Seminar 1

Shiva Chelluri

2025-01-20

Introduction:

You can conduct basic math as follows:

```
# you can conduct basic math
x <- 3 + sin(pi/2)

# you can also conduct more complicated mathematical
sqrt(x)
```

```
## [1] 2
```

```
# you can also save strings dynamically
z <- "word"
```

Sequences:

Sequences are functional in programming such as looping. They can be made using the `seq()` function.

```
v <- seq(1, 10)
# another easier way of making sequences:
v1 <- 1:10
```

Vectors:

You can create vectors of data, which can include numbers, characters and other kinds of types as well:

```
w <- c(12, 1, 2, 3)

# you can also sort vectors using the sort() function
sort(w)
```

```
## [1]  1  2  3 12
```

As you can see, `w` itself does not change because the `sort()` function does apply in place. So, to store this newer version, you need to save it again:

```r
w <- sort(w)
```

You can also do **indexing over a vector:**

```r
w[1]
```

```
## [1] 1
```

Note that the indexing in R starts with one and not zero, like in other languages. Additionally, you can conduct scalar multiplications on vectors:

```r
w*2
```

```
## [1]  2  4  6 24
```

You conduct more complex matrix methods such as inner product and matrix multiplications.

```r
# for doing inner-products
w %*% w
```

```
##      [,1]
## [1,]  158
```

Data Formatting:

Usually, you will get data with the wrong data types, and you need to format the data before such as:

```r
A <- as.character(c(1, 2, 3, 4, 5))
typeof(A)
```

```
## [1] "character"
```

Here, you have vector numbers, but they are saved as characters. So, we use two auxiliary functions to identify two main attributes of a variable:

1. Length of the vector: `length()` returns the length of the vector.
2. Data Type of the vector elements: `typeof()` returns the variable's data type.

```r
length_a <- length(A)
typeof_a <- typeof(A)
print(paste0("The vector A is of length", length_a, "and has the data type of ", typeof(A)))
```

```
## [1] "The vector A is of length5and has the data type of character"
```

Coercion: It is the process of forcing data of one datatype to another datatype

Loops:

You can also conduct for-loops using the `for(){}` function to write loops. They have the common syntax as:

```r
for (i in 1:10){
  print(paste0("Printing loop:", i))
}
```

```
## [1] "Printing loop:1"
## [1] "Printing loop:2"
## [1] "Printing loop:3"
## [1] "Printing loop:4"
## [1] "Printing loop:5"
## [1] "Printing loop:6"
## [1] "Printing loop:7"
## [1] "Printing loop:8"
## [1] "Printing loop:9"
## [1] "Printing loop:10"
```

Packages:

To get external functions, we need to use the `install.packages("")` function to install the packages.

We will primarily use the `tidyverse()` package because it contains most of the other packages for the stuff we are going to do.

```r
# loading the tidyverse package
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Plotting with `ggplot2()`

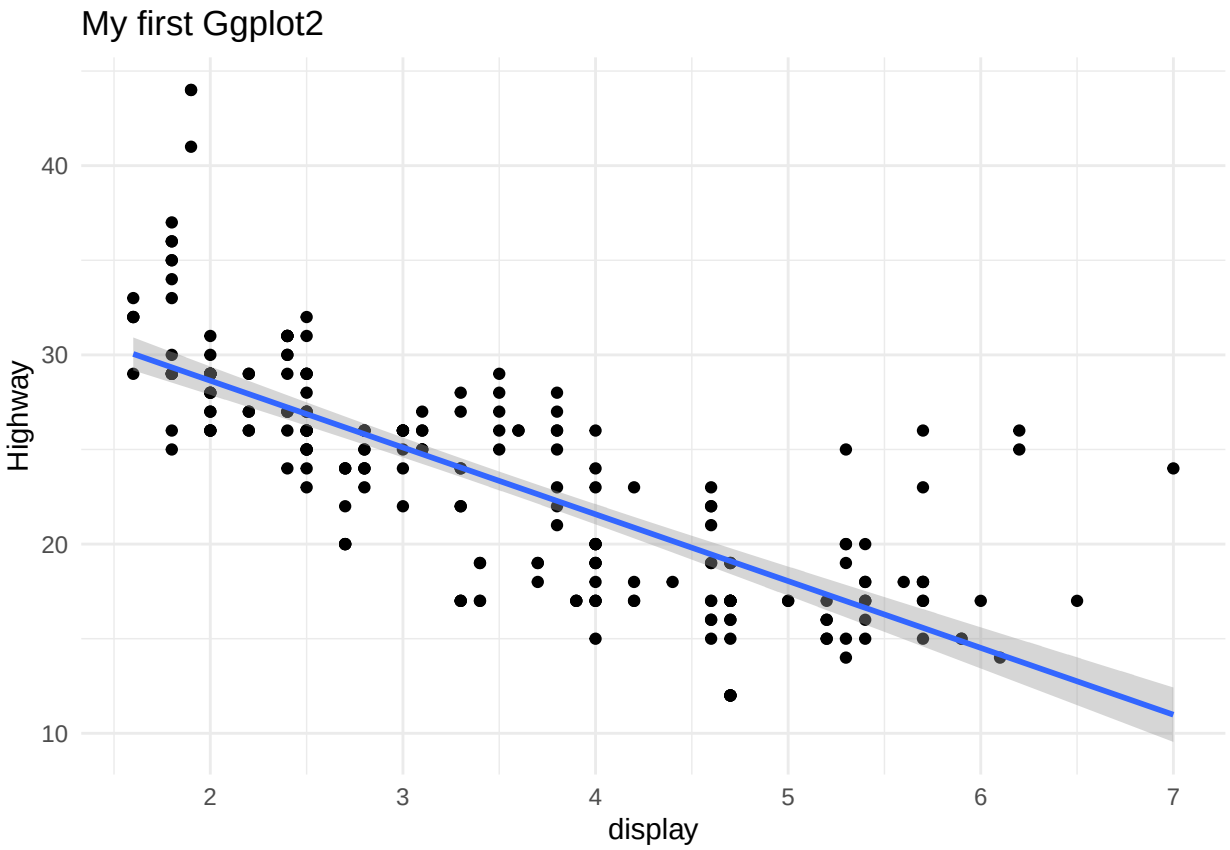
Let us load the `mpg` data set from the `tidyverse` package. And let us conduct basic plotting:

```r
cars <- mpg


# example ggplot with linear regressions
ggplot(data = cars) +
  geom_point(mapping = aes(x=displ, y=hwy, xlab= "Display", ylab="Highway"))+
  geom_smooth(method = "lm", mapping = aes(x=displ, y=hwy))+
  labs(y="Highway", x = "display", title="My first Ggplot2")+
  theme_minimal()
```

```
## Warning in geom_point(mapping = aes(x = displ, y = hwy, xlab = "Display", :
## Ignoring unknown aesthetics: xlab and ylab
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Introduction to Markdown:

You can write bulleted lists as:

- Item 1
- item 2
- item 2

You can also write numbered lists as:

1. Item 1
2. Item 2
3. Item 3

Code Blocks:

You can write code-blocks using the ```` ```{r} ```` and it looks like the following:

```r
print("This is a code block")
```

```
## [1] "This is a code block"
```

Math:

You can write two types of math:

1. Inline math: You do it using "\$\$" symbols. It works such as $y = mx + c$
2. MathJAX Block: You can do it using double dollar signs:

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon$$