Introduction to R

By: Terry Hu

This guide is here to help with R syntax and how to use RStudio, now known as Posit. By copying the code in the code blocks, you will be able to follow along this tutorial.

History

R is a programming language which focuses on statistical computing and graphics that was developed as a GNU Project. The GNU Project is part of the Free Software Movement that sought to make programs free to use for all. R is available under the terms of the Free Software Movement's GNU Public License, and is widely supported across systems including Windows, MacOS and Linux. As a statistical programming language, R lacks the versatility of general purpose languages such as Python or C++. However, R excels at statistical applications of large datasets, including matrix-array operations, data analysis and data visualization.

R's core development team has been committed to documentation, and with continual dedication from the R community, core R libraries and functions are cataloged. Many packages and libraries exists for the language, and users can also write their own functions, increasing its flexibility. For advance programmers, R allows code written in C/C++ to be called and for the objects in R to be manipulated by the C/C++ code.

One of the most common environments to run R is RStudio, which is now know as Posit; RStudio will be the IDE for which this wiki uses (but feel free to use other environments as well).

You can read more about the GNU Project here (https://www.gnu.org/) or about the R Project here (https://www.r-project.org/about.html).

Some key ideas behind R Syntax

Below is a short list of R syntax that users of other languages may need to take notice of when using R. * R is **case sensitive** * *Hastags* (#) are used for comments * <- is used for assignments * R can also be space sensitive

R data structures

R, like many other programming languages, has data structures. The main types of data structures in R are: vectors, matrices, arrays, lists and data frames. The first three are similar, but differ only in their dimensionality. Vectors are usually 1-D, matrices 2-D, and array n-D. Unlike other programming languages, the data type need not be declared in front of the variable assignment (though you can); rather R interprets the data structure, often through the parameters passed in.

Below are some of the basic data structures in R and how to use and declare them.

Vectors

- **Vectors** in R is a group of data elements of the same type stored together
- Vectors have two properties: *type* and *length*
- There are **6** primary types of atomic vectors:

1. Logical (Boolean)

2. Integer
3. Double
4. Character (String)
5. Complex
6. Raw

- To create vectors that contain integers, you must add **L** after the number

```
x<-c(2L,3L,5L)
print(x)
```

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

The above codes declares a vector `x` that contains three integers. * To create vectors that contain strings, you must add ” “ around the string

```
x<-c("Amber","Anin","Kit")
print(x)
```

```
## [1] "Amber" "Anin"  "Kit"
```

The above codes declares a vector `x` that contains three strings. * To find vector length, there is function `length(vector)` * To find vector type, there is function `typeof(vector)`

- You can rename the elements of a vector thru function `names(vector)`

```
x<-c(1,2,3)
names(x)<-c("a","b","c")
print(x)
```

```
## a b c
## 1 2 3
```

The above code renames the elements in vector `x` .

Lists

- Lists can contain elements of **different data types**
- To create a list, you can use function `list()`

```
cars<-list("Taycan", 911L, "A45S", TRUE)
```

In this example, the list consists of two strings (“Taycan” and “A45S”) a integer (911) and a boolean (TRUE). * You can store a list inside a list * Just like everything else, you should use the assignment operator to create a list * To see the structure of a list, use `str()` function

```
str (cars)
```

```
## List of 4
##  $ : chr "Taycan"
##  $ : int 911
##  $ : chr "A45S"
##  $ : logi TRUE
```

The above code will return the elements types in list `cars` . * To rename a list, you can use =

```
porsche <-list('Taycan' =1,'911'=2)
print(porsche)
```

```
## $Taycan
## [1] 1
##
## $`911`
## [1] 2
```

The above code changes the elements in list `porsche` . ## Dates and Time in R Often when working with data, time and dates are important. By default, R uses YYYY-MM-DD. Below are some functions that have to do with date and time: 1. `today()` returns today's date 2. `now()` returns the current time 3. `ymd("string)` , `mdy("string")` , `dmy("string")` to convert to date 4. `as_date()` turns date-time to date 5. `ymd_hms` gives hours, mins and secs

```
install.packages("lubridate")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
library("lubridate")
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
print(today("GMT"))
```

```
## [1] "2023-03-24"
```

```
print(now("GMT"))
```

```
## [1] "2023-03-24 22:11:57 GMT"
```

In the code above, we can see the use of the package “lubridate,” which contains many of the listed functions used to manipulate time and date. You can install packages with the function or command `install.packages`

Boolean and Logical Operators

When attempting to work with data, boolean and logical operators can help the user compare data. Below is a list of commonly used boolean operators in R. Some points to take note of are that both `&` and `&&` can stand for the AND operation and that though `<-` is the assignment operator, the EQUALS operation still uses `==` .

- AND (& or &&)
- OR (| or ||)
- NOT (!)
- EQUALS (==)
- `if()`
- `else()`
- `elseif()`

Other Basic Functions and Structures

- Data frames are 2-D arrays like spreadsheets, they can be created with the function `_data.frame()_`

```
a <-data.frame(x=c(1,2,3), y=c(2,3,4))
print(a)
```

```
##   x y
## 1 1 2
## 2 2 3
## 3 3 4
```

In this example above, a data frame `a` is created that consists of two vectors `x` and `y` .

The following functions below have to deal with file and object manipulation. * `dir.create` creates new directory * `file.create("file_name")` creates new file * `file.copy("file_name", "destination")` * `matrix(vector,nrow=" " or ncol=" ")` creates matrix

```
matrix <-matrix(c(3:8), nrow=2)
print(matrix)
```

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

The code above prints elements in matrix `matrix` .

- `facet_wrap()` creates different plots for different elements
- **`install.packages("package")`** installs packages to diversify functionality
- **`library("package")`** loads package
- usually you will need to install and load packages before being able to use them

- tidyverse is one of the most useful R packages, you can use `tidyverse_update()` to update, tidyverse has the following core packages

1. ggplot2 - for viz
2. tidyr - data cleaning
3. readr - importing data
4. dplyr - data manipulation
5. tibble
6. purrr
7. stringr
8. forcats

- For additional help, you can visit CRAN (https://rmarkdown.rstudio.com/authoring_basics.html)

Pipe

- A pipe in R takes the output of one statement into input of another statement, they can be used thr the `% > %` operator.
- The standard form of a pipe is `FUNCTION 1 % >% FUNCTION 2`
- If the Pipe is successfully implemented, the pipe should be auto-indented
- The pipe operation should be added after each operation except the last one
- A pipe can also be called using **CTRL** + **SHIFT** + **M**

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

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
library("tidyverse")
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr   1.1.1     ✓ readr   2.1.4
## ✓ forcats 1.0.0     ✓ stringr 1.5.0
## ✓ ggplot2 3.4.1     ✓ tibble  3.2.1
## ✓ purrr   1.0.1     ✓ tidyr   1.3.0
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts to be
come errors
```

```
filtered_toothgrowth <-ToothGrowth %>%
  filter(dose==0.5) %>%
  arrange(len)
print(filtered_toothgrowth)
```

```
##     len supp dose
## 1   4.2   VC  0.5
## 2   5.2   VC  0.5
## 3   5.8   VC  0.5
## 4   6.4   VC  0.5
## 5   7.0   VC  0.5
## 6   7.3   VC  0.5
## 7   8.2   OJ  0.5
## 8   9.4   OJ  0.5
## 9   9.7   OJ  0.5
## 10  9.7   OJ  0.5
## 11 10.0   VC  0.5
## 12 10.0   OJ  0.5
## 13 11.2   VC  0.5
## 14 11.2   VC  0.5
## 15 11.5   VC  0.5
## 16 14.5   OJ  0.5
## 17 15.2   OJ  0.5
## 18 16.5   OJ  0.5
## 19 17.6   OJ  0.5
## 20 21.5   OJ  0.5
```

Dataframes and R

- Dataframes are collections of columns
    - columns should be named
    - data stored can be different type
    - each colu,m should contain same number of data items
- tibles ~ streamlined dataframes
    - never change data type of outputs
    - never change variable names
    - never create row names
    - makes printing easier

Functions to work with dataframes and tibbles

1. `head(dataset)` views first rows
2. `str(dataset)` views structure
3. `colnames(dataset)` views column names
4. `mutate(dataset,new_col_name = definition)` add new column
5. `readr` functions, **for all readr functions, remember to put “” around file name -> “file_name.format”**

- **`read.csv()`**
- `read.tsv()`
- `read.table()`
- `read.delim()`
- `read.log()`
- `read.fwf()`

6. To generate summaries of data:

- `skim_without_charts(dataset)` for a comprehensive view
- `glimpse(dataset)`
- `head(dataset)`
- `select(dataset)` select specifics, ie. a single column

```
install.packages("palmerpenguins")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
library("palmerpenguins")
penguins %>%
  select(species)
```

```
## # A tibble: 344 × 1
##    species
##    <fct>
##  1 Adelie
##  2 Adelie
##  3 Adelie
##  4 Adelie
##  5 Adelie
##  6 Adelie
##  7 Adelie
##  8 Adelie
##  9 Adelie
## 10 Adelie
## # ℹ 334 more rows
```

7. `clean_names(dataset)` - just nice to use
8. `rename(column_name_new = column_name_old)` for this to work, you need to first select all data
9. `rename_with(column_name, WHAT TO DO)` ie. TOUPPER or TOLOWER
10. `drop_na()` leaves out missing elements
11. **`summarize()`** prints summary of data, can be customized to see what kind of summary

```
penguins %>%
  group_by(island)%>%
  drop_na() %>%
  summarize(max_bill_length = max(bill_length_mm))
```

```
## # A tibble: 3 × 2
##   island    max_bill_length
##   <fct>               <dbl>
## 1 Biscoe               59.6
## 2 Dream                58
## 3 Torgersen            46
```

12. `dataframe_name <- data.frame(member 1; member 2...)` to create custom data frame, similar to class in C

13. `separate()` separates column into new columns
14. `unite()` combines columns together
15. `pivot_longer()` increases row, decreases column
16. `pivot_shorter()` increases column, decreases row
17. `arange(dataset)` shows data in ascending order

GGPLOT2

GGPLOT2 is one of the most widely used package to create visuals in R. There is extensive support which will allow the user to manipulate different parts of the visualization process to create graphs that help them tell the story they want. In this section, we will see some basic functions of ggplot2 and how to come up with stunning visualizations of data.

Remember to install the package first with `install.packages("ggplot2")`

Benefits of ggplot2	Elements of ggplot2
Can create different types of plots	**Aesthetics**
Customize looks and feel of plot	**Geom -** geometric objects used to represent data
Create high quality visuals	**Facets -** allows you to display subsets of data
Combine data visualization and manipulation	**Labels and Annotations**

Making Viz (visuals) with ggplot2 Usually we will also need to load the packages with the following code:

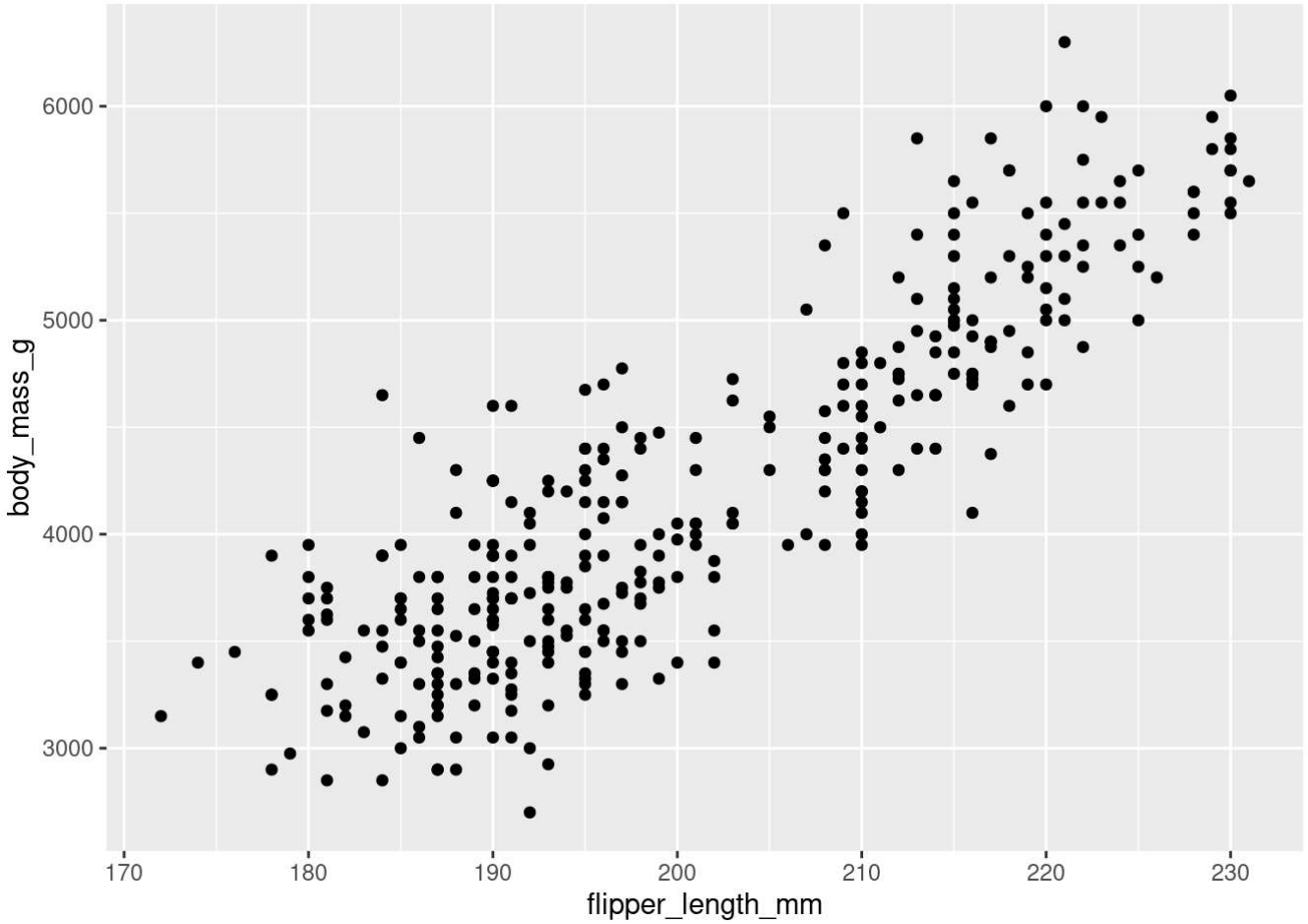
```
install.packages("ggplot2")
library(ggplot2)
install.packages("palmerpenguins")
library(palmerpenguins)
```

but since they are already loaded from before, we will not execute the code to save computing resources

A basic plot from ggplot2 will look as follows:

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

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



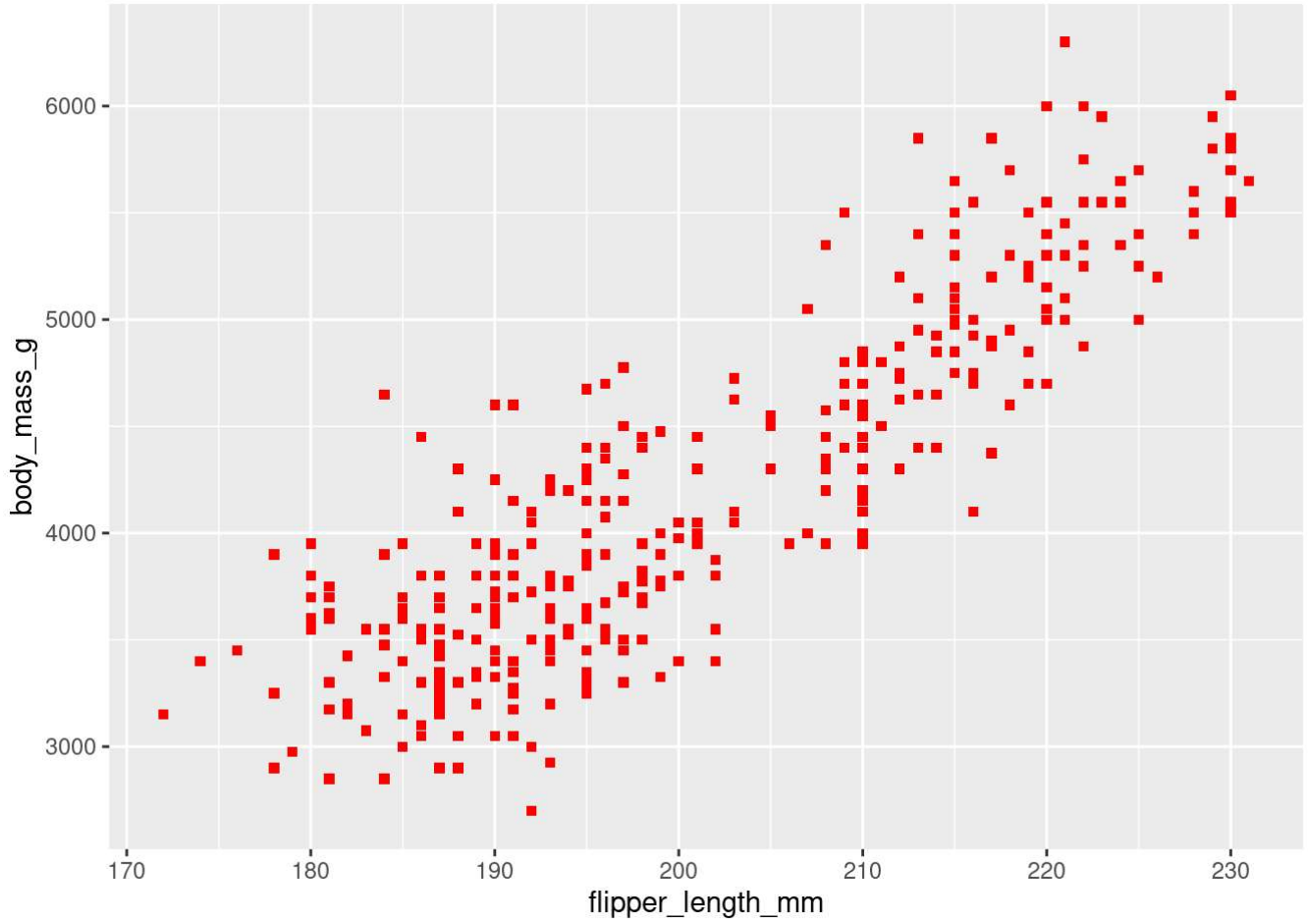
1. To begin the plot, you can use the function `ggplot(data)`
2. Next, use `+` to add layer. The + always has to go at the end of line, similar to pipe
3. Use `geom_function` to display data

4. Map variables using the `aes()` function

- To make aesthetics that are not mapped, make sure to put outside the `aes()` function

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

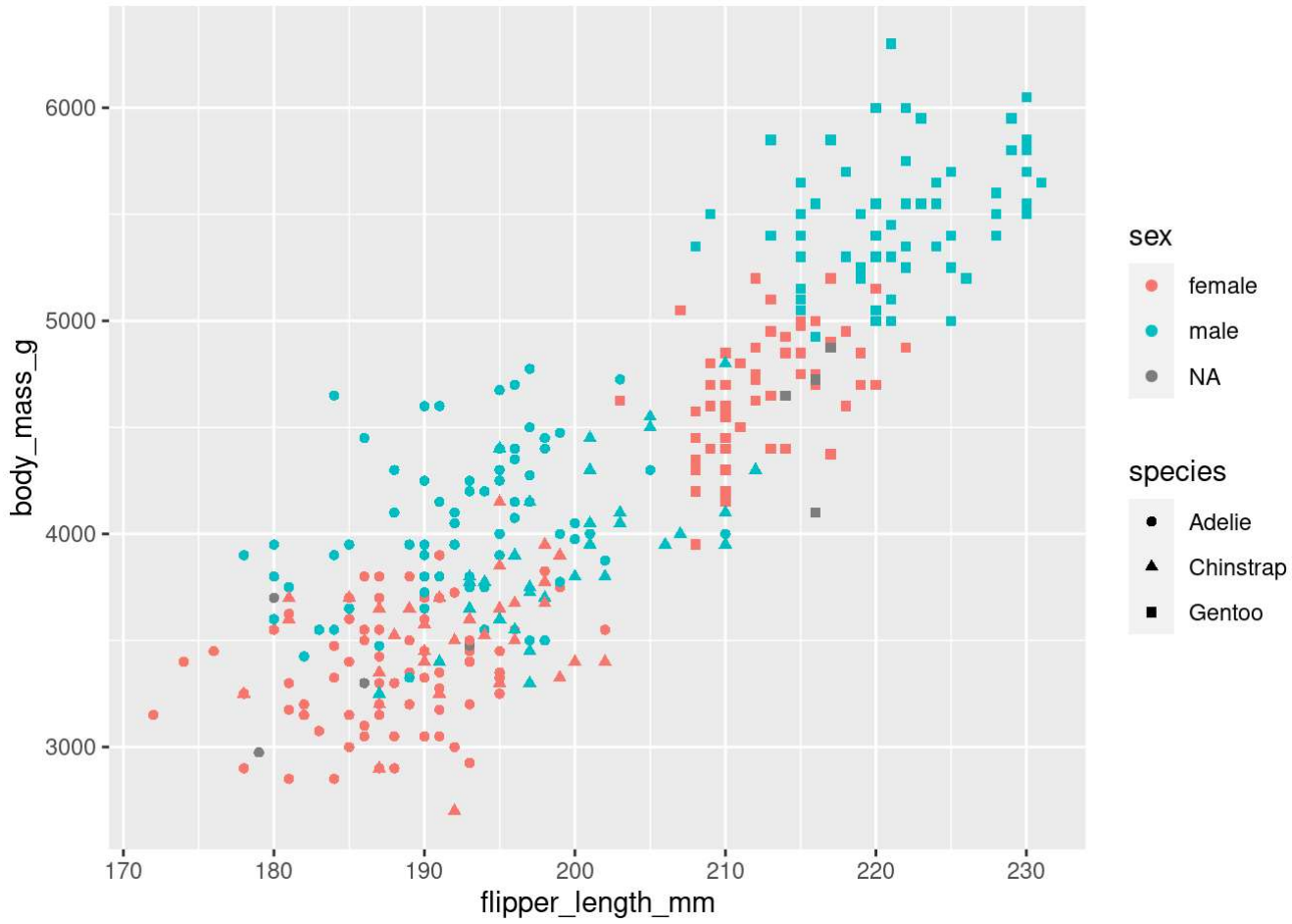
```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



- You can also map multiple things at once

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

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



Types of geom_functions

1. `geom_point()` creates scatter plot
2. `geom_bar()`
3. `geom_line()`
4. `geom_smooth()` for trendline

Types of aesthetics

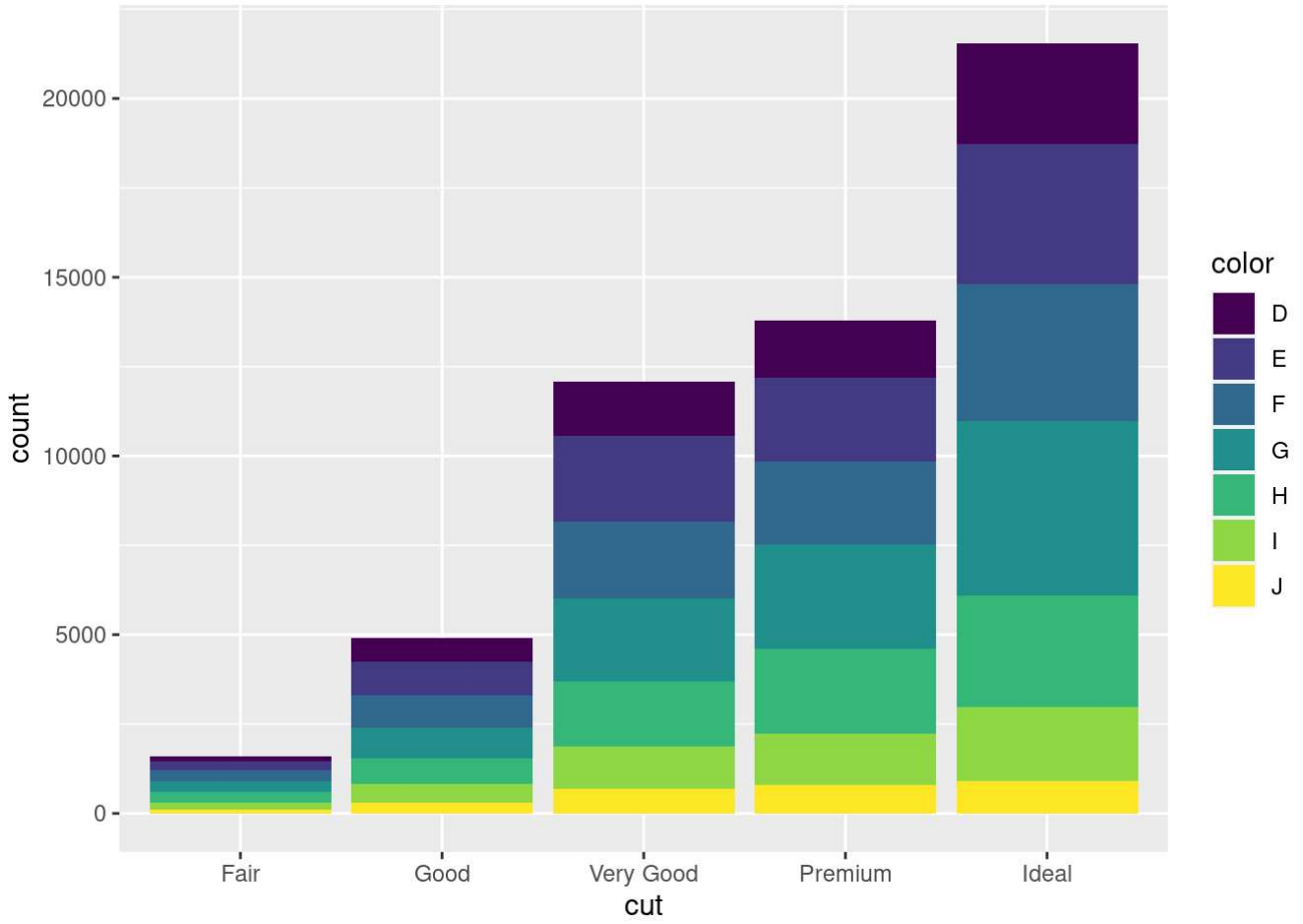
1. X
2. Y
3. Size
4. Shape
5. Alpha - transparency
6. Color

You can get more information from the ggplot2 cheat sheet (https://www.maths.usyd.edu.au/u/UG/SM/STAT3022/r/current/Misc/data-visualization-2.1.pdf)

To make a stacked bar chart

To make a stacked bar chart, you make a normal bar chart but map x to one variable while the fill to another variable For example:

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



Facet Functions

Facet function allows you to view same subset of a dataset, it will usually make many smaller plots side by side 1. `facet_wrap()` - makes 1-D into 2-D, usually for long data 2. `facet_grid()` - turns into matrix for easier viewing

Label Functions

Labels and annotations allow us to add words to the plot. Labels are outside while annotations are inside.

- To make Labels, we can use `labs(title = " ", subtitle = " ", caption = " ")`
- **When making labels, remember to add `+` to add layer**

Annotate Functions

- Use `annotate`
- adds text inside the plot
- **Annotate Arguments**
  - `x=` , `y=` - position of text
  - `label =` - actual text
  - `color =`
  - `fontsize =`

- `size =`
- `angle =`

To Save Plot

1. You can use the export option
2. Use `ggsave()`

- `ggsave("file_name.format")`

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com (http://rmarkdown.rstudio.com).

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

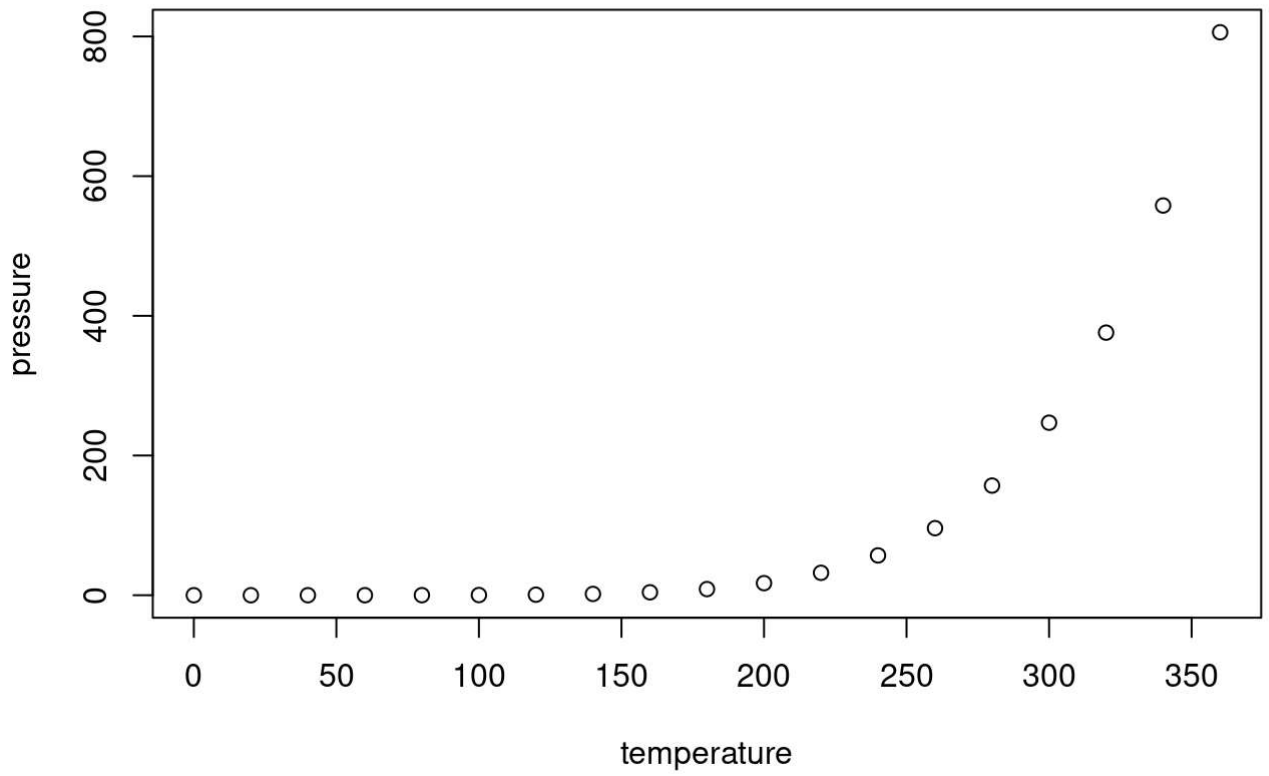
```
##      Length Class  Mode
## [1,] 1      -none- character
## [2,] 1      -none- numeric
## [3,] 1      -none- character
## [4,] 1      -none- logical
```

You can use shortcut **CTRL** + **ALT** + **I**

For more on R Markdown, you can click here (https://rmarkdown.rstudio.com/authoring_basics.html)

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Conclusion

This guide outlines the basic syntax of R, and applications when this language is useful. By following this guide, the user should be able to understand data structures in R, manipulate data and export the results visually. The guide also covers R Markdown, which can be used to create aesthetically pleasing reports transcribing the code and the results. Further reading is available through some of the links attached, and a plethora of other learning resources can be found at the R website and the links attached.

Sources and Links

Below is a list of resources that this wiki has allude to or has based information from: * https://www.gnu.org/ (https://www.gnu.org/) * https://www.r-project.org/about.html (https://www.r-project.org/about.html) * https://rmarkdown.rstudio.com/authoring_basics.html (https://rmarkdown.rstudio.com/authoring_basics.html) * https://www.maths.usyd.edu.au/u/UG/SM/STAT3022/r/current/Misc/data-visualization-2.1.pdf (https://www.maths.usyd.edu.au/u/UG/SM/STAT3022/r/current/Misc/data-visualization-2.1.pdf)