(x = carat, y = price, color
= cut)) +
  geom_point()
```



- **facet_wrap ()** function - creates a diff plot for each type of cut:

```
114 ggplot(data = diamonds, aes(x = carat, y = price, color
    = cut)) +
115   geom_point() +
116   facet_wrap(~cut)
117 ```
```



-

- **R markdown** - allows you to put code and writing in the same place
- Markdown is a simple language for adding formatting to text documents
- **Knit** button exports your work to a readable document for others

- Packages in R include reusable R functions, documentation about how to use the functions, sample datasets, and tests for checking your code
    - Found in repositories - collections of useful packages that are ready to install - **Bioconductor**, **R-Forge**, **rOpenSci**, or **GitHub**, most common - **CRAN**
        - **CRAN (Comprehensive R Archive Network) -** online archive with R packages, source code, manuals and docs
            - Makes sure any R content open to the public meets the required quality standards - since packages not in Base R are created by users
    - Includes the code itself but also the documentation (found in DESCRIPTION in CRAN) that explains the package's author, function
    - Tidyverse is a collection of R packages specifically designed for working with data - **8 core - ggplot2, tibble, tidyr, readr, purr, dplyr, stringr, forcats**
        - Helps you do everything from importing and transforming data to exploiting and visualizing it
        - **tidyverse_update()** to check for updates
        - If you just want to update one package, use the install.packages function otherwise use **update.packages** function - will take longer

- **install.package(name)** to install
- **installed.packages()** - shows you what's already installed in a chart
    - Package column - gives the name of the package
    - Priority column - what's needed to use functions from the package:
        - "Base R" - packages that are available to use in Rstudio by default (installed and loaded)
        - "Recommended" packages - installed but not loaded
            - Load using **library(name of package**)
            - If you see "conflict" after loading - when packages have functions with the same name as other functions - the last package loaded will be the function used
- Bottom right of Rstudio - packages section - includes brief description of each package
    - Checkmarked if loaded
    - Click on name of package and it opens help tab to show topics related
    - Can use help tab to find out what error messages and warnings mean
- Can also just google search packages

- **Vignette -** a documentation that acts as a guide to an R package
    - Shares details about the problem that the package is designed to solve and how the included function can help you solve it
- **browseVignettes** function allows you to read through vignettes of a loaded package
    - browseVignettes("packagename") - V must be capitalized

- 4 packages that are an essential part of the workflow for data analysts:
    - **Ggplot2** - creates a variety of data viz (specifically plots) by applying diff visual properties to the data variables in R
    - **Dplyr** - offers a consistent set of functions that help you complete some common data manipulation tasks (select, filter function)
    - **Tidyr** - package used for data cleaning to make tidy data
        - tidy/clean data - where every part of a data table or data frame is the right type in the right place
    - **Readr** - used for importing data
        - Most common - **(read_csv)** - imports a CSV file (contains data separated by commas in a table format) into R
        - Need to combine the funcion with a column specification, which describes how each column should be converted to the most appropriate data type
- Other 4 that are not used as often:
    - **Tibble** - works with data frames
    - **Purr -** works with functions and vectors helping make your code easier to write and more expressive
    - **Stringr** - includes functions that makes it easier to work with strings
    - **Forcats** - provides tools that solve common problems with factors

        - **Factors (R)** - stores categorical data in R where the data values are limited and usually based on a finite group like country/ year
- Best practices:
    - Structure for overall readability - not too verbose, overly complex, use comments
        - If you notice the same thing multiple times, or same logic and algorithm - should consolidate it and make it more concise
    - Document your work - explains in depth exactly what your code is doing, why it was built, purpose, limitations
    - Build for scalability - so it can be used to handle large data loads and small,
    - Build for dynamic **-** not hard coding any values that don't change when they need to

Week 3 - R Data Frames

- **Data frames -** a collection of columns with names and rows and cells with data (like spreadsheet, SQL table)
    - Columns should be named
    - Data stored can be many diff types (numeric, factor, character)
    - Each column should contain the same number of data items
- **Tibbles -** streamlined data frames
    - Difference - never change the data types of the inputs, the names of your variables, never create row names
    - Makes printing easier - won't overload console - automatically set to pull only the first 10 rows and as many columns that can fit on screen
    - **as_tibble()**

- **Tidy data** - a way of standardizing the organization of data within R:
    - Variables are organized into columns, observations into rows
    - Each value must have its own cell

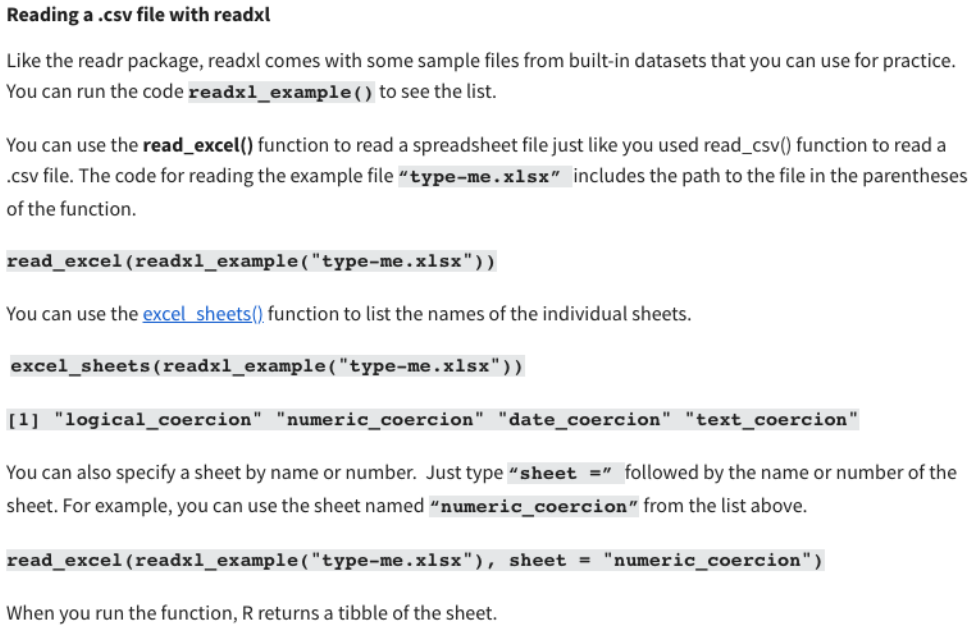
- **head() function** - gives just the first six rows and column names
- **str() -** summarizes the structure of this data frame - column name, data type, number of rows and variables
- **glimpse()** - shows how many rows, columns, columnname, data
- **mutate()** - changes data frame (part of dpylr package)
    - mutate(dataset_name, new_column = old_column+20)
      Eg. mutate(people, age_in_20 = age + 20)

- Creating a dataframe:
    - load packages - install.packages(tidyverse), library(tidyverse)
    - Create vector - names <-c(a,b,c), age<-(1,2,3)
    - Dataframe - people <- data.frame(names,age)

- **data()** - loads lists of datasets in R if you load it without an argument
- To load a specific dataset - data(mtcars) - it will also appear in environment pane - displays names of data objects, data frames and variables, number of observations, variables
- Just type name of dataset to preview it in R console, press control and enter or click on it in the environment pane

- **Readr package** - great tool for reading rectangular data (data that fits nicely inside a rectangle of rows and columns)
    - Files that store rectangular data:
        - **.csv (comma separated values)** - 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)** - file that 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)** - has a specific format that allows for the saving of textual data in an organized fashion.
        - **.log-** a computer-generated file that records events from operating systems and other software programs.
    - Faster than Base R for reading files
    - Part of the core tidyverse
- **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
- **readr_example() -** To generate sample files from built-in databases
- **read_csv(readr_example("mtcars.csv)) -** example of using read_csv to read a specific csv file - prints column specifications and a tibble

- **Readxl package** - transfer data from Excel into R, not a core tidyverse package, need to load using - **library(readxl)**



Reading a .csv file with readxl

Like the readr package, readxl comes with some sample files from built-in datasets that you can use for practice. You can run the code `readxl_example()` to see the list.

You can use the **read_excel()** function to read a spreadsheet file just like you used read_csv() function to read a .csv file. The code for reading the example file `"type-me.xlsx"` includes the path to the file in the parentheses of the function.

```
read_excel(readxl_example("type-me.xlsx"))
```

You can use the excel_sheets() function to list the names of the individual sheets.

```
excel_sheets(readxl_example("type-me.xlsx"))
```

```
[1] "logical_coercion" "numeric_coercion" "date_coercion" "text_coercion"
```

You can also specify a sheet by name or number. Just type `"sheet ="` followed by the name or number of the sheet. For example, you can use the sheet named `"numeric_coercion"` from the list above.

```
read_excel(readxl_example("type-me.xlsx"), sheet = "numeric_coercion")
```

When you run the function, R returns a tibble of the sheet.

- **Importing external data**:
  - Install and load tidyverse package
  - **Assign variable** to the project folder to **create data frame** :
    eg. bookings_df <- read_csv("hotel_books.csv")
    df_name<-read_csv("project folder)
  - Inspect data - head(), colnames(), str()
- new_df <- select(bookings_df, `adr`, adults) - example of creating new dataframe using two specific columns - average daily rate and adults
- Change data frame (but not original database) - mutate()
  - Eg. mutate(new_df, total = "adr"/ adults)

Cleaning data in R - helps you preview and rename data so that it's easier to work with:

- **here() package** - makes referencing files easier
- **skimr()** - makes summarizing data easier
- **janitor() -** has functions for cleaning data
- Install and load all 3
- **skim_without_charts()** - gives summary with name of dataset, number of rows and columns, column types, diff data types
- **select() -** specify or exclude certain columns:
  - Penguins %>%
    select(species)
  - Or Penguins%>%
    select(-species) - will include every column except species
- Penguins %>%
  rename(island name = island)
- **rename_with(**penguins, tolower) or toupper
  - Use so all columns are spelled and formatted correctly

- **clean_names()** - from janitor package - ensures there's only characters, numbers and underscores in the names

Organizational functions - sort, filter, and summarize data

- All part of core tidyverse
- **arrange()** - default is ascending, if want descending, add minus sign:
  - penguins%>% arrange(-bill_length_mm)
  - If want to save as data frame, need to name it:
  - Penguins2 <- penguins %% arrange(-bill_length_mm)
  - The function does not alter the existing data frame, need to use assignment operator to store the arranged data
  - Can also use max(data_frame$column_name), min()
    - Remember $ or you will get an error
- **group_by()** - usually combined with other functions - eg. group by certain column and then perform operation on those groups
  - Eg. penguins %>%
    group_by(island) %>%
    drop_na() %>%
    summarize(mean_bill_length_mm = mean(bill_length_mm))
    - **drop_na()** - leaves out NA values, be careful as it will remove rows from data
    - **summarize()** - gets high level info on data; mean_bill_length_mm is the new name
  - Eg. penguins%>% group_by(island) %>% drop_na() %>% summarize(max_bill_length_mm = max(bill_length_mm)) - to find the penguin with the longest bill in each island
  - Eg. penguins%>%
    group_by(species, island) %>%
    drop_na() %>%
    summarize(max_bl = max(bill_length_mm), mean_bl = mean(bill_length_mm))
    - Group by island and species, summarize to calculate both mean and max

```
# A tibble: 5 x 4
# Groups:   species [3]
  species   island    max_bl mean_bl
  <fct>     <fct>      <dbl>   <dbl>
1 Adelie    Biscoe      45.6    39.0
2 Adelie    Dream       44.1    38.5
3 Adelie    Torgersen   46      39.0
4 Chinstrap Dream       58      48.8
5 Gentoo    Biscoe      59.6    47.6
>
```

  - Eg. example_df <- bookings_df %>%
    summarize(number_canceled = sum(is_canceled),
    average_lead_time = mean(lead_time))
    - Calculate total number of canceled bookings and average lead time for bookings

- **filter()** - eg. penguins %>% filter(species == "Adelie") - only shows data that contains Adelie penguins, == means exactly equal in R

Transformational functions - help separate, combine data as well as create new variables:

- **separate()** - separate(employee, name, into = c("first_name", "last_name"), sep = " ")
    - separate(data_frame, column, into = c(“column1”, “column2”), sep = “with space or period”)
- **unite() -** combine data from diff columns, opposite of separate
    - unite(data_frame, “column_name”, which columns combining(no quotations needed), sep = “ “)
- penguins %>%
  mutate(body_mass_kg=body_mass_g/1000,
  flipper_length_m=flipper_length_mm/1000)
    - Added column for body mass in kg and flipper length in m
- **Wide data** - has observations across several columns, each column contains data from a diff condition of the variable
- **Long data** - has all the observations in a single column and variables in separate columns
- **pivot_longer()** - convert data to have less columns and more rows
    - Data %>%
      pivot_longer(names_to = “x”, values_to = “y”, range eg.(columnX:ColumnY)
- **pivot_wider()** - convert data to have more columns and fewer rows

- **Anscombe’s quartet** - 4 datasets that have nearly identical summary statistics
- quartet %>%
  group_by(set) %>%
  summarize(mean(x), sd(x), mean(y), sd(y), cor(x,y))
    - Standard deviation (sd) - explains the spread of values in a dataset, shows how far each value is from the mean
    - Correlation - how strong the relationship is between the two variables, closer to 1, stronger the positive correlation is, closer to -1, stronger the negative correlation is = 1 - two variables are perfectly correlated

- **Bias function** - finds the average amount that the actual outcome is greater than the predicted outcome
    - Unbiased - should be pretty close to 0
    - High negative number - biased - predicted outcome is larger
    - High positive number - biased - actual outcome is larger
    - In SimDesign package
    - Actual_sales <- (150,203,137,247)
      predicted_sales<-(200,300,150, 250)
      bias(actual_sales, predicted_sales)
- **sample() function** - allows you to add a random sample of elements

Week 4 - Visualization with R

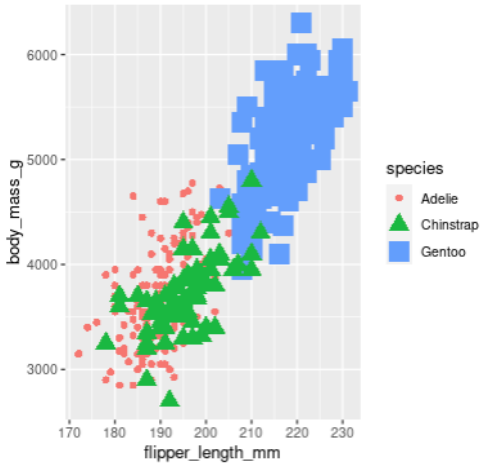
- Visualization packages - ggplot2 (Grammer of Graphics), Ploty, Lattice, RGL, Dygraphs, Leaflet, Highcharter, Patchwork, gganimate, ggridges
- **?function_name** - if and when you want to learn more about a function
- **Benefits of ggplot2:**
    - Create diff types of plots - scatter plots, bar charts, line diagrams
    - Customize the look and feel of plots - colors, layout, dimensions, add text elements - titles, captions, labels
    - Create high quality visuals
    - Combine date manipulation and visualization
- **Aesthetic (R)** - a visual property of an object in your plot (size, shape, color)
    - A connection or mapping between a visual feature in your plot and a variable in your data

Steps to create a plot in ggplot2:

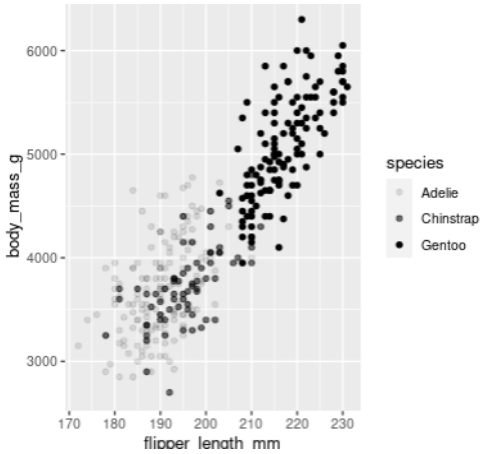
- ggplot(data = penguins) + geom_point(mapping = aes(x = flipper_length_mm, y = body_mass_g))
    - **ggplot(data=)** - tells R what data to use for your plot
    - **+** to add new layer, must always go at the end of the code
    - Choose a geom function - **geom_point(), geom_bar()**, etc.
        - Each geom function takes a **mapping** argument - matching up a specific variable in your dataset with a specific aesthetic
        - Mapping is always paired with the **aes function -** specifies which variables to map to x and y-axis

Enhancements with aesthetics:

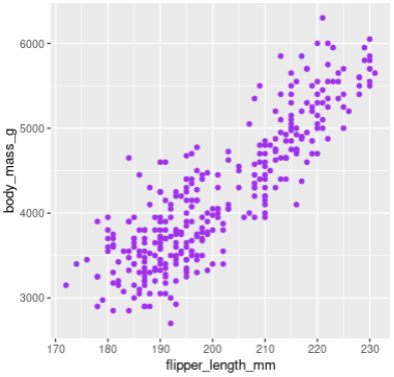
- ggplot(data=penguins) + geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g, **color=species, shape = species, size=species))**



- **Alpha = species** - controls the transparency of the points, good option when you have a dense plot with lots of data
- ggplot(data=penguins) + geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g, alpha=species))
-



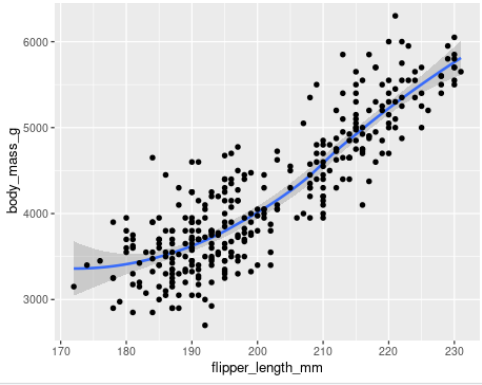
- Setting aesthetic apart from a specific value, set it outside of the aes function ggplot(data=penguins) + geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g), color = "purple")



-

- **Geom (R) -** the geometric object used to represent your data (points, bars, lines)
    - Geom_point - scatterplot
    - Geom_bar - bar graph
    - Geom_line - line graph
    - Geom_smooth - smooth line instead of scatterplot, useful to show general trends
    - Combining both to show relationship between trend line and data points:

- ggplot(data=penguins) + **geom_smooth**(mapping=aes(x=flipper_length_mm,y=body_mass_g))+ geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g))

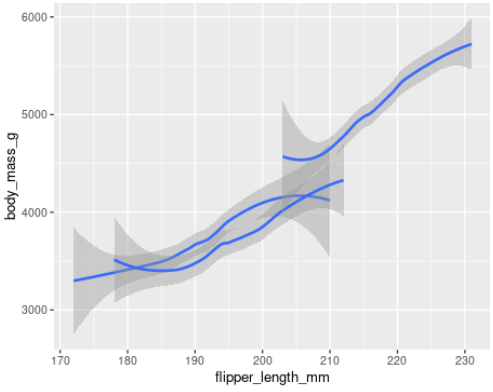


- **Smoothing -** enables detection of a data trend even when you can't easily notice a trend from the plotted data points, makes plots more readable
- **Loess smoothing** - plots with less than 1000 points
- **Gam**(generalized additive model) **smoothing** - useful for larger number of points

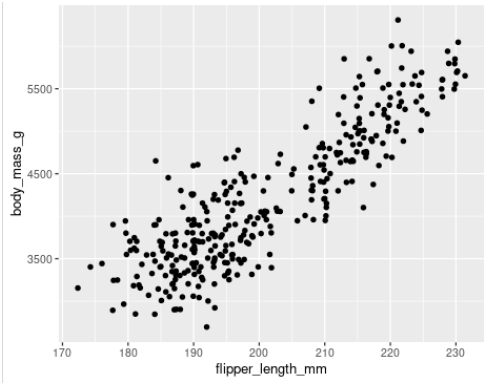
Example code
`ggplot(data, aes(x=, y=))+ geom_point() + geom_smooth(method="loess")`
`ggplot(data, aes(x=, y=)) + geom_point() + geom_smooth(method="gam", formula = y ~s(x))`

-

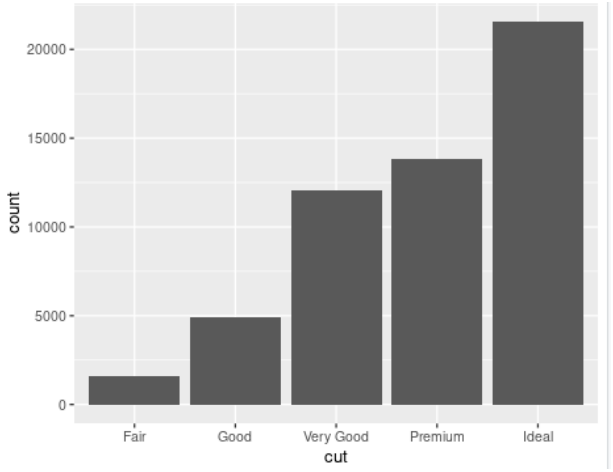
- **Line_type** - plots separate line for each species
- ggplot(data=penguins) + geom_smooth(mapping=aes(x=flipper_length_mm,y=body_mass_g, line_type = species))



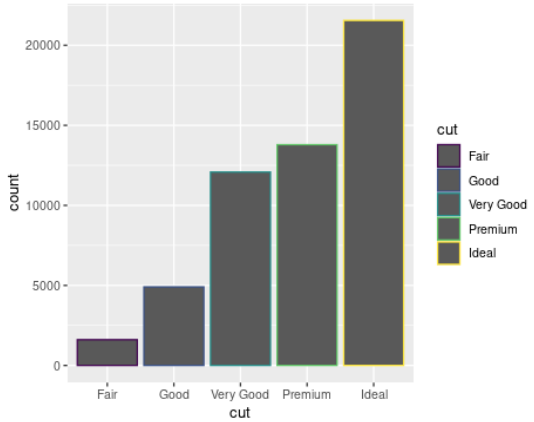
- **Geom_jitter -** creates scatter plot then adds a small amount of random noise to each point, helps with over-plotting (when data points in a plot overlap with each other); makes points easier to find
- ggplot(data=penguins) + geom_jitter(mapping=aes(x=flipper_length_mm,y=body_mass_g))



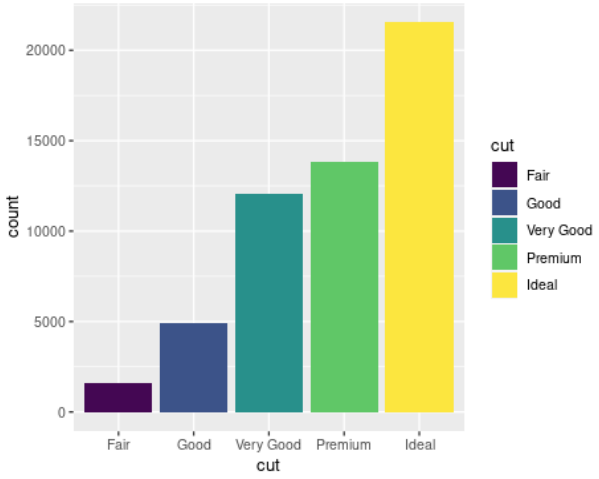
- Bar graphs - ggplot(data = diamonds) + **geom_bar(**mapping = aes(x = cut))
  - R automatically counts how many times each x-value appears in the data and shows the counts on the y-axis, so don't need y variable
  - X axis shows 5 categories of cut , y is the the number of diamonds in each

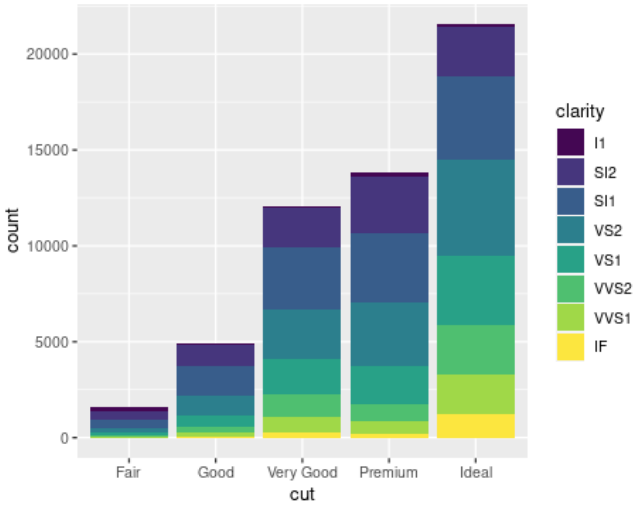


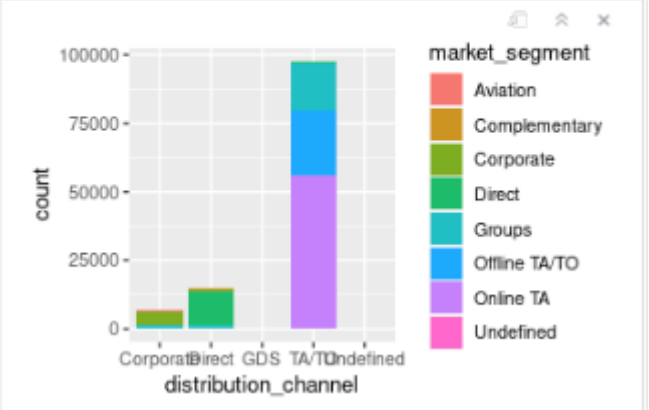
-
- ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, **color=cut**))
  - Only colors the outline:



-

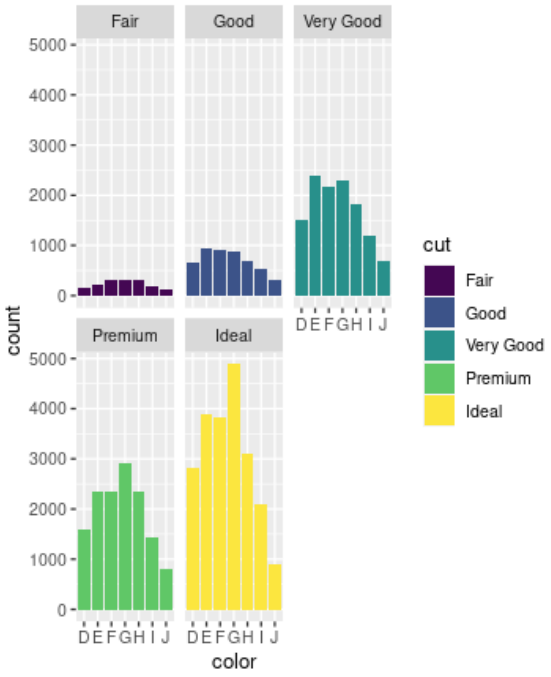
- ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, **fill=cut**))
    - Adds color to the inside of the bar:
    -


-
- **Stacked bar chart -** filling the map with another variable
    - Each combo has it's own colored rectangle, now we know the volume between cuts and figure out the difference in clarity within each cut
    - ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, **fill=clarity**))
    -


    -
```
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x =
distribution_channel, fill=market_segment))
```


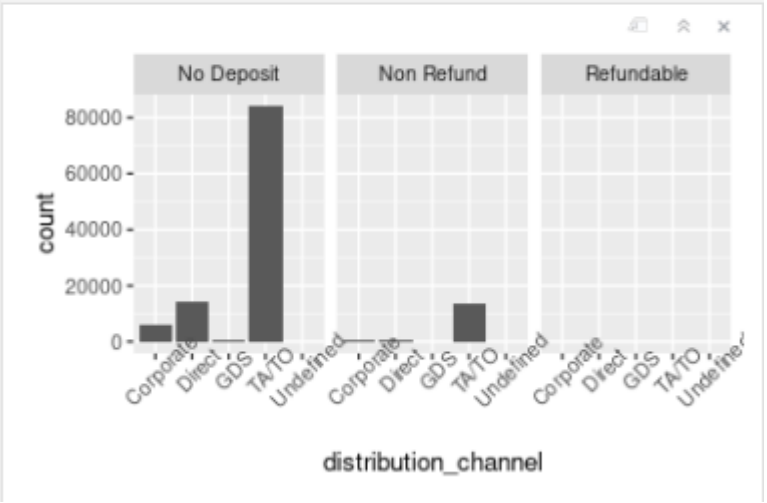
    - Eg. 2

- **Facets (R) -** displays smaller groups or subsets of your data - can separate plots for all the variables in your dataset
    - Helps reorganize data to show specific relationships between variables and reveal important patterns and trends
- **Facet_wrap** - facet plot with a single variable:
    - ggplot(data=penguins) +
      geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g, color=species))+
      facet_wrap(~species)
    - 

    - ggplot(data = diamonds) +
      geom_bar(mapping = aes(x = color, fill=cut))+
      facet_wrap(~cut)
    - 

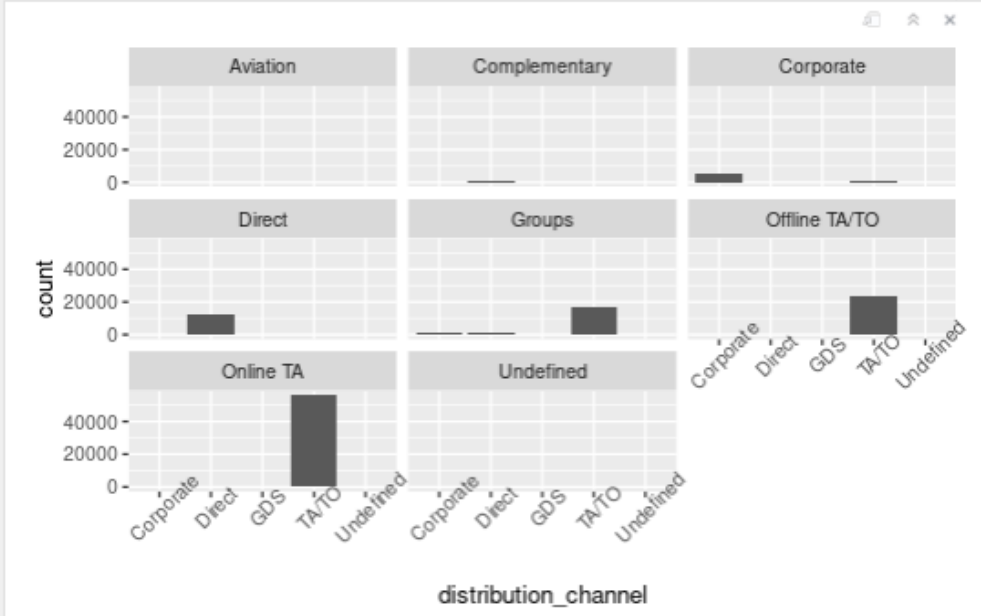
- **theme(axis.text.x = element_text(angle = 45)):**

```
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_wrap(~deposit_type) +
  theme(axis.text.x = element_text(angle = 45))
```



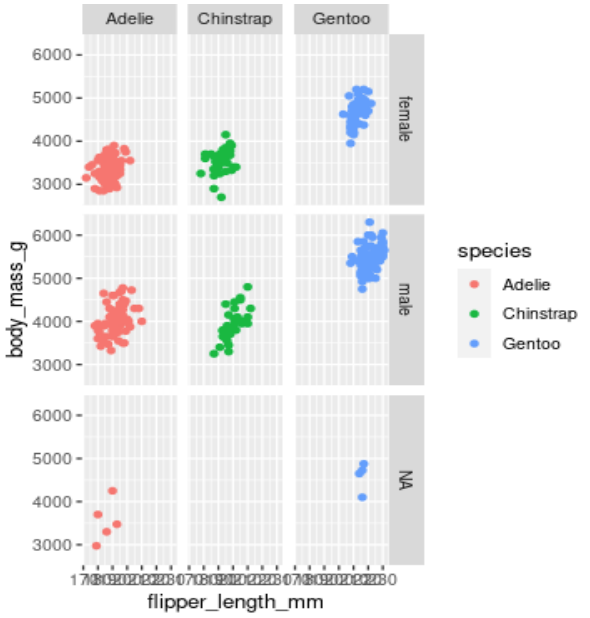
-

```
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_wrap(~market_segment) +
  theme(axis.text.x = element_text(angle = 45))
```



-

- **Facet_grid** - facet plot with two variables - splits the plot vertically by values of the first variable and horizontally by values of the second variable
    - ggplot(data=penguins) +
      geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g, color=species))+
      facet_grid(sex~species)



- Using the **filter function** from dplyr

```
```{r filtering a dataset with the pipe}
onlineta_city_hotels_v2 <- hotel_bookings %>%
  filter(hotel=="City Hotel") %>%
  filter(market_segment=="Online TA")
```
```

-

```
```{r filtering a dataset to just city hotels that are online TA}
onlineta_city_hotels <- filter(hotel_bookings,
                               (hotel=="City Hotel" &

hotel_bookings$market_segment=="Online TA"))
```
```

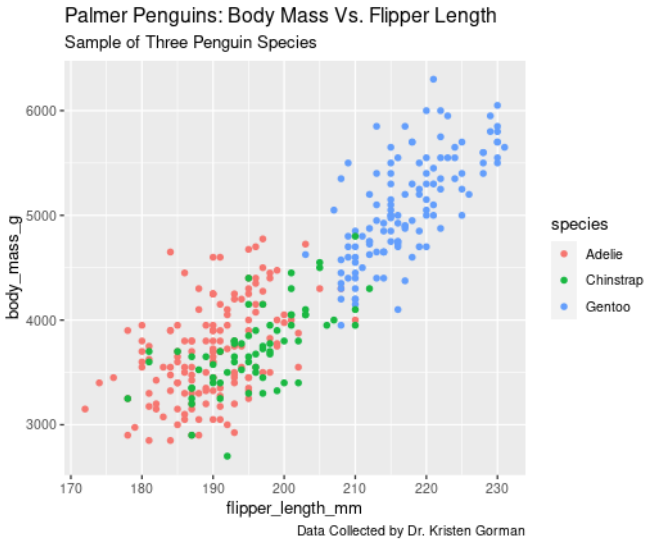
Note that you can use the '&' character to demonstrate that you want two different conditions to be true. Also, you can use the '$' character to specify which column in the data frame 'hotel_bookings' you are referencing (for example, 'market_segment').

-
- Eg.

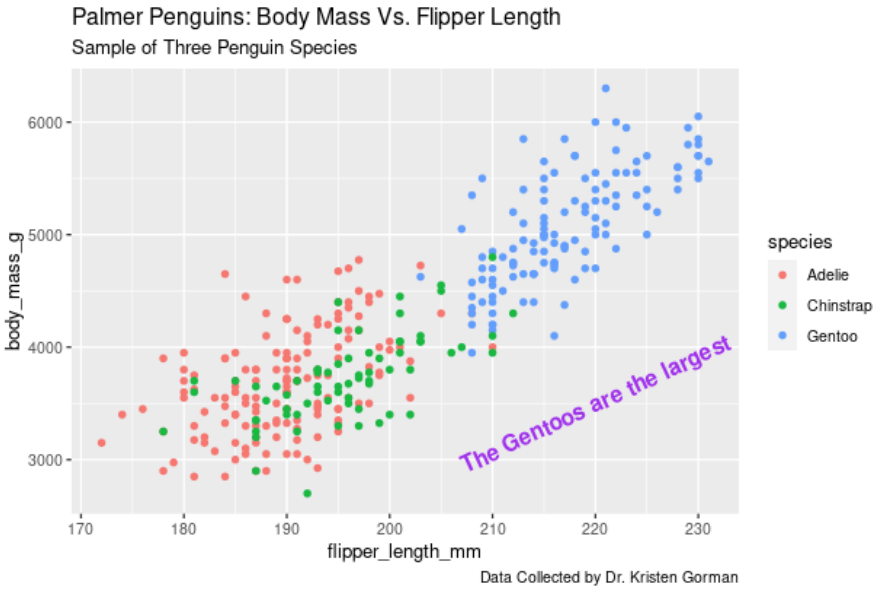
```
data %>%
    filter(variable1 == "DS") %>%
    ggplot(aes(x = weight, y = variable2, colour = variable1)) +
    geom_point(alpha = 0.3,  position = position_jitter()) +
stat_smooth(method = "lm")
```

-

- **Labels and annotations (R)** - customize plot with titles, subtitles, captions
- **Annotate** - add notes to a doc or diagram to explain or comment upon it
- **labs() function,** inside - **Title =, subtitle =, caption=**
    - ggplot(data=penguins) +
      geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g, color=species))+
      labs(title="Palmer Penguins: Body Mass Vs. Flipper Length", **subtitle =** "Sample of Three Penguin Species", **caption =** "Data Collected by Dr. Kristen Gorman")



-
- **annotate("label type", specific location (x and y coordinates, label name)**
    - ggplot(data=penguins) +
      geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g, color=species))+
      labs(title="Palmer Penguins: Body Mass Vs. Flipper Length", subtitle = "Sample of Three Penguin Species", caption = "Data Collected by Dr. Kristen Gorman") +
      annotate("**text**", **x=**220, **y=**3500, **label =** "The Gentoos are the largest", **color** = "purple", **fontface** = "bold", **size** = 4.5, **angle** = 25)



- Can use assignment operator to shorten code:
    - p <- ggplot(data=penguins) +
      geom_point(mapping=aes(x=flipper_length_mm,y=body_mass_g, color=species))+
      labs(title="Palmer Penguins: Body Mass Vs. Flipper Length", subtitle = "Sample of Three Penguin Species", caption = "Data Collected by Dr. Kristen Gorman")
    - p + annotate("text", x=220, y=3500, label = "The Gentoos are the largest")

- **Col = :**

Blue and yellow bars

To highlight underperforming products, use an aesthetics function: *col = ifelse (x<2, 'blue', 'yellow').*

- **Saving your visualizations -** export option in the plots tab or **ggsave -** defaults to saving the last plot you displayed
    - ggsave(“Name_viz. png”)

- 2. When creating a plot in ggplot you must set the mapping argument of a function. Which function has the mapping argument?

  - ○ The annotate function
  - ○ The ggplot function
  - ○ The aesthetic function
  - ◉ The geometric function

  ✓ Correct

Week 5 - Documentation and Reports

- **R Markdown** - a file format for making dynamic docs with R
    - Saves, organizes, documents every step of your analysis, summarize findings for stakeholders and team members
    - Can convert files to HTML, PDF, word doc, slide presentation, dashboard
- **Markdown** - syntax for formatting plain text files

- **R Notebook** - an interactive R Markdown Option - lets users run code from R Markdown doc and display charts and graphs to visualize the code generate shareable reports for stakeholders
    - Jupyter, Kaggle, Google Colab are Notebooks that do similar things
- File -> new file-> R Markdown - > Knit gives you HTML doc
    - Hashtags are used for headers ###Results - Header 3 style heading
    - Code chunks are run and the output appears in the HTML report
- **YAML** header section - first part of R Notebook
    - YAML is a language for data that translates it so it's readable to humans
    - 3 hyphens marks the beginning and end
    - Populates info you provide and other general info:

```
---
title: "Penguins Plots"
author: "DA Cert"
date: "2/26/2021"
output: html_document
---
```

    - Basically for metadata (data about the data in the rest of the file)
    - Output - change html to pdf, word
- Grey background is the code chunk - means it will be shown on your final report
    - All chunks begin and end with delimiters, start : ```{r}, end: ```
    - **Code chunk** - code added in an. Rmd file
        - Creates viz in the same doc as text, interactive for users
    - Cmd, Option, I to **add code** or click Add chunk command
    - Label code chunk will appear in the bottom of script pane
    - White settings button to customize what output to display
- **Inline code** - can be inserted directly into the text of an R Markdown file and can refer to it directly in the write-up
- White background - type plain text with markdown syntax
- Basic formatting options:
    - To start a new paragraph, end a line with two spaces
    - To apply **italics,** place an **asterisk** at the beginning and at the end of the word or phrase, eg. *italics works*
    - To apply **bold**, place **two asterisks** at the beginning and at the end of the word or phrase, eg. **bold is useful**
    - To create a **header**, type a **hashtag** (#) followed by a space and your text for eg. # Getting Started with R Markdown
        - Headers will appear in blue, a single hashtag is the largest header, the more hashtags you add (up to 6) the smaller
    - Angled brackets <> for links
    - Single * for bullet points
    - **Embed link** - type click here [link], change angle brackets to parentheses
    - **Embed image** - type ![caption for the image], next paste URL inside parentheses

R packages with template:

- The vitae package contains templates for creating and maintaining a résumé or curriculum vitae (CV)
- The rticles package provides templates for various journals and publishers
- The learnr package makes it easy to turn any R Markdown document into an interactive tutorial
- The bookdown package facilitates writing books and long-form articles
- The flexdashboard package lets you publish a group of related data visualizations as a dashboard
-