R Workshop

Importing Data into R - Part 1

Introduction

In this chapter, we will focus on

- reading data from flat or delimited files
- specifying column data types
- reading specific columns only

Libraries, Data & Code

We will use the readr package. The data sets can be downloaded from here and the codes from here.

```
library(readr)
```

Overview

Type	readr	Base R
comma	read_csv()	read.csv()
semicolon	read_csv2()	read.csv2()
tab	read_tsv()	read.delim() / read.table()
space	read_table()	read.table()
multiple spaces	read_table2()	read.table()
any delimiter	read_delim()	read.delim()

The above table gives an overview of the functions for reading different types of files in readr and Base R. All the functions in readr offer a common set of options which are described below. We will learn about them in greater detail in the next section.

- `col_names`: whether data includes column names
- `n_max`: maximum number of lines/rows to read
- `col_types`: data type of the columns
- `skip`: number of lines/rows to skip

Case Study

In this section, we will read data from a csv (comma separated values) file and explore the options listed in the previous section.

```
read_csv('hsb2.csv')
```

```
## Parsed with column specification:
## cols(
```

```
##   id = col_integer(),
##   female = col_integer(),
##   race = col_integer(),
##   ses = col_integer(),
##   schtyp = col_integer(),
##   prog = col_integer(),
##   read = col_integer(),
##   write = col_integer(),
##   math = col_integer(),
##   science = col_integer(),
##   socst = col_integer()
## )
```

```
## # A tibble: 200 x 11
##       id female  race   ses schtyp  prog  read write  math science socst
##    <int>  <int> <int> <int>  <int> <int> <int> <int> <int>   <int> <int>
##  1    70      0     4     1      1     1    57    52    41      47    57
##  2   121      1     4     2      1     3    68    59    53      63    61
##  3    86      0     4     3      1     1    44    33    54      58    31
##  4   141      0     4     3      1     3    63    44    47      53    56
##  5   172      0     4     2      1     2    47    52    57      53    61
##  6   113      0     4     2      1     2    44    52    51      63    61
##  7    50      0     3     2      1     1    50    59    42      53    61
##  8    11      0     1     2      1     2    34    46    45      39    36
##  9    84      0     4     2      1     1    63    57    54      58    51
## 10    48      0     3     2      1     2    57    55    52      50    51
## # ... with 190 more rows
```

Great! If you see the above output, you have successfully read data into R. In case you get an error, do not worry and do the following:

- check the separator in the file and ensure it is a `comma`
- check the path to the file

When you read data using readr, it will display the data type detected for each column/variable in the data set. If you want to check the data types before reading the data, use `spec_csv()`. We will learn to specify the column types in the next section.

```
spec_csv('hsb2.csv')
```

```
## cols(
##   id = col_integer(),
##   female = col_integer(),
##   race = col_integer(),
##   ses = col_integer(),
##   schtyp = col_integer(),
##   prog = col_integer(),
##   read = col_integer(),
##   write = col_integer(),
##   math = col_integer(),
##   science = col_integer(),
##   socst = col_integer()
## )
```

Options

Column Names

Use `col_names` to indicate whether the data includes column names. It takes two values, `TRUE` and `FALSE`. If set to `FALSE`, readr will generate column names. In the below example, we will read data from a file which does not have column names present in the first row.

```
read_csv('hsb3.csv', col_names = FALSE)
```

```
## # A tibble: 200 x 11
##       X1    X2    X3    X4    X5    X6    X7    X8    X9   X10   X11
##    <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
##  1    70     0     4     1     1     1    57    52    41    47    57
##  2   121     1     4     2     1     3    68    59    53    63    61
##  3    86     0     4     3     1     1    44    33    54    58    31
##  4   141     0     4     3     1     3    63    44    47    53    56
##  5   172     0     4     2     1     2    47    52    57    53    61
##  6   113     0     4     2     1     2    44    52    51    63    61
##  7    50     0     3     2     1     1    50    59    42    53    61
##  8    11     0     1     2     1     2    34    46    45    39    36
##  9    84     0     4     2     1     1    63    57    54    58    51
## 10    48     0     3     2     1     2    57    55    52    50    51
## # ... with 190 more rows
```

`col_names` can be used to specify column names while reading data. We need to store the names as a character vector and supply it to `col_names`. Let us reread `hsb3` and specify column names.

```
cnames <- c("id", "female", "race", "ses", "schtyp", "prog", "read", "write", "math", "science", "socst
read_csv('hsb3.csv', col_names = cnames)
```

```
## # A tibble: 200 x 11
##       id female  race   ses schtyp  prog  read write  math science socst
##    <int>  <int> <int> <int>  <int> <int> <int> <int> <int>   <int> <int>
##  1    70      0     4     1      1     1    57    52    41      47    57
##  2   121      1     4     2      1     3    68    59    53      63    61
##  3    86      0     4     3      1     1    44    33    54      58    31
##  4   141      0     4     3      1     3    63    44    47      53    56
##  5   172      0     4     2      1     2    47    52    57      53    61
##  6   113      0     4     2      1     2    44    52    51      63    61
##  7    50      0     3     2      1     1    50    59    42      53    61
##  8    11      0     1     2      1     2    34    46    45      39    36
##  9    84      0     4     2      1     1    63    57    54      58    51
## 10    48      0     3     2      1     2    57    55    52      50    51
## # ... with 190 more rows
```

Skip Lines

Use `skip` to skip a certain number of lines. For example, if the file has contents other than data in the first few lines, we need to skip them before reading the data. In the below example, we will skip the first 3 lines as they contain information about the data set which we do not need.

```
read_csv('hsb4.csv', skip = 3)
```

```
## # A tibble: 200 x 11
##       id female  race   ses schtyp  prog  read write  math science socst
```

```
##    <int>  <int> <int> <int>  <int> <int> <int> <int> <int>   <int> <int>
##  1    70      0     4     1      1     1    57    52    41      47    57
##  2   121      1     4     2      1     3    68    59    53      63    61
##  3    86      0     4     3      1     1    44    33    54      58    31
##  4   141      0     4     3      1     3    63    44    47      53    56
##  5   172      0     4     2      1     2    47    52    57      53    61
##  6   113      0     4     2      1     2    44    52    51      63    61
##  7    50      0     3     2      1     1    50    59    42      53    61
##  8    11      0     1     2      1     2    34    46    45      39    36
##  9    84      0     4     2      1     1    63    57    54      58    51
## 10    48      0     3     2      1     2    57    55    52      50    51
## # ... with 190 more rows
```

Maximum Lines

Use `n_max` to specify the maximum number of lines to read. Suppose we want to read only 100 rows of data from a file, we can set `n_max` equal to 100. In the next example, we will read the first 120 rows from the `hsb2` file. If you observe the last row in the output, it says `# ... with 110 more rows`, indicating that only 120 rows of data has been read from the file.

```
read_csv('hsb2.csv', n_max = 120)
```

```
## # A tibble: 120 x 11
##       id female  race   ses schtyp  prog  read write  math science socst
##    <int>  <int> <int> <int>  <int> <int> <int> <int> <int>   <int> <int>
##  1    70      0     4     1      1     1    57    52    41      47    57
##  2   121      1     4     2      1     3    68    59    53      63    61
##  3    86      0     4     3      1     1    44    33    54      58    31
##  4   141      0     4     3      1     3    63    44    47      53    56
##  5   172      0     4     2      1     2    47    52    57      53    61
##  6   113      0     4     2      1     2    44    52    51      63    61
##  7    50      0     3     2      1     1    50    59    42      53    61
##  8    11      0     1     2      1     2    34    46    45      39    36
##  9    84      0     4     2      1     1    63    57    54      58    51
## 10    48      0     3     2      1     2    57    55    52      50    51
## # ... with 110 more rows
```

Column Types

In certain cases, we need to specify the data type of the columns. It might be related to dates or categorical variables. readr allows us to specify the data types using `col_xxx` functions which include:

- `col_double()`
- `col_integer()`
- `col_factor()`
- `col_character()`
- `col_datetime()`

To specify the data types, we will use `col_types` argument and supply to it a list indicating the data type (using `col_xxx`) of each column in the data set. In the below example, we read data from `hsb2` file while sprcifying the data types. Keep in mind that we need to specify the data type for each column.

```
read_csv('hsb2.csv', col_types = list(
  col_integer(), col_factor(levels = c(0, 1)),
  col_factor(levels = c(1, 2, 3, 4)), col_factor(levels = c(1, 2, 3)),
```

```
  col_factor(levels = c(1, 2)), col_factor(levels = c(1, 2, 3)),
  col_integer(), col_integer(), col_integer(), col_integer(),
  col_integer())
)
```

```
## # A tibble: 200 x 11
##       id female race  ses   schtyp prog   read write  math science socst
##    <int> <fct>  <fct> <fct> <fct>  <fct> <int> <int> <int>   <int> <int>
##  1    70 0      4     1     1      1        57    52    41      47    57
##  2   121 1      4     2     1      3        68    59    53      63    61
##  3    86 0      4     3     1      1        44    33    54      58    31
##  4   141 0      4     3     1      3        63    44    47      53    56
##  5   172 0      4     2     1      2        47    52    57      53    61
##  6   113 0      4     2     1      2        44    52    51      63    61
##  7    50 0      3     2     1      1        50    59    42      53    61
##  8    11 0      1     2     1      2        34    46    45      39    36
##  9    84 0      4     2     1      1        63    57    54      58    51
## 10    48 0      3     2     1      2        57    55    52      50    51
## # ... with 190 more rows
```

Specific Columns

We may not always want to read all the columns from a file. In such cases, we can specify the columns to be read using `col_types` argument and supplying to it the names of the columns to be read. We will use `cols_only()` to specify the column names and their respective data types.

```
read_csv('hsb2.csv', col_types = cols_only(id = col_integer(),
  prog = col_factor(levels = c(1, 2, 3)), read = col_integer())
)
```

```
## # A tibble: 200 x 3
##       id prog   read
##    <int> <fct> <int>
##  1    70 1        57
##  2   121 3        68
##  3    86 1        44
##  4   141 3        63
##  5   172 2        47
##  6   113 2        44
##  7    50 1        50
##  8    11 2        34
##  9    84 1        63
## 10    48 2        57
## # ... with 190 more rows
```

Practice

- check the separator type in the following files and read them using appropriate `read_xxx()` function:
  - `hsb.csv`
  - `mtcars.tsv`
  - `hsb1.csv`
  - `hsb.txt`

Summary

In this chapter, we explored:

- reading data from flat/delimited files
- reading specific columns
- specifying
  - column data types
  - number of skipping lines
  - maximum number of lines to read
  - if data includes column names

Importing Data into R - Part 2

Introduction

This is the second chapter in the series **Importing Data into R**. In the previous chapter, we explored reading data from flat/delimited files. In this chapter, we will:

- list sheets in an excel file
- read data from an excel sheet
- read specific cells from an excel sheet
- read specific rows
- read specific columns
- read data from - SAS - SPSS - STATA

Libraries, Data & Code

We will use the readxl package. It has no external dependencies as compared to other packages available for reading data from Excel. The data sets can be downloaded from here and the codes from here.

```
library(readxl)
```

List Sheets

Before we read data from an excel file, let us see how many sheets are present using `excel_sheets()`.

```
excel_sheets('sample.xls')
```

```
## [1] "ecom"
```

Read Sheet

To read data from a particular sheet, use `read_excel()` and specify the file name and the sheet number. Below is a simple example:

```
read_excel('sample.xls', sheet = 1)
```

```
## # A tibble: 7 x 5
##   channel        users new_users sessions bounce_rate
##   <chr>          <dbl>     <dbl>    <dbl> <chr>
## 1 Organic Search 43296     40238    50810 48.72%
## 2 Direct         12916     12311    16419 49.27%
## 3 Referral       10983      7636    18105 22.26%
## 4 Social         10346     10029    11101 61.92%
## 5 Display         5564      4790     7220 83.30%
## 6 Paid Search     2687      2205     3438 38.02%
## 7 Affiliates      1773      1585     2167 55.75%
```

Read Specific Cells

To read data from specific cells or a range of cells, use the `range` arguments and specify the range of cells from which we want to read data. For example, to read data from first 4 rows of columns **B** and **C**, we will specify the range as `"B1:C4"`.

```
read_excel('sample.xls', sheet = 1, range = "B1:C4")
```

```
## # A tibble: 3 x 2
##   users new_users
##   <dbl>     <dbl>
## 1 43296     40238
## 2 12916     12311
## 3 10983      7636
```

To read data from first 5 rows of columns **A**, **B** and **C**, we will specify the range as `"A1:C5"`.

```
readxl::read_excel('sample.xls', sheet = 1, range = "A1:C5")
```

```
## # A tibble: 4 x 3
##   channel        users new_users
##   <chr>          <dbl>     <dbl>
## 1 Organic Search 43296     40238
## 2 Direct         12916     12311
## 3 Referral       10983      7636
## 4 Social         10346     10029
```

Another way to read specific cells is by providing a particular cell and then specifying the number of rows and columns keeping that cell as anchorage. In the below example, we want to read 3 rows and 2 columns starting from the cell `A4`.

```
readxl::read_excel('sample.xls', sheet = 1, col_names = FALSE,
  range = anchored("A4", dim = c(3, 2)))
```

```
## # A tibble: 3 x 2
##   X__1      X__2
##   <chr>    <dbl>
## 1 Referral 10983
## 2 Social   10346
## 3 Display   5564
```

Use cell_limits to specify one end of the rectangle such as top left or top right.

```
readxl::read_excel('sample.xls', sheet = 1,
  range = cell_limits(c(1, 2), c(NA, NA)))
```

```
## # A tibble: 7 x 4
##   users new_users sessions bounce_rate
##   <dbl>     <dbl>    <dbl> <chr>
## 1 43296     40238    50810 48.72%
## 2 12916     12311    16419 49.27%
## 3 10983      7636    18105 22.26%
## 4 10346     10029    11101 61.92%
## 5  5564      4790     7220 83.30%
## 6  2687      2205     3438 38.02%
## 7  1773      1585     2167 55.75%
```

```
readxl::read_excel('sample.xls', sheet = 1,
  range = cell_limits(c(1, NA), c(NA, 2)))
```

```
## # A tibble: 7 x 2
##   channel        users
##   <chr>          <dbl>
## 1 Organic Search 43296
```

```
## 2 Direct         12916
## 3 Referral       10983
## 4 Social         10346
## 5 Display         5564
## 6 Paid Search     2687
## 7 Affiliates      1773
```

Specify Rows

Use `cell_rows()` to specify the rows from which data must be read. In the below example, we want to read the first 4 rows of data from the file.

```
readxl::read_excel('sample.xls', sheet = 1, range = cell_rows(1:4))
```

```
## # A tibble: 3 x 5
##   channel        users new_users sessions bounce_rate
##   <chr>          <dbl>     <dbl>    <dbl> <chr>
## 1 Organic Search 43296     40238    50810 48.72%
## 2 Direct         12916     12311    16419 49.27%
## 3 Referral       10983      7636    18105 22.26%
```

Specify Columns

Use `cell_cols()` to specify the columns from which data must be read. In the below example, we want to read the 2nd and 3rd column from the file.

```
readxl::read_excel('sample.xls', sheet = 1, range = cell_cols(2:3))
```

```
## # A tibble: 7 x 2
##   users new_users
##   <dbl>     <dbl>
## 1 43296     40238
## 2 12916     12311
## 3 10983      7636
## 4 10346     10029
## 5  5564      4790
## 6  2687      2205
## 7  1773      1585
```

Statistical Softwares

We will use the haven package to read data from files of other statistical softwares such as:

- SAS
- SPSS
- STATA

Library

```
library(haven)
```

STATA

```
read_stata('airline.dta')
```

```
## # A tibble: 32 x 6
##     year     y     w     r     l     k
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  1948  1.21 0.243 0.145  1.41 0.612
##  2  1949  1.35 0.260 0.218  1.38 0.559
##  3  1950  1.57 0.278 0.316  1.39 0.573
##  4  1951  1.95 0.297 0.394  1.55 0.564
##  5  1952  2.27 0.310 0.356  1.80 0.574
##  6  1953  2.73 0.322 0.359  1.93 0.711
##  7  1954  3.03 0.335 0.403  1.96 0.776
##  8  1955  3.56 0.350 0.396  2.12 0.827
##  9  1956  3.98 0.361 0.382  2.43 0.800
## 10  1957  4.42 0.379 0.305  2.71 0.921
## # ... with 22 more rows
```

SPSS

```
read_spss('employee.sav')
```

```
## # A tibble: 474 x 9
##       id gender    educ   jobcat  salary salbegin jobtime prevexp minority
##    <dbl> <chr+lbl> <dbl+> <dbl+l> <dbl+> <dbl+lb> <dbl+l> <dbl+l> <dbl+lb>
##  1  1.00 m         15     3       57000  27000    98      144     0
##  2  2.00 m         16     1       40200  18750    98      36      0
##  3  3.00 f         12     1       21450  12000    98      381     0
##  4  4.00 f         8      1       21900  13200    98      190     0
##  5  5.00 m         15     1       45000  21000    98      138     0
##  6  6.00 m         15     1       32100  13500    98      67      0
##  7  7.00 m         15     1       36000  18750    98      114     0
##  8  8.00 f         12     1       21900  9750     98      0       0
##  9  9.00 f         15     1       27900  12750    98      115     0
## 10 10.0  f         12     1       24000  13500    98      244     0
## # ... with 464 more rows
```

SAS

```
read_sas('airline.sas7bdat')
```

```
## # A tibble: 32 x 6
##     YEAR     Y     W     R     L     K
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  1948  1.21 0.243 0.145  1.41 0.612
##  2  1949  1.35 0.260 0.218  1.38 0.559
##  3  1950  1.57 0.278 0.316  1.39 0.573
##  4  1951  1.95 0.297 0.394  1.55 0.564
##  5  1952  2.27 0.310 0.356  1.80 0.574
##  6  1953  2.73 0.322 0.359  1.93 0.711
##  7  1954  3.03 0.335 0.403  1.96 0.776
```

```
##  8  1955  3.56 0.350 0.396  2.12 0.827
##  9  1956  3.98 0.361 0.382  2.43 0.800
## 10  1957  4.42 0.379 0.305  2.71 0.921
## # ... with 22 more rows
```

Data Wrangling with dplyr - Part 1

Introduction

According to a survey by CrowdFlower, data scientists spend most of their time cleaning and manipulating data rather than mining or modeling them for insights. As such, it becomes important to have tools that make data manipulation faster and easier. In today's chapter, we introduce you to dplyr, a grammar of data manipulation.

Libraries, Code & Data

We will use the following libraries:

- dplyr
- and readr

The data sets can be downloaded from here and the codes from here.

```
library(dplyr)
library(readr)
```

dplyr Verbs

dplyr provides a set of verbs that help us solve the most common data manipulation challenges while working with tabular data (dataframes, tibbles):

- `select`: returns subset of columns
- `filter`: returns a subset of rows
- `arrange`: re-order or arrange rows according to single/multiple variables
- `mutate`: create new columns from existing columns
- `summarise`: reduce data to a single summary

Case Study

We will explore a dummy data set that resembles web logs of an online retail company. You can download the data from here or import it directly using `read_csv()` from the readr package. We will use dplyr to answer the following:

- what is the average order value by device types?
- what is the average number of pages visited by purchasers and non-purchasers?
- what is the average time on site for purchasers vs non-purchasers?
- what is the average number of pages visited by purchasers and non-purchasers using mobile?

Data

```
ecom <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/web.csv')
ecom
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
```

```
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     4 bing     tablet false          3   18.0     468   China
##  5     5 yahoo    mobile true           9    1.00    955   Poland
##  6     6 yahoo    laptop false          5    5.00    135   South Africa
##  7     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  8     8 direct   mobile true          10    1.00    908   Indonesia
##  9     9 bing     mobile false          3   19.0     209   Netherlands
## 10    10 google   mobile true           6    1.00    208   Czech Republic
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Data Dictionary

Below is the description of the data set:

- id: row id
- referrer: referrer website/search engine
- os: operating system
- browser: browser
- device: device used to visit the website
- n_pages: number of pages visited
- duration: time spent on the website (in seconds)
- repeat: frequency of visits
- country: country of origin
- purchase: whether visitor purchased
- order_value: order value of visitor (in dollars)

Average Order Value

AOV by Devices

```
ecom %>%
  filter(purchase == 'true') %>%
  select(device, order_value, order_items) %>%
  group_by(device) %>%
  summarise_all(funs(sum)) %>%
  mutate(
    aov = order_value / order_items
  ) %>%
  select(device, aov)
```

```
## # A tibble: 3 x 2
##   device   aov
##   <chr>  <dbl>
## 1 laptop   353
## 2 mobile   280
## 3 tablet   261
```

Syntax

Before we start exploring the dplyr verbs, let us look at their syntax:

- the first argument is always a `data.frame` or `tibble`

- the subsequent arguments provide the information required for the verbs to take action
- the name of columns in the data need not be surrounded by quotes

Filter Rows

In order to compute the AOV, we must first separate the purchasers from non-purchasers. We will do this by filtering the data related to purchasers using the `filter()` function. It allows us to filter rows that meet a specific criteria/condition. The first argument is the name of the data frame and the rest of the arguments are expressions for filtering the data. Let us look at a few examples:

The first example we will look at filters all visits from device **mobile**. As we had learnt in the previous section, the first argument is our data set `ecom` and the next argument is the condition for filtering rows.

```
filter(ecom, device == "mobile")
```

```
## # A tibble: 344 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     5 yahoo    mobile true           9    1.00    955   Poland
##  2     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  3     8 direct   mobile true          10    1.00    908   Indonesia
##  4     9 bing     mobile false          3   19.0     209   Netherlands
##  5    10 google   mobile true           6    1.00    208   Czech Republic
##  6    13 direct   mobile false          9   14.0     406   Ireland
##  7    15 yahoo    mobile false          7    1.00     19.0 France
##  8    22 google   mobile true           5    1.00    147   Brazil
##  9    23 bing     mobile false          0    7.00    196   Russia
## 10    29 google   mobile true          10    1.00    338   Russia
## # ... with 334 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

We can specify multiple filtering conditions as well. In the below example, we specify two filter conditions:

- visit from device **tablet**
- resulted in a purchase or conversion

```
filter(ecom, device == "tablet", purchase == "true")
```

```
## # A tibble: 36 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     4 bing     tablet false          3    18.0      468 China
##  2    17 bing     tablet false          5    16.0      368 Peru
##  3    19 social   tablet false          7    10.0      290 Colombia
##  4    27 direct   tablet false          2    19.0      342 Japan
##  5    34 social   tablet false          9    20.0      420 Indonesia
##  6    94 yahoo    tablet false          2    16.0      480 China
##  7   101 yahoo    tablet false          2    14.0      364 Poland
##  8   158 google   tablet false          7    12.0      324 Philippines
##  9   166 direct   tablet false          6    20.0      580 Sudan
## 10   221 direct   tablet false          6    19.0      304 Indonesia
## # ... with 26 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Here is another example where we specify multiple conditions:

- visit from device **tablet**

- made a purchase
- browsed less than 15 pages

```
filter(ecom, device == "tablet", purchase == "true", n_pages < 15)
```

```
## # A tibble: 12 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1    19 social   tablet false          7    10.0      290 Colombia
##  2   101 yahoo    tablet false          2    14.0      364 Poland
##  3   158 google   tablet false          7    12.0      324 Philippines
##  4   225 direct   tablet false          3    12.0      324 Norway
##  5   292 yahoo    tablet false          0    13.0      390 Canada
##  6   445 social   tablet false          2    12.0      300 Philippines
##  7   472 direct   tablet false          6    13.0      338 Poland
##  8   561 yahoo    tablet false          2    10.0      280 China
##  9   713 social   tablet false         10    10.0      290 Philippines
## 10   785 direct   tablet false          3    10.0      260 Philippines
## 11   868 google   tablet false          9    14.0      308 Democratic Rep~
## 12   924 social   tablet false         10    11.0      330 China
## # ... with 3 more variables: purchase <chr>, order_items <dbl>,
## #   order_value <dbl>
```

Case Study

Let us apply what we have learnt to the case study. We want to filter all visits that resulted in a purchase.

```
filter(ecom, purchase == "true")
```

```
## # A tibble: 103 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     4 bing     tablet false          3    18.0      468 China
##  2    13 direct   mobile false          9    14.0      406 Ireland
##  3    17 bing     tablet false          5    16.0      368 Peru
##  4    19 social   tablet false          7    10.0      290 Colombia
##  5    27 direct   tablet false          2    19.0      342 Japan
##  6    34 social   tablet false          9    20.0      420 Indonesia
##  7    41 bing     mobile false          4    20.0      440 Czech Republic
##  8    94 yahoo    tablet false          2    16.0      480 China
##  9    98 bing     mobile false          3    18.0      288 Portugal
## 10   101 yahoo    tablet false          2    14.0      364 Poland
## # ... with 93 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Select Columns

After filtering the data, we need to select relevent variables to compute the AOV. Remember, we do not need all the columns in the data to compute a required metric (in our case, AOV). The `select()` function allows us to select a subset of columns. The first argument is the name of the data frame and the subsequent arguments specify the columns by name or position.

To select the `device` and `purchase` columns, we specify the data set i.e. `ecom` followed by the name of the columns.

```
select(ecom, device, purchase)
```

```
## # A tibble: 1,000 x 2
##    device purchase
##    <chr>  <chr>
##  1 laptop false
##  2 tablet false
##  3 laptop false
##  4 tablet true
##  5 mobile false
##  6 laptop false
##  7 mobile false
##  8 mobile false
##  9 mobile false
## 10 mobile false
## # ... with 990 more rows
```

We can select a set of columns using `:`. In the below example, we select all the columns starting from `device` up to `purchase`. Remember that we can use `:` only when the columns are adjacent to each other in the data set.

```
select(ecom, device:purchase)
```

```
## # A tibble: 1,000 x 7
##    device bouncers n_visit n_pages duration country        purchase
##    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>          <chr>
##  1 laptop true          10    1.00    693   Czech Republic false
##  2 tablet true           9    1.00    459   Yemen          false
##  3 laptop true           0    1.00    996   Brazil         false
##  4 tablet false          3   18.0     468   China          true
##  5 mobile true           9    1.00    955   Poland         false
##  6 laptop false          5    5.00    135   South Africa   false
##  7 mobile true          10    1.00     75.0 Bangladesh     false
##  8 mobile true          10    1.00    908   Indonesia      false
##  9 mobile false          3   19.0     209   Netherlands    false
## 10 mobile true           6    1.00    208   Czech Republic false
## # ... with 990 more rows
```

What if you want to select all columns except a few? Typing the name of many columns can be cumbersome and may also result in spelling errors. We may use `:` only if the columns are adjacent to each other but that may not always be the case. dplyr allows us to specify columns that need not be selected using `-`. In the below example, we select all columns except `id` and `country`. Notice the `-` before both of them.

```
select(ecom, -id, -country)
```

```
## # A tibble: 1,000 x 9
##    referrer device bouncers n_visit n_pages duration purchase order_items
##    <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>          <dbl>
##  1 google   laptop true          10    1.00    693   false           0
##  2 yahoo    tablet true           9    1.00    459   false           0
##  3 direct   laptop true           0    1.00    996   false           0
##  4 bing     tablet false          3   18.0     468   true            6.00
##  5 yahoo    mobile true           9    1.00    955   false           0
##  6 yahoo    laptop false          5    5.00    135   false           0
##  7 yahoo    mobile true          10    1.00     75.0 false           0
##  8 direct   mobile true          10    1.00    908   false           0
```

```
##  9 bing     mobile false          3   19.0     209   false           0
## 10 google   mobile true           6    1.00    208   false           0
## # ... with 990 more rows, and 1 more variable: order_value <dbl>
```

Case Study

For our case study, we need to select the columns `order_value` and `order_items` to calculate the AOV. We also need to select the `device` column as we are computing the AOV for each device type.

```
select(ecom, device, order_value, order_items)
```

```
## # A tibble: 1,000 x 3
##    device order_value order_items
##    <chr>        <dbl>       <dbl>
##  1 laptop           0        0
##  2 tablet           0        0
##  3 laptop           0        0
##  4 tablet         434        6.00
##  5 mobile           0        0
##  6 laptop           0        0
##  7 mobile           0        0
##  8 mobile           0        0
##  9 mobile           0        0
## 10 mobile           0        0
## # ... with 990 more rows
```

But we want the above data only for purchasers. Let us combine `filter()` and `select()` functions to extract `order_value` and `order_items` only for those visis that resulted in a purchase.

```
# filter all visits that resulted in a purchase
ecom1 <- filter(ecom, purchase == "true")

# select the relevant columns
ecom2 <- select(ecom1, device, order_value, order_items)

# view data
ecom2
```

```
## # A tibble: 103 x 3
##    device order_value order_items
##    <chr>        <dbl>       <dbl>
##  1 tablet         434        6.00
##  2 mobile         651        3.00
##  3 tablet        1049        6.00
##  4 tablet        1304        9.00
##  5 tablet         622        5.00
##  6 tablet        1613        7.00
##  7 mobile         184        3.00
##  8 tablet         286        9.00
##  9 mobile         764        6.00
## 10 tablet        1667        6.00
## # ... with 93 more rows
```

Grouping Data

We need to compute the total order value and total order items for each device in order to compute their AOV. To achieve this, we need to group the selected `order_value` and `order_items` by device type. `group_by()` allows us to group or split data based on particular (discrete) variable. The first argument is the name of the data set and the second argument is the name of the column based on which the data will be split.

To split the data by referrer type, we use `group_by` and specify the data set i.e. `ecom` and the column based on which to split the data i.e. `referrer`.

```
group_by(ecom, referrer)
```

```
## # A tibble: 1,000 x 11
## # Groups:   referrer [5]
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     4 bing     tablet false          3   18.0     468   China
##  5     5 yahoo    mobile true           9    1.00    955   Poland
##  6     6 yahoo    laptop false          5    5.00    135   South Africa
##  7     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  8     8 direct   mobile true          10    1.00    908   Indonesia
##  9     9 bing     mobile false          3   19.0     209   Netherlands
## 10    10 google   mobile true           6    1.00    208   Czech Republic
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Case Study

In the second line in the previous output, you can observe `Groups: referrer [5]` . The data is split into 5 groups as the referrer variable has 5 distinct values. For our case study, we need to group the data by `device` type.

```
# split ecom2 by device type
ecom3 <- group_by(ecom2, device)
ecom3
```

```
## # A tibble: 103 x 3
## # Groups:   device [3]
##    device order_value order_items
##    <chr>        <dbl>       <dbl>
##  1 tablet         434        6.00
##  2 mobile         651        3.00
##  3 tablet        1049        6.00
##  4 tablet        1304        9.00
##  5 tablet         622        5.00
##  6 tablet        1613        7.00
##  7 mobile         184        3.00
##  8 tablet         286        9.00
##  9 mobile         764        6.00
## 10 tablet        1667        6.00
## # ... with 93 more rows
```

Summarise Data

The next step is to compute the total order value and total order items for each device. i.e. we need to reduce the order value and order items data to a single summary. We can achieve this using `summarise()`. As usual, the first argument is the name of a data set and the subsequent arguments are functions that can summarise data. For example, we can use `min`, `max`, `sum`, `mean` etc.

Let us compute the average number of pages browsed by referrer type:

- split data by `referrer` type
- compute the average number of pages using `mean`

```
# split data by referrer type
step_1 <- group_by(ecom, referrer)

# compute average number of pages
step_2 <- summarise(step_1, mean(n_pages))
step_2
```

```
## # A tibble: 5 x 2
##   referrer `mean(n_pages)`
##   <chr>              <dbl>
## 1 bing                6.13
## 2 direct              6.38
## 3 google              5.73
## 4 social              5.42
## 5 yahoo               5.99
```

Now let us compute both the `mean` and the `median`.

```
# split data by referrer type
step_1 <- group_by(ecom, referrer)

# compute average number of pages
step_2 <- summarise(step_1, mean(n_pages), median(n_pages))
step_2
```

```
## # A tibble: 5 x 3
##   referrer `mean(n_pages)` `median(n_pages)`
##   <chr>              <dbl>             <dbl>
## 1 bing                6.13              1.00
## 2 direct              6.38              1.00
## 3 google              5.73              1.00
## 4 social              5.42              1.00
## 5 yahoo               5.99              2.00
```

Another way to achieve the above result is to use the `summarise_all()` function. How does that work? It generates the specified summary for all the columns in the data set except for the column based on which the data has been grouped or split. So we need to ensure that the data does not have any irrelevant columns.

- split data by `referrer` type
- select `order_value` and `order_items`
- compute the average number of pages by applying the `mean` function to all the columns

```
# select relevant columns
step_1 <- select(ecom, referrer, order_value, order_items)

# split data by referrer type
```

```
step_2 <- group_by(step_1, referrer)

# compute average number of pages
step_3 <- summarise_all(step_2, funs(mean))
step_3
```

```
## # A tibble: 5 x 3
##   referrer order_value order_items
##   <chr>          <dbl>       <dbl>
## 1 bing             316        1.22
## 2 direct           441        1.51
## 3 google           328        1.11
## 4 social           380        1.36
## 5 yahoo            470        1.71
```

Let us compute `mean` and `median` number of pages for each referre type using `summarise_all`.

```
# select relevant columns
step_1 <- select(ecom, referrer, order_value, order_items)

# split data by referrer type
step_2 <- group_by(step_1, referrer)

# compute mean and median number of pages
step_3 <- summarise_all(step_2, funs(mean, median))
step_3
```

```
## # A tibble: 5 x 5
##   referrer order_value_mean order_items_mean order_value_median
##   <chr>               <dbl>            <dbl>              <dbl>
## 1 bing                  316             1.22                  0
## 2 direct                441             1.51                  0
## 3 google                328             1.11                  0
## 4 social                380             1.36                  0
## 5 yahoo                 470             1.71                  0
## # ... with 1 more variable: order_items_median <dbl>
```

Case Study

So far, we have split the data based on the `device` type and we have selected 2 columns, `order_value` and `order_items`. We need the sum of order value and order items. What function can we use to obtain them? The `sum()` function will generate the sum of the values and hence we will use it inside the `summarise()` function. Remember, we need to provide a name to the summary being generated.

```
ecom4 <- summarise(ecom3, total_value = sum(order_value),
          total_items = sum(order_items))
ecom4
```

```
## # A tibble: 3 x 3
##   device total_value total_items
##   <chr>        <dbl>       <dbl>
## 1 laptop       56531         160
## 2 mobile       51504         184
## 3 tablet       51321         197
```

There you go, we have the total order value and total order items for each device type. If we use

`summarise_all()`, it will generate the summary for the selected columns based on the function specified. To specify the functions, we need to use another argument `funs` and it can take any number of valid functions.

```
ecom4 <- summarise_all(ecom3, funs(sum))
ecom4
```

```
## # A tibble: 3 x 3
##   device order_value order_items
##   <chr>        <dbl>       <dbl>
## 1 laptop       56531         160
## 2 mobile       51504         184
## 3 tablet       51321         197
```

Create Columns

To create a new column, we will use `mutate()`. The first argument is the name of the data set and the subsequent arguments are expressions for creating new columns based out of existing columns.

Let us add a new column `avg_page_time` i.e. time on site divided by number of pages visited.

```
# select duration and n_pages from ecom
mutate_1 <- select(ecom, n_pages, duration)
mutate(mutate_1, avg_page_time = duration / n_pages)
```

```
## # A tibble: 1,000 x 3
##    n_pages duration avg_page_time
##      <dbl>    <dbl>         <dbl>
##  1    1.00    693           693
##  2    1.00    459           459
##  3    1.00    996           996
##  4   18.0     468            26.0
##  5    1.00    955           955
##  6    5.00    135            27.0
##  7    1.00     75.0          75.0
##  8    1.00    908           908
##  9   19.0     209            11.0
## 10    1.00    208           208
## # ... with 990 more rows
```

We can create new columns based on other columns created using `mutate`. Let us create another column `sqrt_avg_page_time` i.e. square root of the average time on page using `avg_page_time`.

```
mutate(mutate_1,
       avg_page_time = duration / n_pages,
       sqrt_avg_page_time = sqrt(avg_page_time))
```

```
## # A tibble: 1,000 x 4
##    n_pages duration avg_page_time sqrt_avg_page_time
##      <dbl>    <dbl>         <dbl>              <dbl>
##  1    1.00    693           693                26.3
##  2    1.00    459           459                21.4
##  3    1.00    996           996                31.6
##  4   18.0     468            26.0               5.10
##  5    1.00    955           955                30.9
##  6    5.00    135            27.0               5.20
##  7    1.00     75.0          75.0               8.66
```

```
##  8    1.00    908           908                30.1
##  9   19.0     209            11.0               3.32
## 10    1.00    208           208                14.4
## # ... with 990 more rows
```

Case Study

Back to our case study, from the last step we have the total order value and total order items for each device category and can compute the AOV. We will create a new column to store AOV.

```
ecom5 <- mutate(ecom4, aov = order_value / order_items)
ecom5
```

```
## # A tibble: 3 x 4
##   device order_value order_items   aov
##   <chr>        <dbl>       <dbl> <dbl>
## 1 laptop       56531         160   353
## 2 mobile       51504         184   280
## 3 tablet       51321         197   261
```

Select Columns

The last step is to select the relevant columns. We will select the device type and the corresponding aov while getting rid of other columns. Use `select()` to extract the relevant columns.

```
ecom6 <- select(ecom5, device, aov)
ecom6
```

```
## # A tibble: 3 x 2
##   device   aov
##   <chr>  <dbl>
## 1 laptop   353
## 2 mobile   280
## 3 tablet   261
```

Arrange Data

Arranging data in ascending or descending order is one of the most common tasks in data manipulation. We can use `arrange` to arrange data by different columns. Let us say we want to arrange data by the number of pages browsed.

```
arrange(ecom, n_pages)
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     5 yahoo    mobile true           9    1.00    955   Poland
##  5     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  6     8 direct   mobile true          10    1.00    908   Indonesia
##  7    10 google   mobile true           6    1.00    208   Czech Republic
##  8    11 direct   laptop true           9    1.00    738   Jamaica
##  9    15 yahoo    mobile false          7    1.00     19.0 France
```

```
## 10    16 bing     laptop true           1    1.00    995   United States
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

If we want to arrange the data in descending order, we can use `desc()`. Let us arrange the data in descending order.

```
arrange(ecom , desc(n_pages))
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1    34 social   tablet false          9    20.0      420 Indonesia
##  2    41 bing     mobile false          4    20.0      440 Czech Republic
##  3   136 yahoo    tablet false          0    20.0      200 Indonesia
##  4   166 direct   tablet false          6    20.0      580 Sudan
##  5   219 social   mobile false          1    20.0      520 United States
##  6   253 google   mobile false          8    20.0      300 Sweden
##  7   276 social   laptop false          4    20.0      200 Indonesia
##  8   314 yahoo    mobile false          3    20.0      480 China
##  9   348 social   laptop false         10    20.0      280 Japan
## 10   373 yahoo    mobile false          2    20.0      240 Portugal
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Data can be arranged by multiple variables as well. Let us arrange data first by number of visits and then by number of pages in a descending order.

```
arrange(ecom, n_visit, desc(n_pages))
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1   136 yahoo    tablet false          0    20.0      200 Indonesia
##  2   448 google   laptop false          0    19.0      418 Ukraine
##  3   402 bing     laptop false          0    18.0      180 Russia
##  4   642 yahoo    laptop false          0    18.0      522 Syria
##  5   884 direct   tablet false          0    18.0      252 Brazil
##  6   651 social   laptop false          0    17.0      204 China
##  7   749 bing     laptop false          0    17.0      272 Indonesia
##  8   886 bing     mobile false          0    16.0      272 Peru
##  9   871 yahoo    mobile false          0    15.0      255 China
## 10   988 direct   laptop false          0    15.0      255 Indonesia
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Case Study

If you observe `ecom6`, the `aov` column is arranged in descending order.

```
arrange(ecom6, aov)
```

```
## # A tibble: 3 x 2
##   device   aov
##   <chr>  <dbl>
## 1 tablet   261
## 2 mobile   280
```

```
## 3 laptop   353
```

AOV by Devices

Let us combine all the code from the above steps:

```
ecom1 <- filter(ecom, purchase == "true")
ecom2 <- select(ecom1, device, order_value, order_items)
ecom3 <- group_by(ecom2, device)
ecom4 <- summarise_all(ecom3, funs(sum))
ecom5 <- mutate(ecom4, aov = order_value / order_items)
ecom6 <- select(ecom5, device, aov)
ecom7 <- arrange(ecom6, aov)
ecom7
```

```
## # A tibble: 3 x 2
##   device   aov
##   <chr>  <dbl>
## 1 tablet   261
## 2 mobile   280
## 3 laptop   353
```

If you observe, at each step we create a new variable(data frame) and then use it as an input in the next step i.e. the output from one step becomes the input for the next. Can we achieve the final outcome i.e. `ecom7` without creating the intermediate data (ecom1 - ecom6)? Yes, we can. We will use the **%>%** operator to chain the steps and get rid of the intermediate data.

```
ecom %>%
  filter(purchase == 'true') %>%
  select(device, order_value, order_items) %>%
  group_by(device) %>%
  summarise_all(funs(sum)) %>%
  mutate(
    aov = order_value / order_items
  ) %>%
  select(device, aov) %>%
  arrange(aov)
```

```
## # A tibble: 3 x 2
##   device   aov
##   <chr>  <dbl>
## 1 tablet   261
## 2 mobile   280
## 3 laptop   353
```

Below we map the description of each step to dplyr verbs.

Your Turn

- what is the average number of pages visited by purchasers and non-purchasers?
- what is the average time on site for purchasers vs non-purchasers?
- what is the average number of pages visited by purchasers and non-purchasers using mobile?

Data Wrangling with dplyr - Part 2

Introduction

In the previous chapter, we learnt to combine tables using dplyr. In this chapter, we will explore a set of helper functions in order to:

- extract unique rows
- rename columns
- sample data
- extract columns
- slice rows
- arrange rows
- compare tables
- extract/mutate data using predicate functions
- count observations for different levels of a variable

Libraries, Code & Data

We will use the following packages:

- dplyr
- readr

The data sets can be downloaded from here and the codes from here.

```
library(dplyr)
library(readr)
```

Case Study

Let us look at a case study (e-commerce data) and see how we can use dplyr helper functions to answer questions we have about and to modify/transform the underlying data set.

Data

```
ecom <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/web.csv')
ecom
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     4 bing     tablet false          3   18.0     468   China
##  5     5 yahoo    mobile true           9    1.00    955   Poland
##  6     6 yahoo    laptop false          5    5.00    135   South Africa
##  7     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  8     8 direct   mobile true          10    1.00    908   Indonesia
##  9     9 bing     mobile false          3   19.0     209   Netherlands
## 10    10 google   mobile true           6    1.00    208   Czech Republic
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
```

```
## #   order_items <dbl>, order_value <dbl>
```

Data Dictionary

- id: row id
- referrer: referrer website/search engine
- os: operating system
- browser: browser
- device: device used to visit the website
- n_pages: number of pages visited
- duration: time spent on the website (in seconds)
- repeat: frequency of visits
- country: country of origin
- purchase: whether visitor purchased
- order_value: order value of visitor (in dollars)

Data Sanitization

Let us ensure that the data is sanitized by checking the sources of traffic and devices used to visit the site. We will use `distinct` to examine the values in the `referrer` column

```
ecom %>%
  distinct(referrer)
```

```
## # A tibble: 5 x 1
##   referrer
##   <chr>
## 1 google
## 2 yahoo
## 3 direct
## 4 bing
## 5 social
```

and the `device` column as well.

```
ecom %>%
  distinct(device)
```

```
## # A tibble: 3 x 1
##   device
##   <chr>
## 1 laptop
## 2 tablet
## 3 mobile
```

Data Tabulation

Let us now look at the proportion or share of visits driven by different sources of traffic.

```
ecom %>%
  group_by(referrer) %>%
  tally()
```

```
## # A tibble: 5 x 2
```

```
##   referrer     n
##   <chr>    <int>
## 1 bing       194
## 2 direct     191
## 3 google     208
## 4 social     200
## 5 yahoo      207
```

We would also like to know the number of bouncers driven by the different sources of traffic.

```
ecom %>%
  group_by(referrer, bouncers) %>%
  tally()
```

```
## # A tibble: 10 x 3
## # Groups:   referrer [?]
##    referrer bouncers     n
##    <chr>    <chr>    <int>
##  1 bing     false      104
##  2 bing     true        90
##  3 direct   false       98
##  4 direct   true        93
##  5 google   false      101
##  6 google   true       107
##  7 social   false       93
##  8 social   true       107
##  9 yahoo    false      110
## 10 yahoo    true        97
```

Let us look at how many conversions happen across different devices.

```
ecom %>%
  group_by(device, purchase) %>%
  tally() %>%
  filter(purchase == 'true')
```

```
## # A tibble: 3 x 3
## # Groups:   device [3]
##   device purchase     n
##   <chr>  <chr>    <int>
## 1 laptop true        31
## 2 mobile true        36
## 3 tablet true        36
```

Another way to extract the above information is by using `count`

```
ecom %>%
  count(referrer, purchase) %>%
  filter(purchase == "true")
```

```
## # A tibble: 5 x 3
##   referrer purchase     n
##   <chr>    <chr>    <int>
## 1 bing     true        17
## 2 direct   true        25
## 3 google   true        19
## 4 social   true        20
## 5 yahoo    true        22
```

Sampling Data

dplyr offers sampling functions which allow us to specify either the number or percentage of observations. `sample_n()` allows sampling a specific number of observations.

```
ecom %>%
  sample_n(700)
```

```
## # A tibble: 700 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1   488 google   mobile true           9    1.00    201   China
##  2   302 google   laptop false          6    9.00    135   Greece
##  3   704 yahoo    laptop false          1   11.0     231   China
##  4   847 yahoo    mobile true           0    1.00    895   Russia
##  5    73 google   tablet true           4    1.00    565   China
##  6   541 bing     laptop true           5    1.00    700   France
##  7   708 yahoo    laptop false          7    6.00    150   Philippines
##  8    68 direct   mobile true           7    1.00    912   France
##  9   498 bing     mobile false          0    4.00     72.0 Iran
## 10   611 yahoo    mobile true           1    1.00    710   Ukraine
## # ... with 690 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

We can combine the sampling functions with other dplyr functions as shown below where we sample observation after grouping them according to the source of traffic.

```
ecom %>%
  group_by(referrer) %>%
  sample_n(100)
```

```
## # A tibble: 500 x 11
## # Groups:   referrer [5]
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1   828 bing     tablet false          3   15.0     390   Peru
##  2   248 bing     tablet true           5    1.00    151   Macedonia
##  3   149 bing     tablet false          6    3.00     54.0 Sweden
##  4   869 bing     laptop true           6    1.00    199   China
##  5    50 bing     tablet true           5    1.00    831   Iran
##  6   303 bing     mobile true           4    1.00    979   Madagascar
##  7   652 bing     tablet false          6    9.00    198   Thailand
##  8    52 bing     mobile true           7    1.00    569   Brazil
##  9   655 bing     mobile true           6    1.00    604   Nigeria
## 10   574 bing     mobile true          10    1.00     98.0 Poland
## # ... with 490 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

`sample_frac()` allows a specific percentage of observations.

```
ecom %>%
  sample_frac(size = 0.7)
```

```
## # A tibble: 700 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1   135 social   mobile false          0    3.00     78.0 Indonesia
```

```
##  2   106 direct   mobile true           6    1.00    227   China
##  3   179 google   laptop true           2    1.00    773   Nigeria
##  4   846 direct   tablet true           9    1.00    709   Russia
##  5   485 bing     tablet true           7    1.00    423   Qatar
##  6   265 bing     laptop true          10    1.00    777   Japan
##  7   243 bing     tablet false          6    1.00     22.0 Russia
##  8   916 social   mobile false          7   12.0     192   Philippines
##  9   307 google   laptop false          3    8.00    192   Greece
## 10   892 google   tablet true           8    1.00    891   China
## # ... with 690 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Data Extraction

In the previous chapter, we had observed that dplyr verbs always returned a tibble. What if you want to extract a specific column or a bunch of rows but not as a tibble?

Use `pull` to extract columns either by name or position. It will return a vector. In the below example, we extract the `device` column as a vector. I am using `head` in addition to limit the output printed.

```
ecom %>%
  pull(device) %>%
  head
```

```
## [1] "laptop" "tablet" "laptop" "tablet" "mobile" "laptop"
```

Let us extract the first column from `ecom` using column position instead of name.

```
ecom %>%
  pull(1) %>%
  head
```

```
## [1] 1 2 3 4 5 6
```

You can use `-` before the column position to indicate the position in reverse. The below example extracts data from the last column.

```
ecom %>%
  pull(-1) %>%
  head
```

```
## [1]   0   0   0 434   0   0
```

Let us now look at extracting rows using `slice()`. In the below example, we extract data starting from the 5th row and upto the 15th row.

```
ecom %>%
  slice(5:15)
```

```
## # A tibble: 11 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     5 yahoo    mobile true           9    1.00    955   Poland
##  2     6 yahoo    laptop false          5    5.00    135   South Africa
##  3     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  4     8 direct   mobile true          10    1.00    908   Indonesia
##  5     9 bing     mobile false          3   19.0     209   Netherlands
##  6    10 google   mobile true           6    1.00    208   Czech Republic
##  7    11 direct   laptop true           9    1.00    738   Jamaica
```

```
##  8    12 direct   tablet false          6   12.0     132   Estonia
##  9    13 direct   mobile false          9   14.0     406   Ireland
## 10    14 yahoo    tablet false          5    8.00     80.0 Philippines
## 11    15 yahoo    mobile false          7    1.00     19.0 France
## # ... with 3 more variables: purchase <chr>, order_items <dbl>,
## #   order_value <dbl>
```

Use `n()` inside `slice()` to extract the last row.

```
ecom %>%
  slice(n())
```

```
## # A tibble: 1 x 11
##      id referrer device bouncers n_visit n_pages duration country purchase
##   <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>   <chr>
## 1  1000 google   mobile true           9    1.00      269 China   false
## # ... with 2 more variables: order_items <dbl>, order_value <dbl>
```

Between

`between()` allows us to test if the values in a column lie between two specific values. In the below example, we check how many visits browsed pages between 5 and 15.

```
ecom_sample <-
  ecom %>%
  sample_n(30)

ecom_sample %>%
  pull(n_pages) %>%
  between(5, 15)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [12]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE
## [23]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
```

Case When

`case_when()` is an alternative to `if else`. It allows us to lay down the conditions clearly and makes the code more readable. In the below example, we create a new column `repeat_visit` from `n_visit` (the number of previous visits).

```
ecom %>%
  mutate(
    repeat_visit = case_when(
      n_visit > 0 ~ TRUE,
      TRUE ~ FALSE
    )
  ) %>%
  select(n_visit, repeat_visit)
```

```
## # A tibble: 1,000 x 2
##    n_visit repeat_visit
##      <int> <lgl>
##  1      10 T
##  2       9 T
```

```
##  3       0 F
##  4       3 T
##  5       9 T
##  6       5 T
##  7      10 T
##  8      10 T
##  9       3 T
## 10       6 T
## # ... with 990 more rows
```

Data Visulaization - Introduction

Introduction

In this chapter, we will:

- understand the philosophy of Grammar of Graphics
- explore different aspects of ggplot2
- learn to build some of the basic plots regularly used for exploring data

ggplot2 is an **awesome** alternative to base R for data visualization. It is based on The Grammar of Graphics. In this chapter, we will understand the philosophy behind **ggplot2** and learn to build some of the most frequently used plots for visualizing data.

Libraries, Code & Data

We will use the following libraries in this chapter:

- readr
- ggplot2

All the data sets used in this chapter can be found here and code can be downloaded from here.

Grammar of Graphics

Grammar of graphics is a formal system for building plots. The core idea is that any plot can be uniquely described as a combination of

- a dataset
- a geom
- a set of mappings
- a statistic
- a position adjustment
- a coordinate system
- a faceting scheme

Data

Let us build a scatter plot from scratch using the `mtcars` data. We will build the plot incrementally and understand the above layers. The first step in any data visualization exercise is to identify the data set. In `ggplot2`, we can specify the data set using `ggplot()`.

```
ggplot(data = mtcars)
```

If you observe, `ggplot()` does not generate any plot, it just creates a coordinate system.

Geom

After specifying the data set, we have to decide how the data will be visualized. We will do this using `geoms`. It basically details the geometric shapes that must be used to display the data. In our case, we want the data to displayed as `points`.

There are several `geoms` and we will explore them one by one. For the time time being, let us use `geom_point()`. This tells ggplot2 it must use points to represent the data. The next step is to specify the variables that will be represented by the `X` and `Y` axis. To do this we will use `mapping` and `aes`.

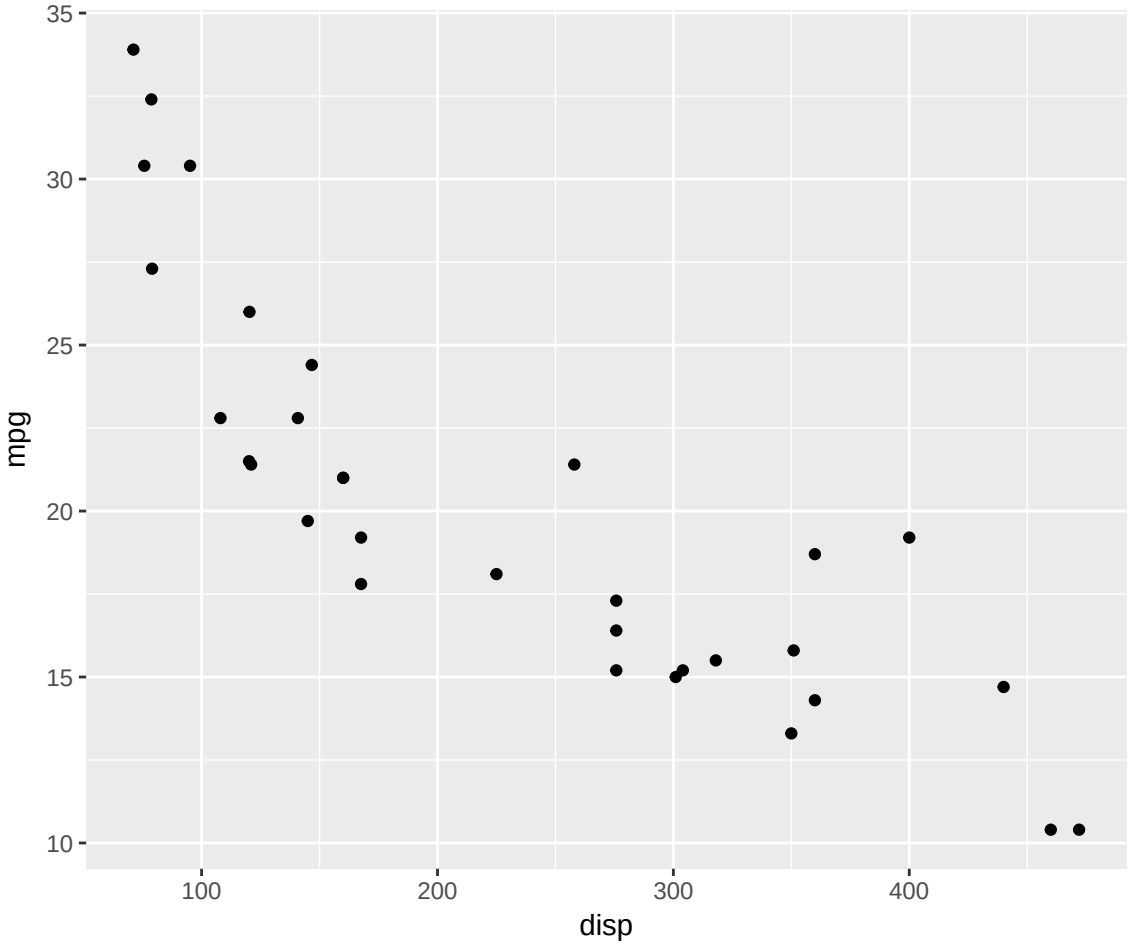
`aes` is the short for aesthetics. Using `mapping`, we can map variables to aesthetics. We specify the aesthetic type and the corresponding variable within `aes`.In our example, we want the `X` axis to be represented by `disp` and `Y` axis by `mpg`. ggplot2 will search for these variables in the data we have provided in `ggplot` which is `mtcars`. If ggplot2 can't find the variables, it will return an error.

So far we have provided:

- data set
- geometric shape to represent data
- variables to represent `X` and `Y` axis

The above layers are the bare minimum required to create a plot in ggplot2.

```
ggplot(data = mtcars) +
  geom_point(mapping = aes(x = disp, y = mpg))
```



Aesthetics

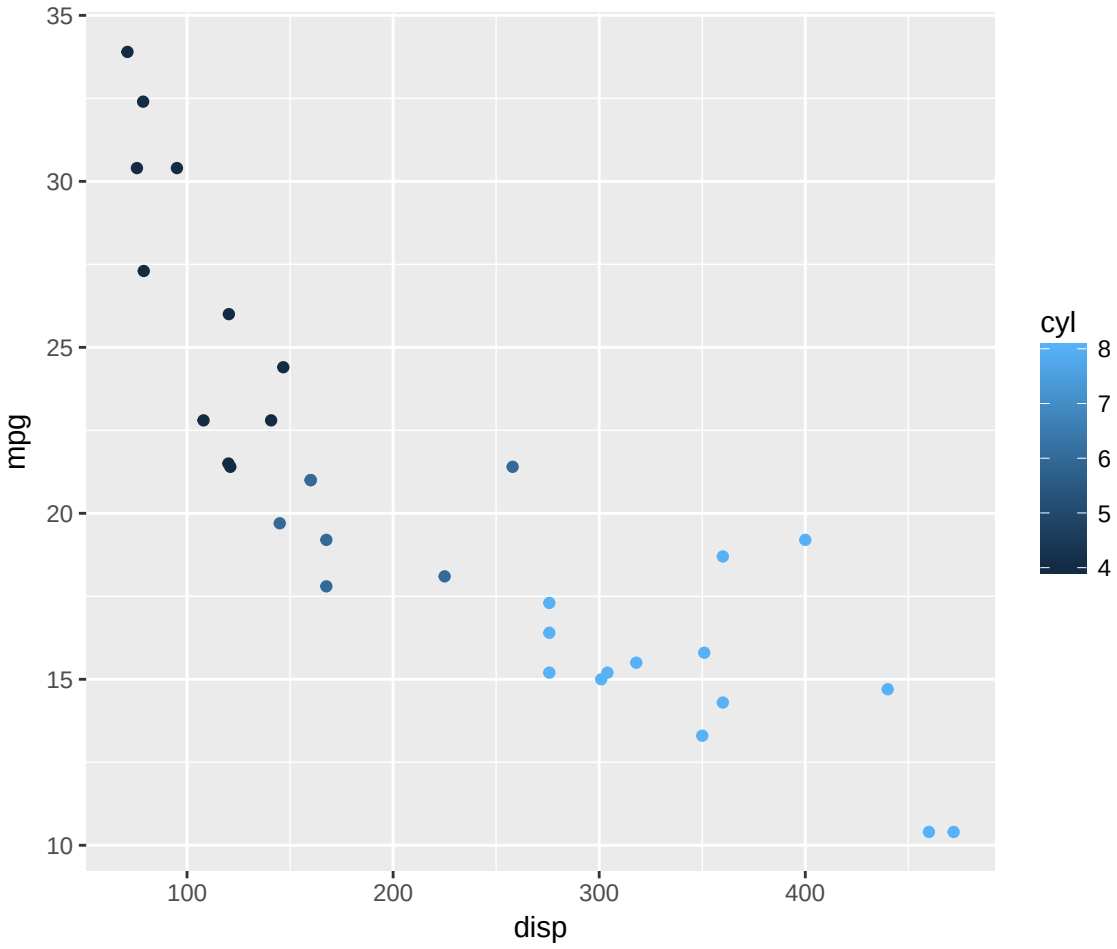
Aestheics are the visual properties of the objects in the plot. We can display the geometric object in different ways by changing the values of its aesthetic properties such as:

- shape
- size
- color
- opaqueness

Let us modify the appearance of the scatter plot by changing the following:

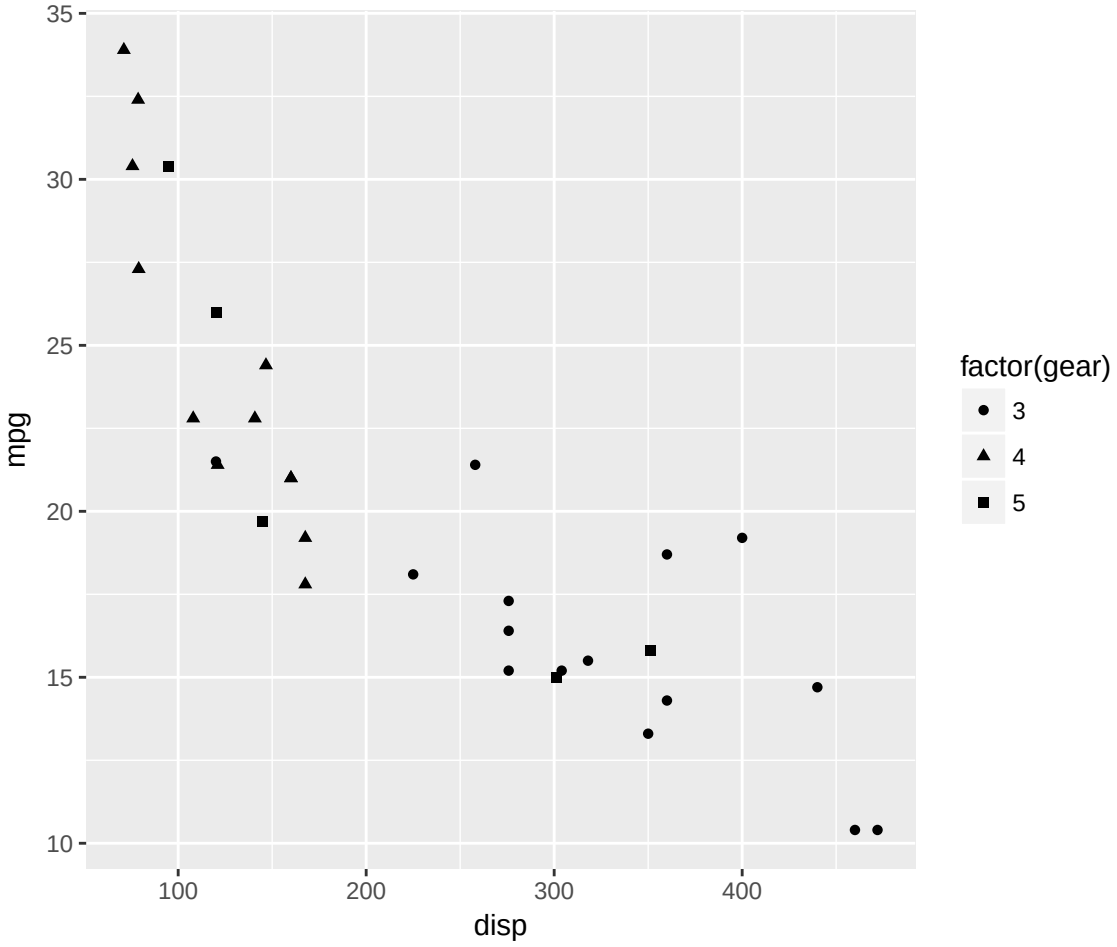
Color

```
ggplot(data = mtcars) +
  geom_point(mapping = aes(x = disp, y = mpg, color = cyl))
```



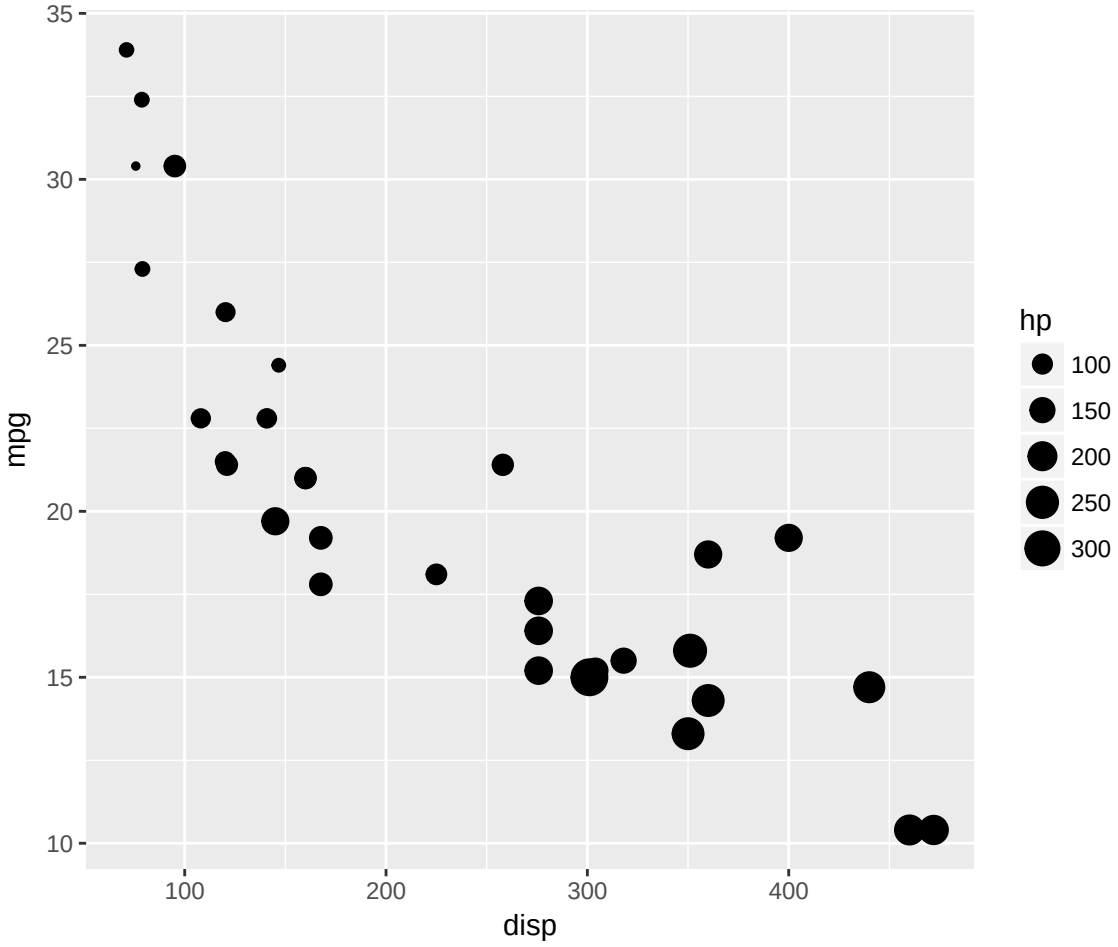
Shape

```
ggplot(data = mtcars) +
  geom_point(mapping = aes(x = disp, y = mpg, shape = factor(gear)))
```



Size

```
ggplot(data = mtcars) +
  geom_point(mapping = aes(x = disp, y = mpg, size = hp))
```



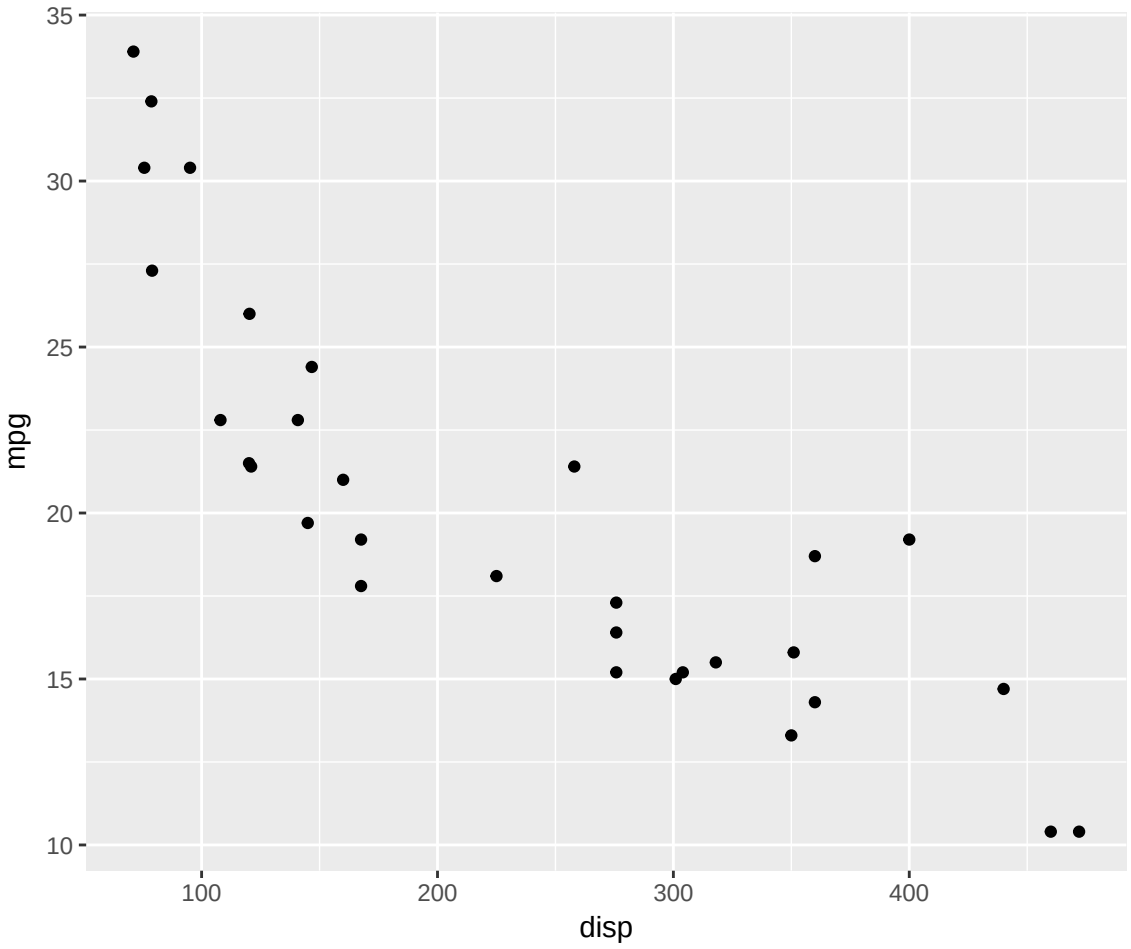
Stat

Some graphs plot the raw data set, but others like bar plot, box plot and histograms compute new values and plot them. In this section, we will look at how the data is transformed before representing them in a plot.

Plot unique values

Let us say we want to remove duplicates from the scatter plot. We can use `stat_unique()` which removes duplicate values before plotting the data.

```
ggplot(data = mtcars) +
  geom_point(mapping = aes(x = disp, y = mpg), stat = "unique")
```



Position

Sometimes the geoms tend to overlap each other. In such cases, we might want to reposition them to aid better visualization. In this section, we will learn to adjust the position of the geoms using:

- `position_dodge`
- `position_identity`
- `position_jitter`
- `position_fill`

Coordinate System

The default coordinate system of ggplot2 is the cartesian coordinate system. In this section, we will learn to tweak the system using different functions such as:

- `coord_flip`

- `coord_polar`

Facet

Faceting allows us to generate multiple visualizations for different subsets of data. In this section, we will generate multiple plots using:

- `facet_grid`
- `facet_wrap`

Summary

In this chapter, we learnt

- about grammar of graphics
- the components of a plot/chart
- how to build a plot step by step

Up Next..

In the next chapter, we will learn to quickly build a set of plots/charts that are routinely used in exploring data.

Data Visulaization - Geoms

Introduction

In the previous chapter, we learnt how to create plots using the `qplot()` function. In this chapter, we will create some of the most routinely used plots to explore data using the `geom_*` functions.

Libraries, Code & Data

We will use the following libraries in this chapter:

- readr
- ggplot2
- tibble
- dplyr

All the data sets used in this chapter can be found here and code can be downloaded from here.

Data

```
ecom <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/web.csv')
ecom
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     4 bing     tablet false          3   18.0     468   China
##  5     5 yahoo    mobile true           9    1.00    955   Poland
##  6     6 yahoo    laptop false          5    5.00    135   South Africa
##  7     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  8     8 direct   mobile true          10    1.00    908   Indonesia
##  9     9 bing     mobile false          3   19.0     209   Netherlands
## 10    10 google   mobile true           6    1.00    208   Czech Republic
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Data Dictionary

- id: row id
- referrer: referrer website/search engine
- os: operating system
- browser: browser
- device: device used to visit the website
- n_pages: number of pages visited
- duration: time spent on the website (in seconds)
- repeat: frequency of visits
- country: country of origin
- purchase: whether visitor purchased
- order_value: order value of visitor (in dollars)

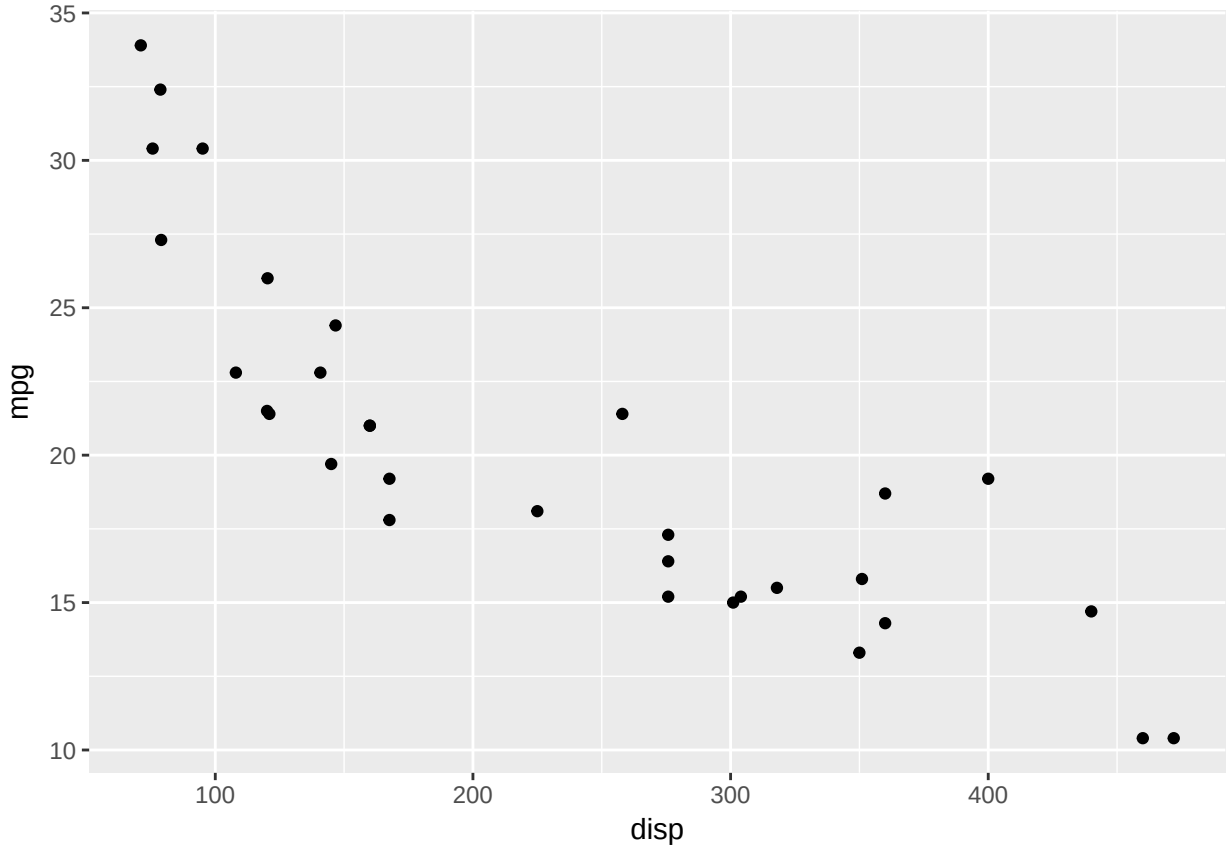
Scatter Plot

A scatter plot displays the relationship between two continuous variables. In ggplot2, we can build a scatter plot using `geom_point()`. Scatterplots can show you visually:

- the strength of the relationship between the variables
- the direction of the relationship between the variables
- and whether outliers exist

Point

```
ggplot(mtcars, aes(x = disp, y = mpg)) +
  geom_point()
```



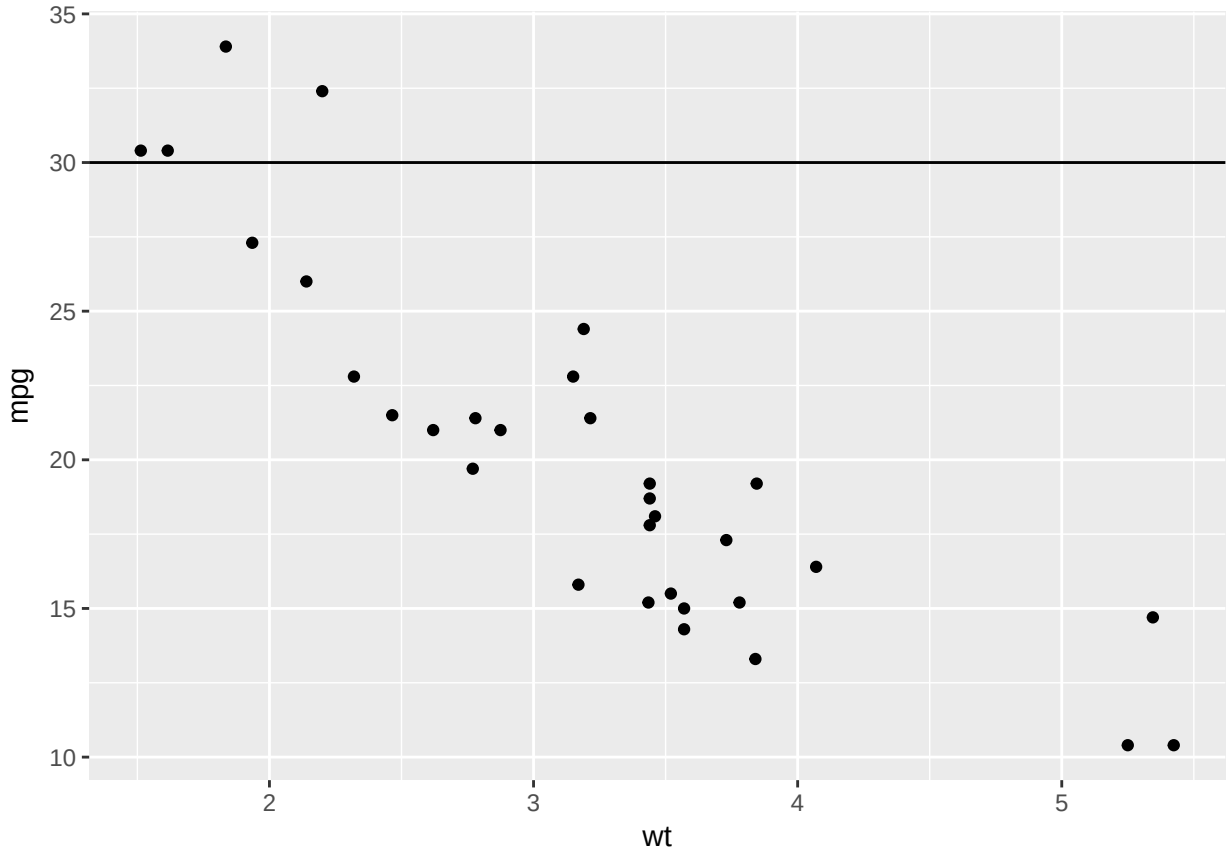
Horizontal/Vertical Lines

Add horizontal or vertical lines using:

- `geom_hline()`
- `geom_vline()`

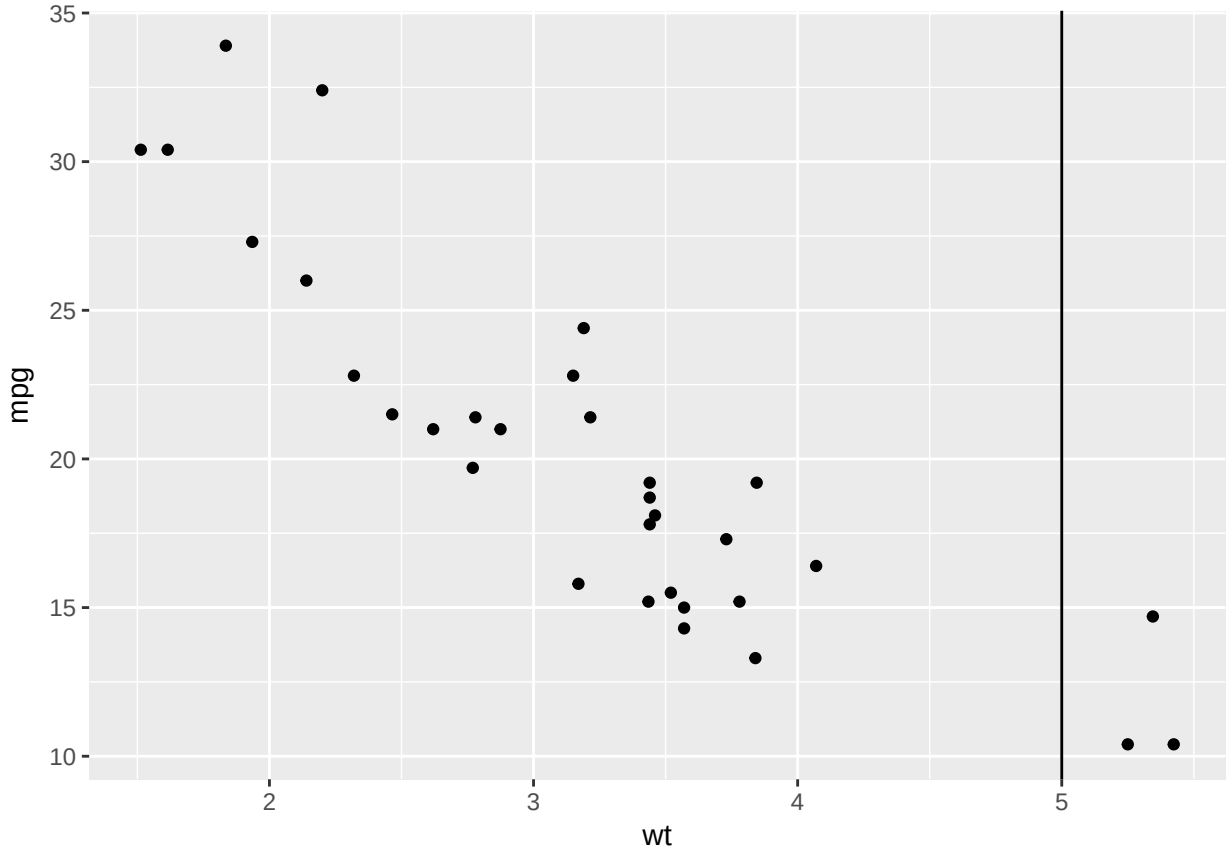
To add a horizontal line, we need to specify the location on the Y axis where the line will be drawn. Use `yintercept` to specify the location of the line.

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_hline(yintercept = 30)
```



For the vertical line, we need to specify the location on the X axis using `xintercept`.

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_vline(xintercept = 5)
```



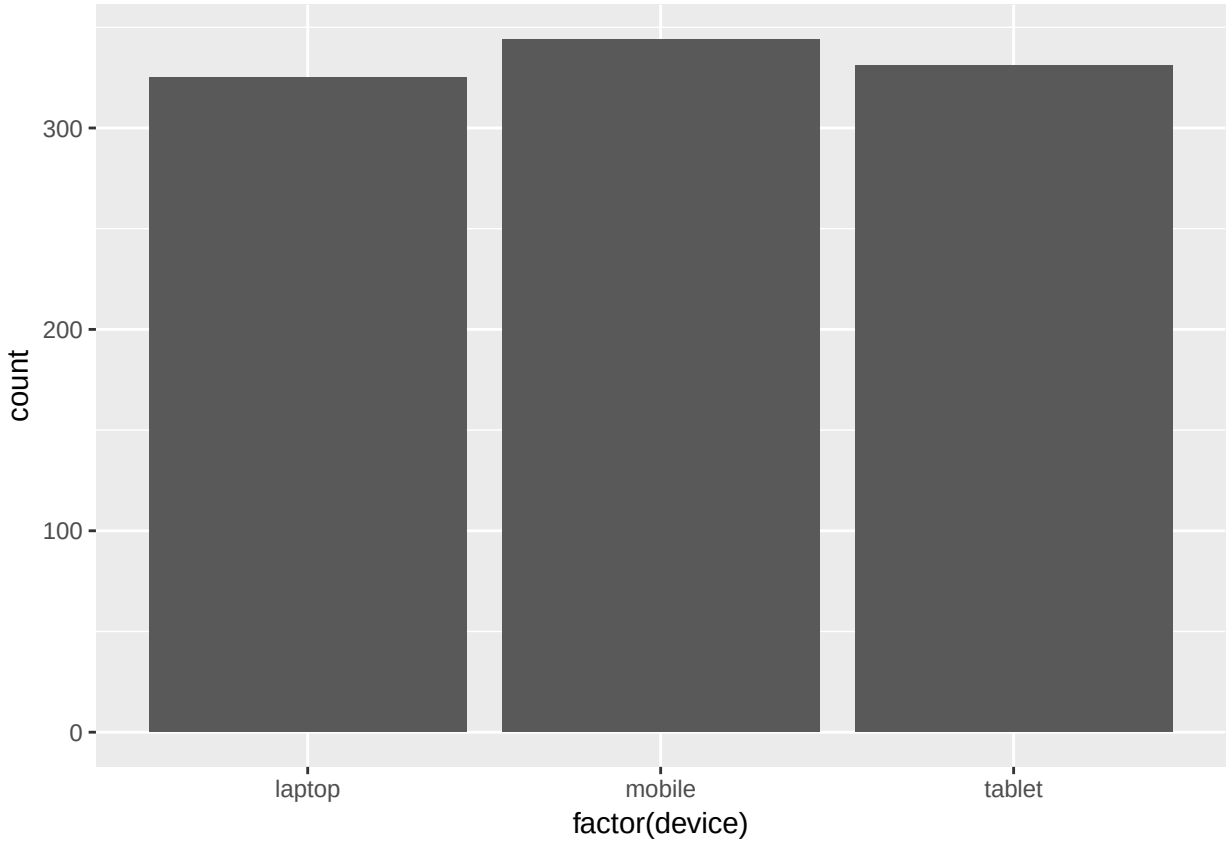
Bar Plot

Bar plots present grouped data with rectangular bars. The bars may represent the frequency of the groups or values. Bar plots can be:

- horizontal
- vertical
- grouped
- stacked
- proportional

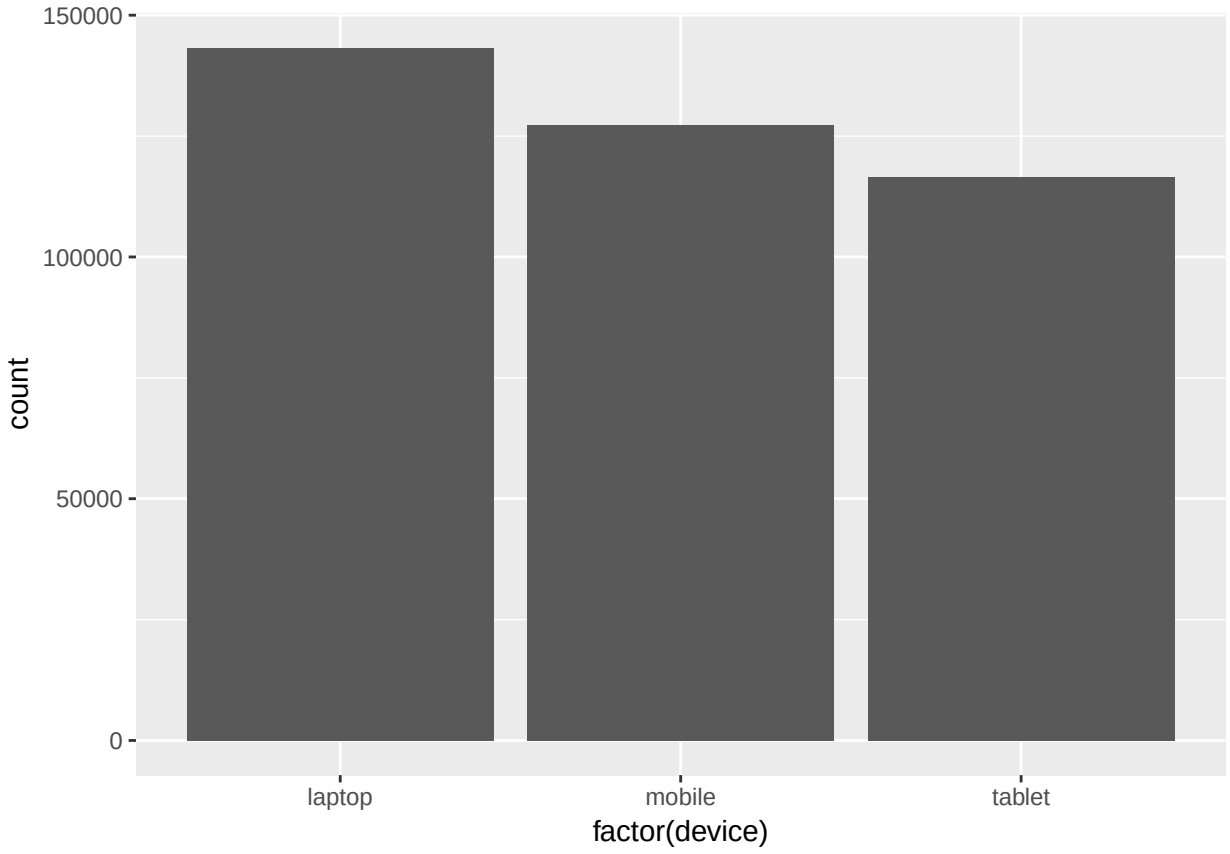
Let us build a simple bar plot to visualize the traffic driven by different device types.

```
ggplot(ecom, aes(x = factor(device))) +
  geom_bar()
```



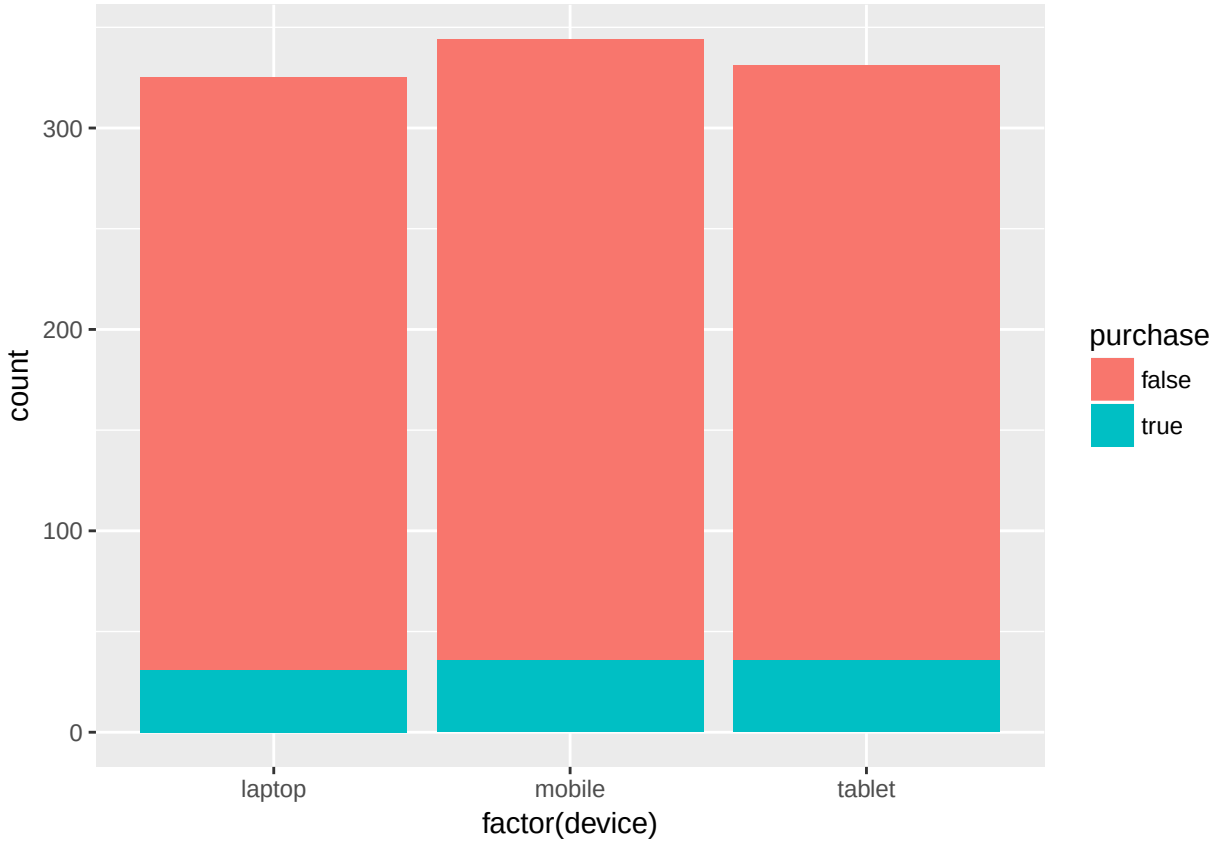
Instead of the frequency of visits, we can visualize the order value driven by the different devices. Use the `weight` argument within `aes` to indicate that the bars should represent the variable specified and not the frequency.

```
ggplot(ecom, aes(x = factor(device))) +
  geom_bar(aes(weight = order_value))
```



To view the proportion of purchasers and non-purchasers for each device type, we will map `fill` to `purchase`. The color of the bar represents purchasers and non-purchasers.

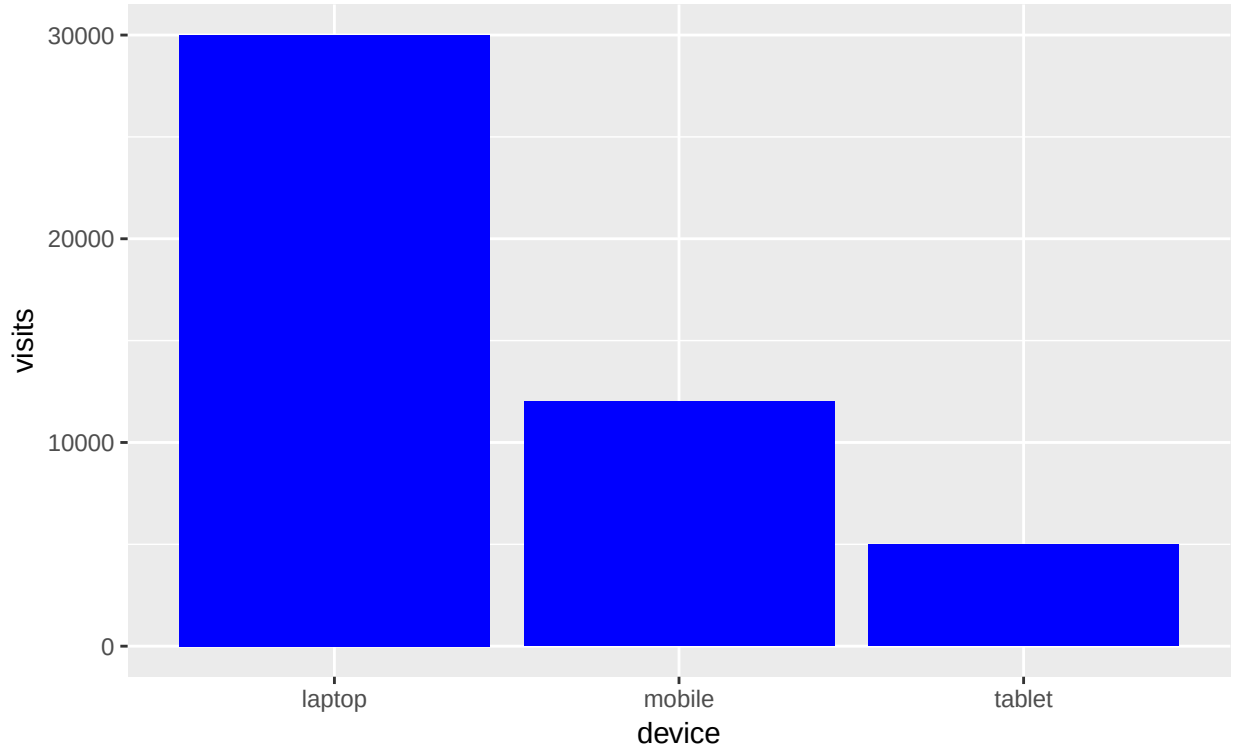
```
ggplot(ecom, aes(x = factor(device))) +
  geom_bar(aes(fill = purchase))
```



Columns

In some instances, we do not have access to the raw data and are provided summaries or transformed data. In the below example, we have data that summarizes the visits from each device type. Such data can be visualized using `geom_col()`.

```r
device <- c('laptop', 'mobile', 'tablet')
visits <- c(30000, 12000, 5000)
traffic <- tibble::tibble(device, visits)
ggplot(traffic, aes(x = device, y = visits)) +
  geom_col(fill = 'blue')
```



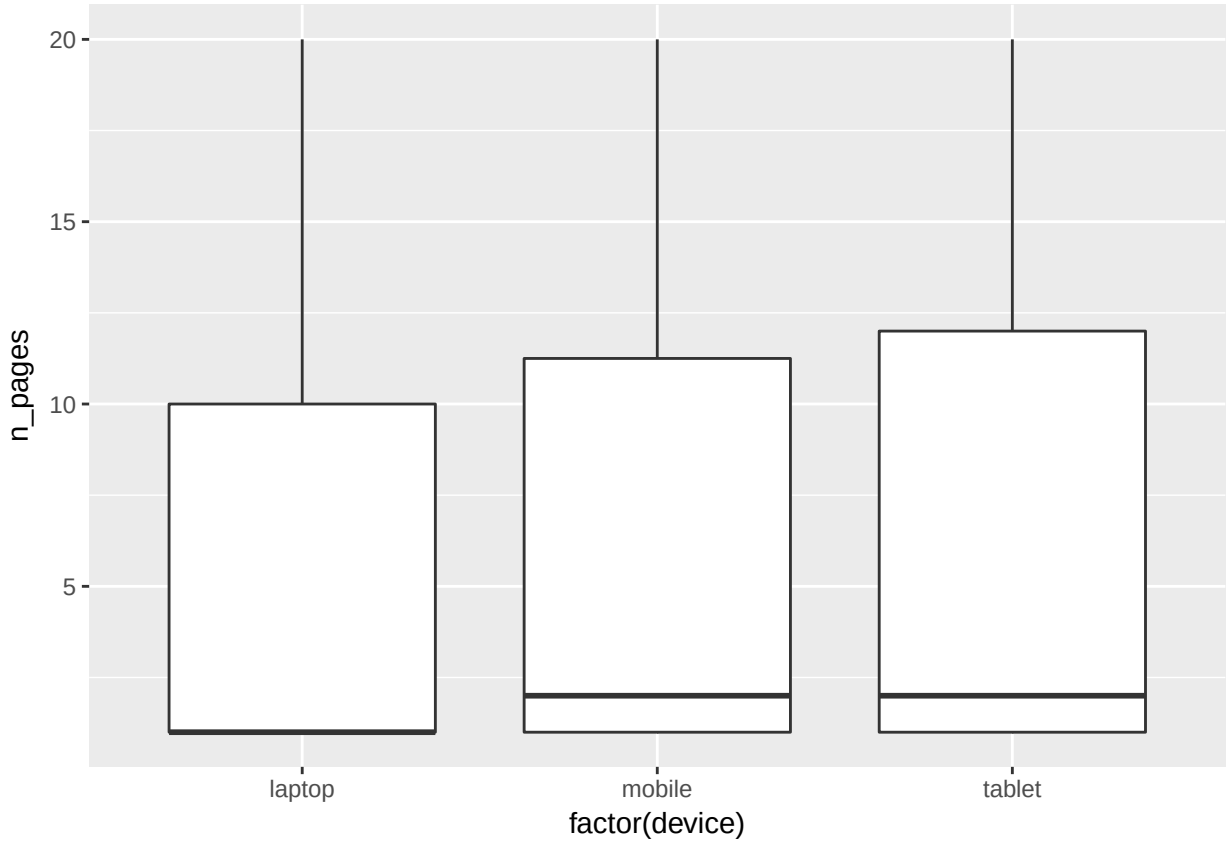
Boxplot

Box plots can be used to:

- examine the distribution of a variable
- detect outliers, boxplots are very handy

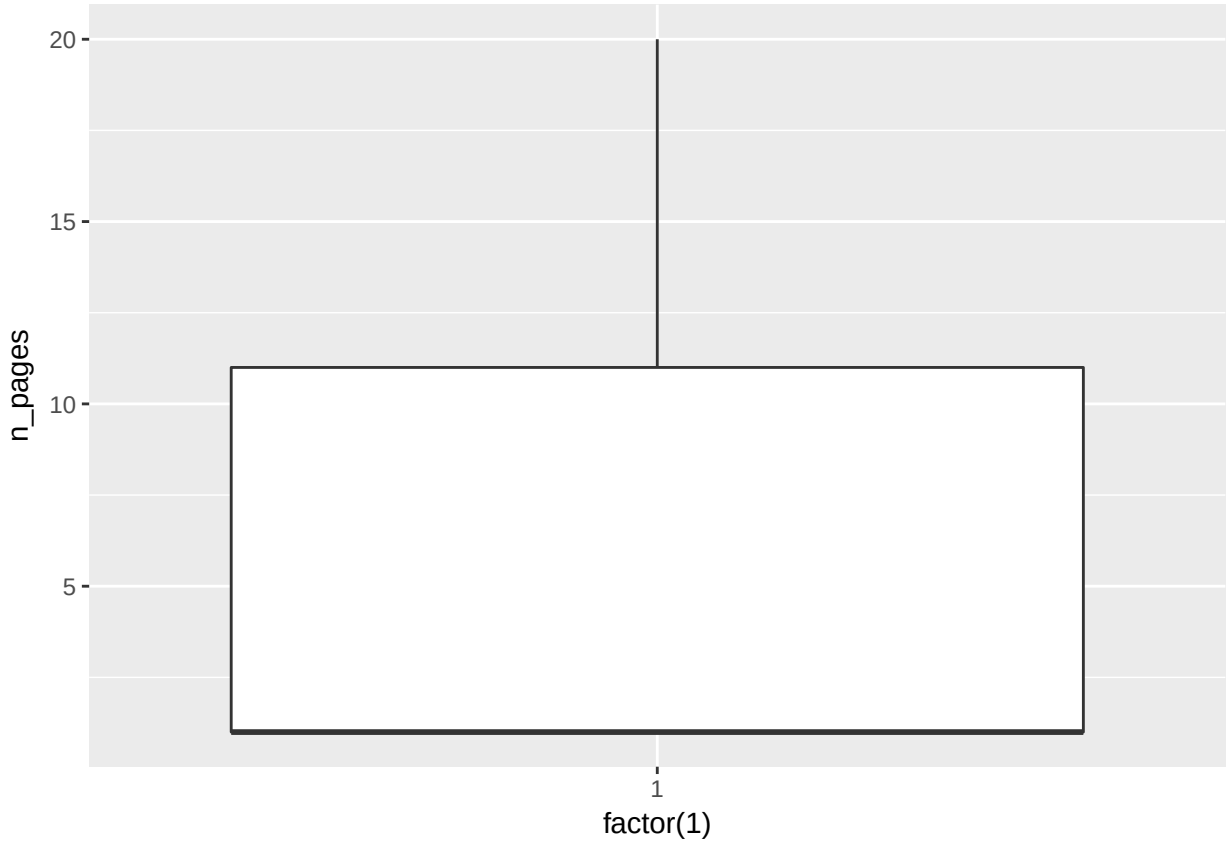
`geom_boxplot()` is used to create box plots.

```
ggplot(ecom, aes(x = factor(device), y = n_pages)) +
  geom_boxplot()
```



We must specify both the `x` and `y` aesthetic. If you are not comparing the distribution across groups, use the below method to generate the box plot.

```
ggplot(ecom, aes(x = factor(1), y = n_pages)) +
  geom_boxplot()
```



Histogram

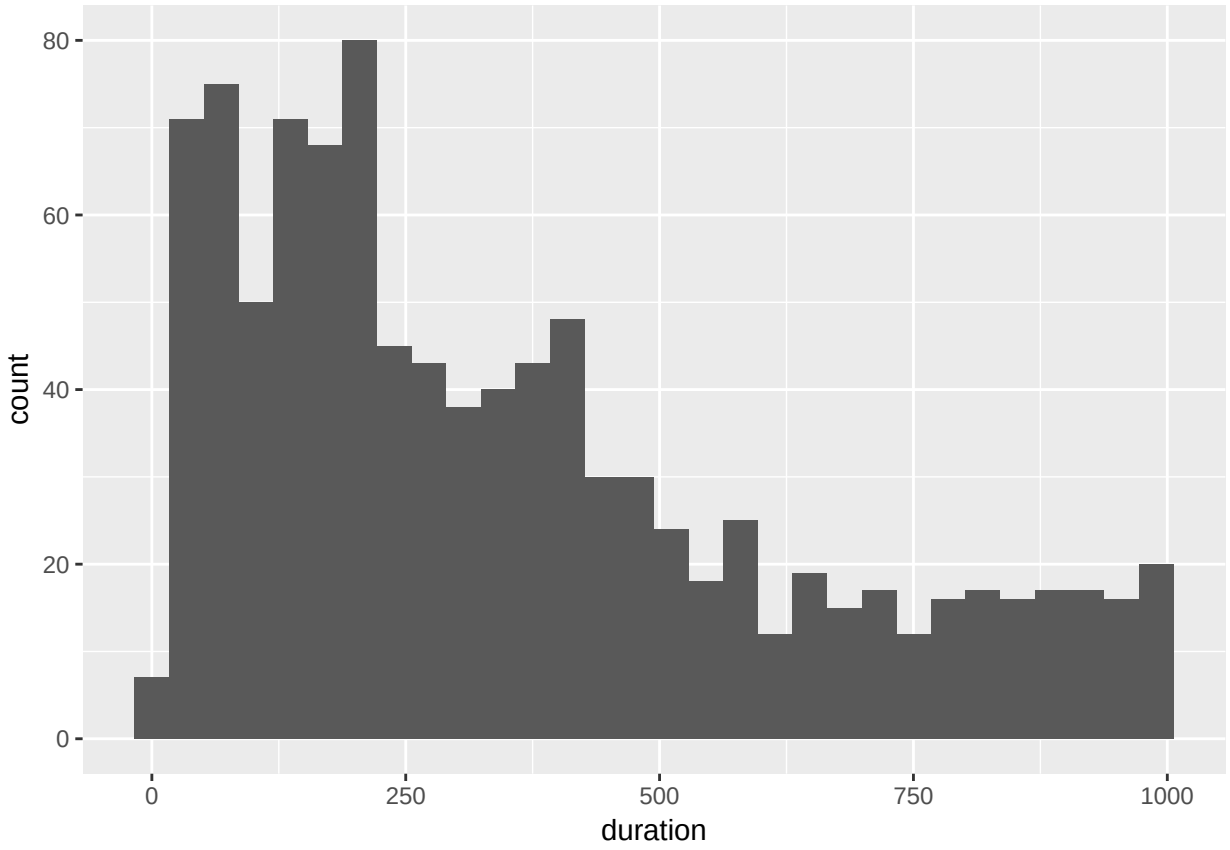
Histograms are used to examine:

- distribution of a continuous variable
- skewness and kurtosis

We can create a histogram using `geom_histogram()`. Only the `x` aesthetic needs to be supplied.

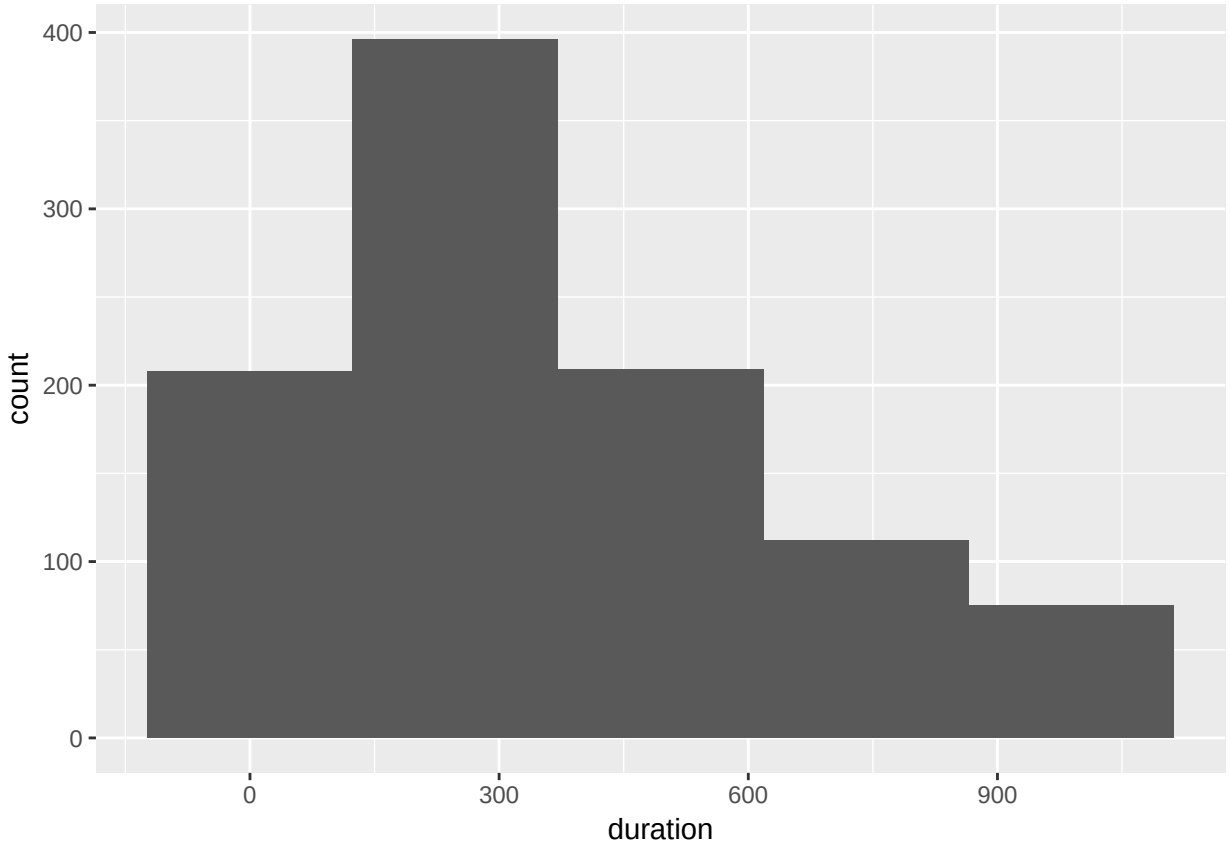
```
ggplot(ecom, aes(x = duration)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The default number of bins used is 30 which may not be helpful always. Use `bins` argument to specify the appropriate number of bins for the histogram.

```
ggplot(ecom, aes(x = duration)) +
  geom_histogram(bins = 5)
```



Line

Line charts are used to examine trends over time.

Data

```
gdp <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/gdp.csv')
```

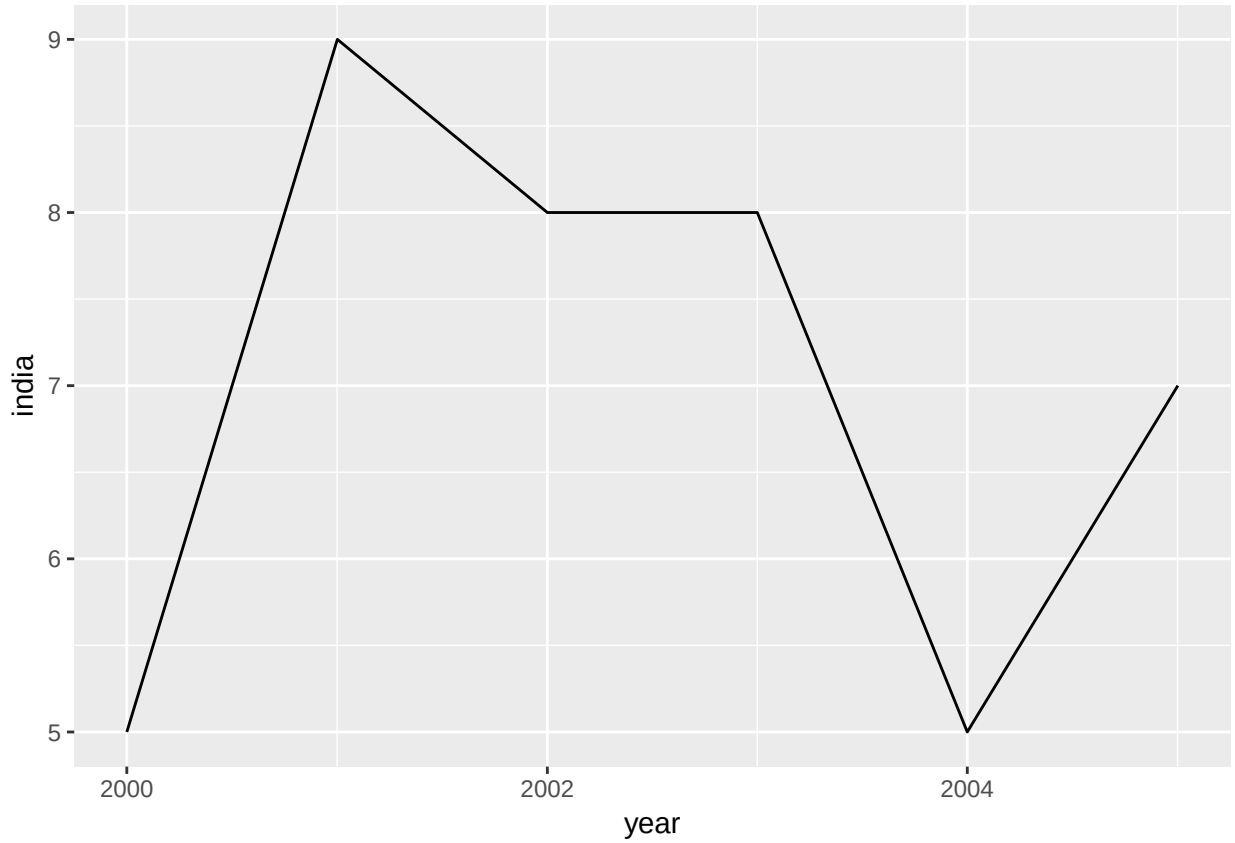
```
## Warning: Missing column names filled in: 'X1' [1]
```

```
gdp
```

```
## # A tibble: 6 x 6
##      X1     X year       growth india china
##   <int> <int> <date>      <int> <int> <int>
## 1     1     1 2000-01-01      6     5     8
## 2     2     2 2001-01-01      9     9     5
## 3     3     3 2002-01-01      8     8     6
## 4     4     4 2003-01-01      9     8     8
## 5     5     5 2004-01-01      9     5     9
## 6     6     6 2005-01-01      8     7     8
```

Use `geom_line()` to create line plots.

```
ggplot(gdp, aes(year, india)) +
  geom_line()
```

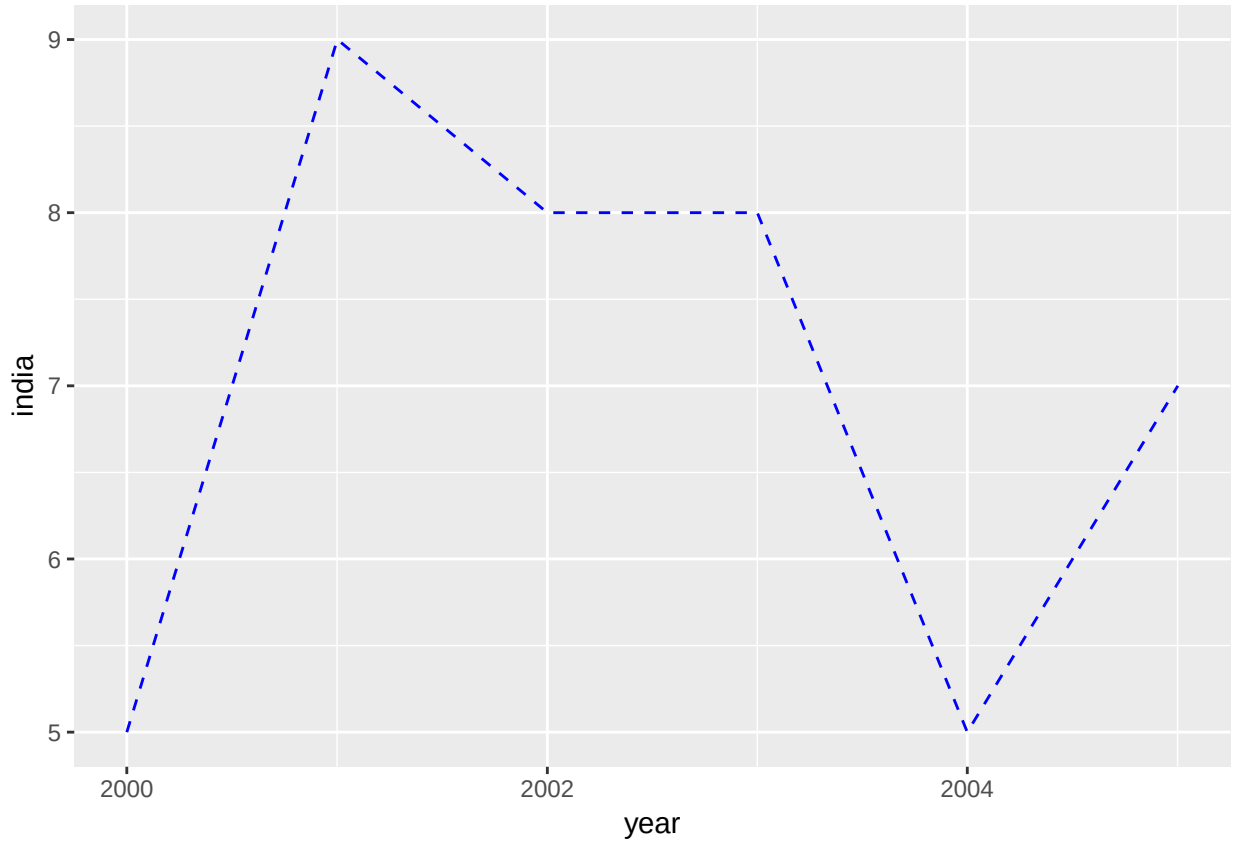


The following can be modified to improve the appearance of the line:

- `color`
- `size`
- `linetype`

In the below example, we modify the color and type of the line.

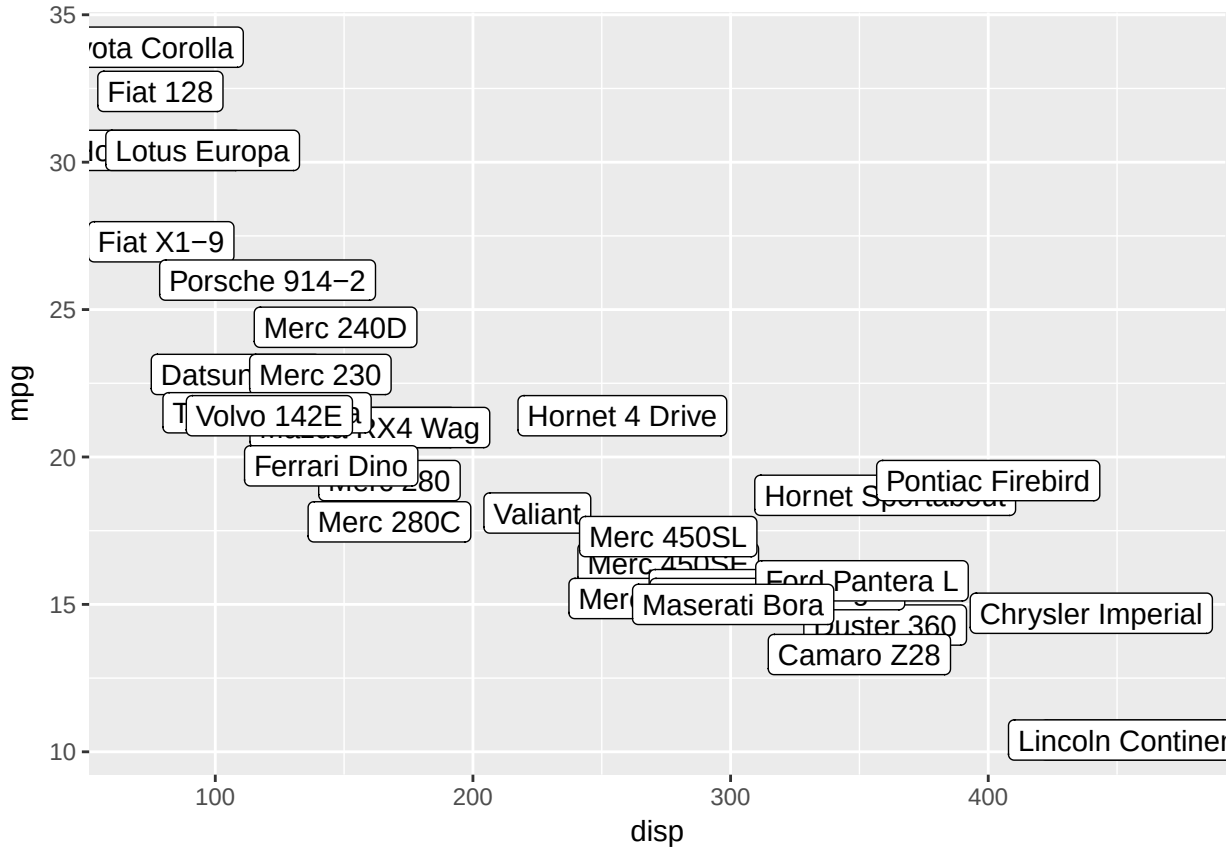
```
ggplot(gdp, aes(year, india)) +
  geom_line(color = 'blue', linetype = 'dashed')
```



Label

Labels can be added to identify data points using `geom_label()`.

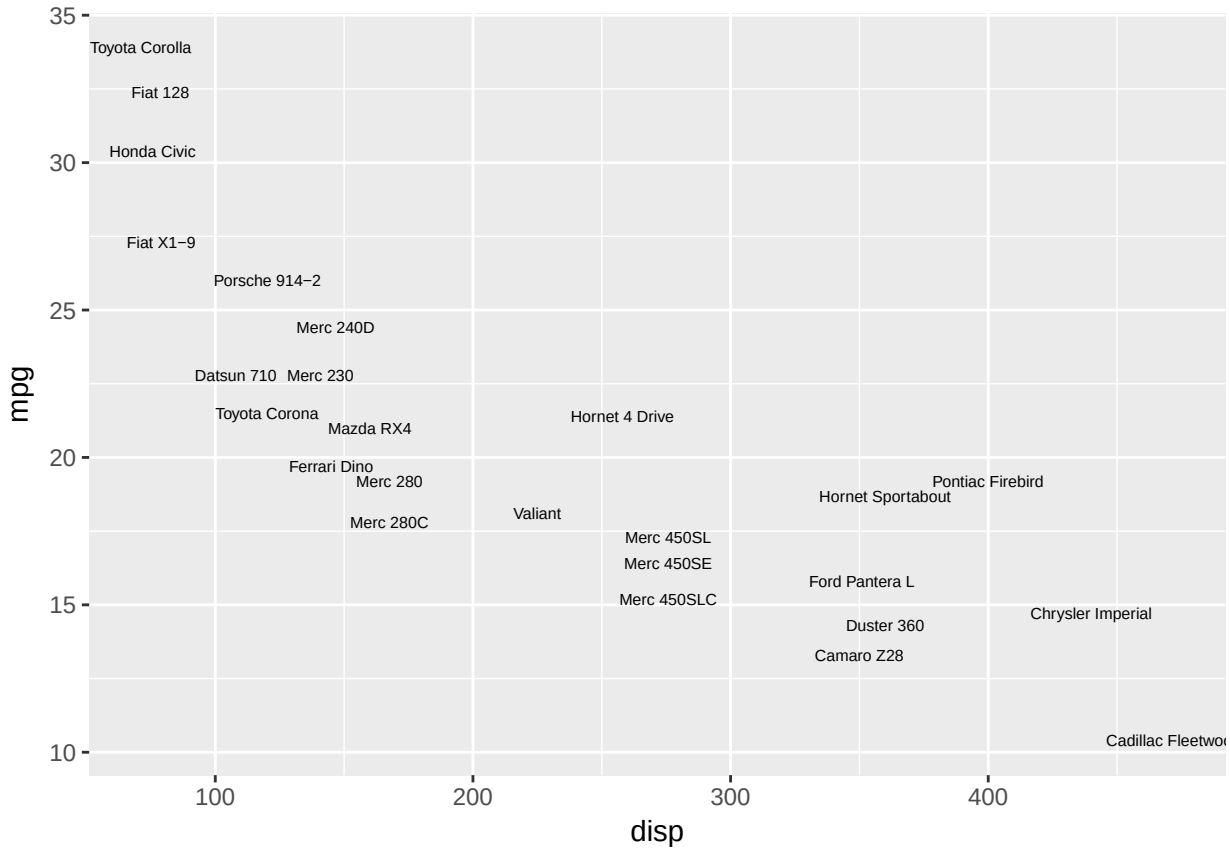
```
ggplot(mtcars, aes(disp, mpg, label = rownames(mtcars))) +
  geom_label()
```



Text

Use `geom_text()` to add text to the plot. We can modify the size of the text and ensure that they do not overlap using `check_overlap` argument.

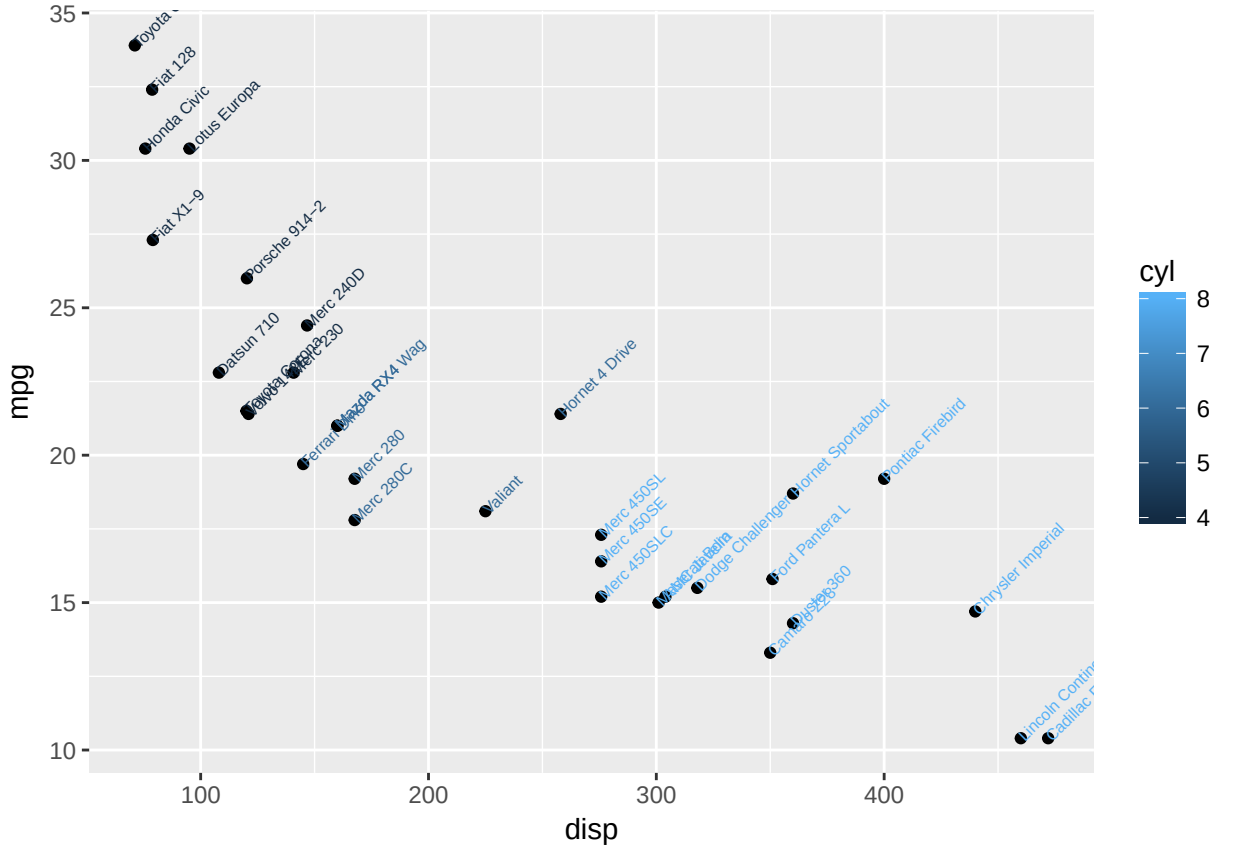
```
ggplot(mtcars, aes(disp, mpg, label = rownames(mtcars))) +
  geom_text(check_overlap = TRUE, size = 2)
```



In the below example, we:

- map the color of the text to `cyl` variable
- nudge the text to avoid overlapping with the points
- and change the horizontal justification, size and angle of text

```
ggplot(mtcars, aes(x = disp, y = mpg, label = rownames(mtcars))) +
  geom_point() +
  geom_text(aes(color = cyl), hjust = 0, nudge_x = 0.05,
            size = 2, angle = 45)
```



Summary

In this chapter, we learnt to build different types of plots using `geom_*` instead of `qplot()`.

Up Next..

In the next chapter, we will learn about aesthetics.

Data Visulaization - Title & Axis Labels

Introduction

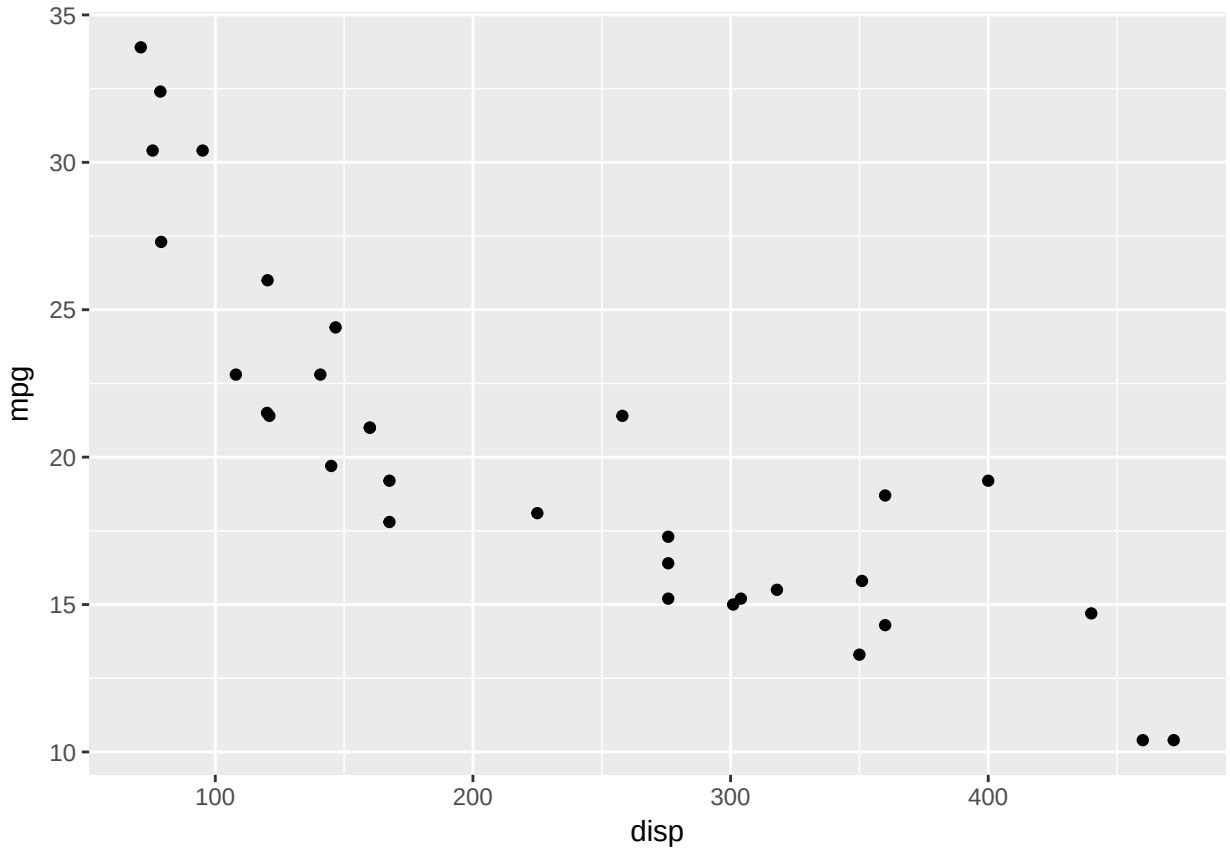
In the previous chapter, we learnt about aesthetics. In this chapter, we will learn to:

- add title and subtitle to the plot
- modify axis labels
- modify axis range

Title & Subtitle

Let us create a simple scatter plot with which we will work in the rest of the chapter.

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg))
```



There are two ways to add title to a plot:

- `ggtitle()`
- `labs()`

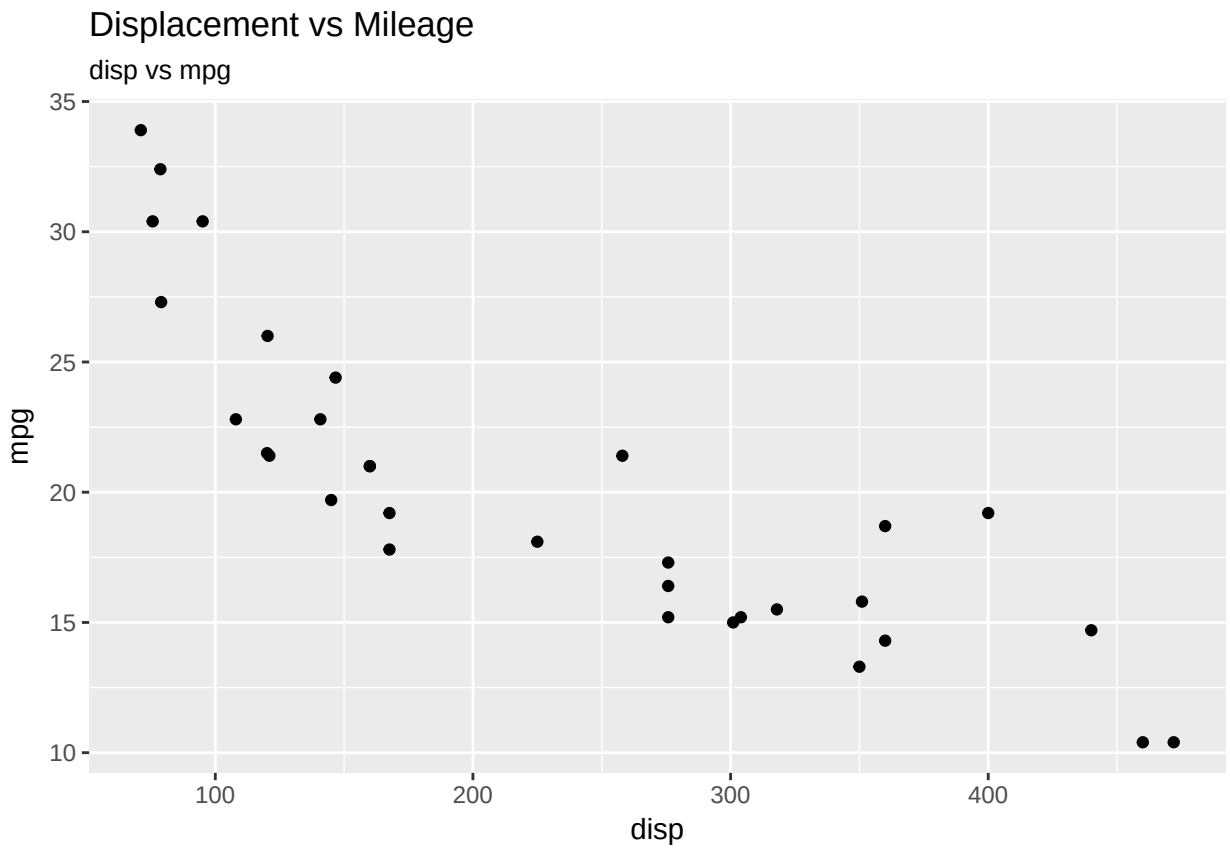
ggtitle()

Let us explore the `ggtitle()` function first. It takes two arguments:

- label: title of the plot
- subtitle: subtitle of the plot

Title & Subtitle

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  ggtitle(label = 'Displacement vs Mileage', subtitle = 'disp vs mpg')
```



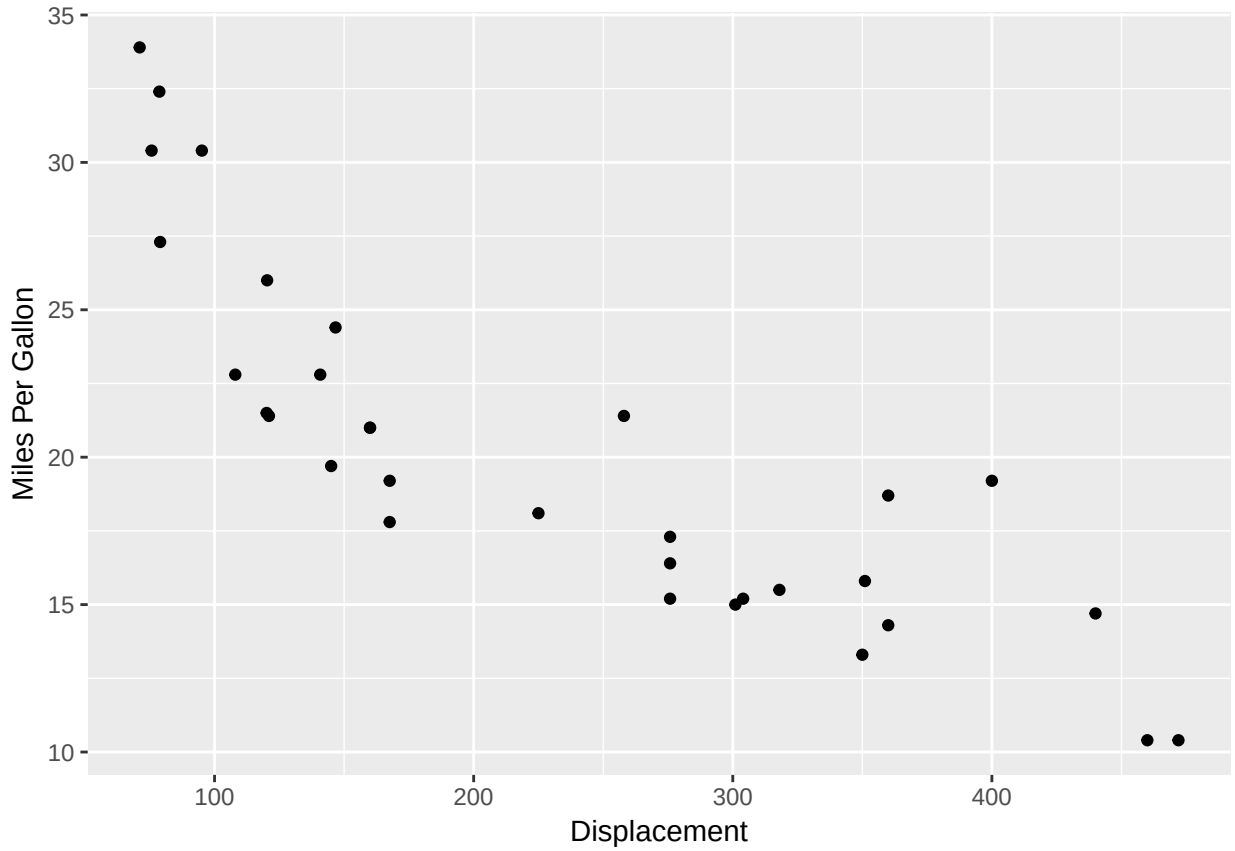
Axis Labels

You can add labels to the axis using:

- `xlab()`
- `ylab()`
- `labs()`

Let us add labels using `xlab()` for the X axis and `ylab()` for the Y axis.

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  xlab('Displacement') + ylab('Miles Per Gallon')
```



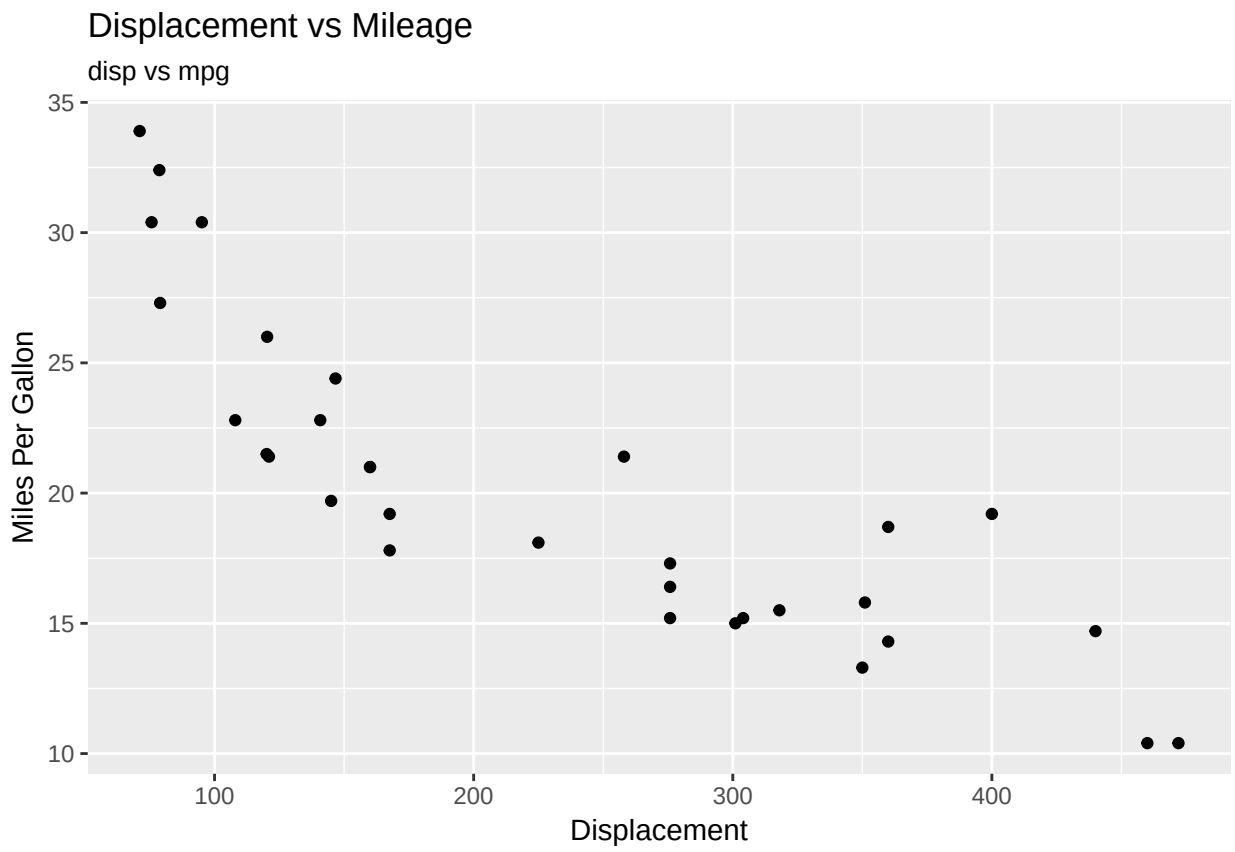
Labs

The `labs()` function can be used to add the following to a plot:

- title
- subtitle
- X axis label
- Y axis label

Let us add title and axis labels using `labs()`.

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  labs(title = 'Displacement vs Mileage', subtitle = 'disp vs mpg',
    x = 'Displacement', y = 'Miles Per Gallon')
```



Axis Range

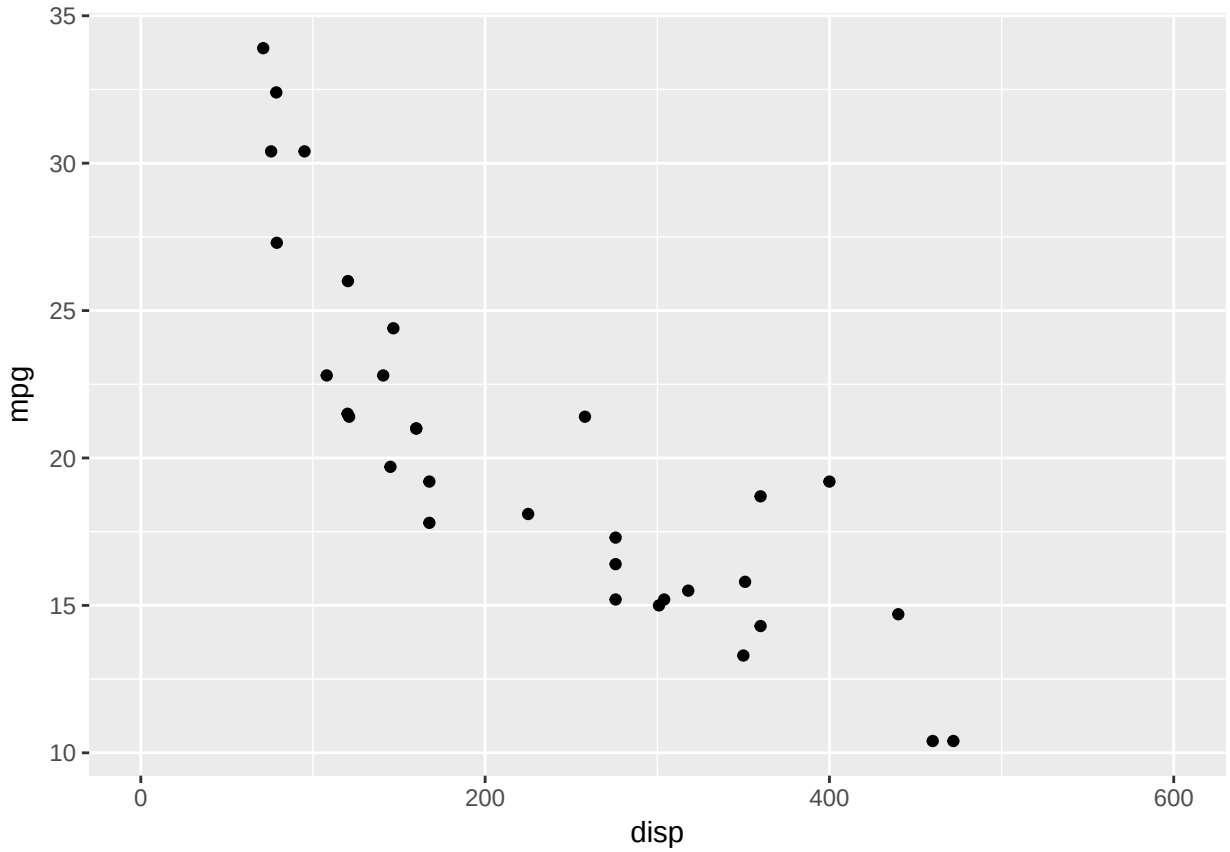
In certain scenarios, you may want to modify the range of the axis. In ggplot2, we can achieve this using:

- `xlim()`
- `ylim()`
- `expand_limits()`

`xlim()` and `ylim()` take a numeric vector of length 2 as input. `expand_limits()` takes two numeric vectors (each of length 2), one for each axis. In all of the above functions, the first element represents the lower limit and the second element represents the upper limit.

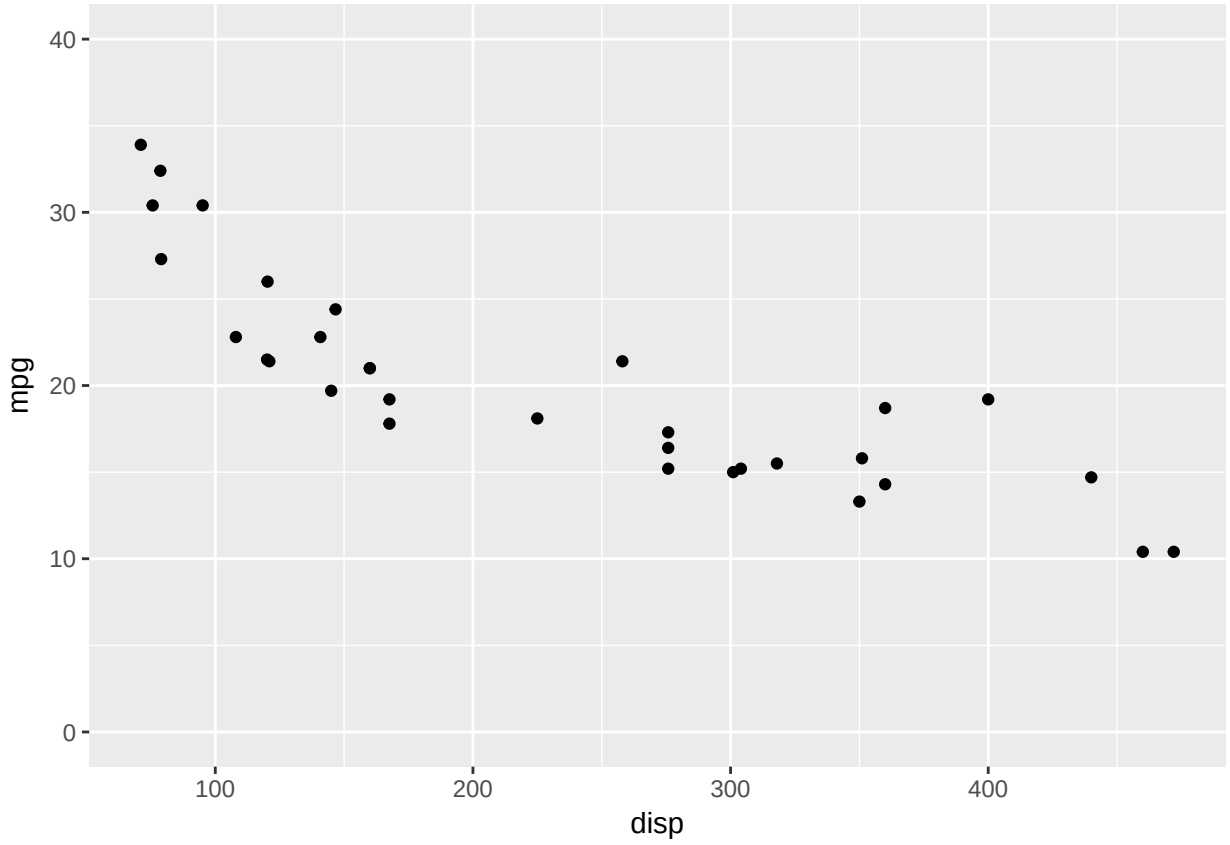
Let us modify the range of the X axis using `xlim()`. The lower limit will be 0 and the upper limit 600.

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  xlim(c(0, 600))
```



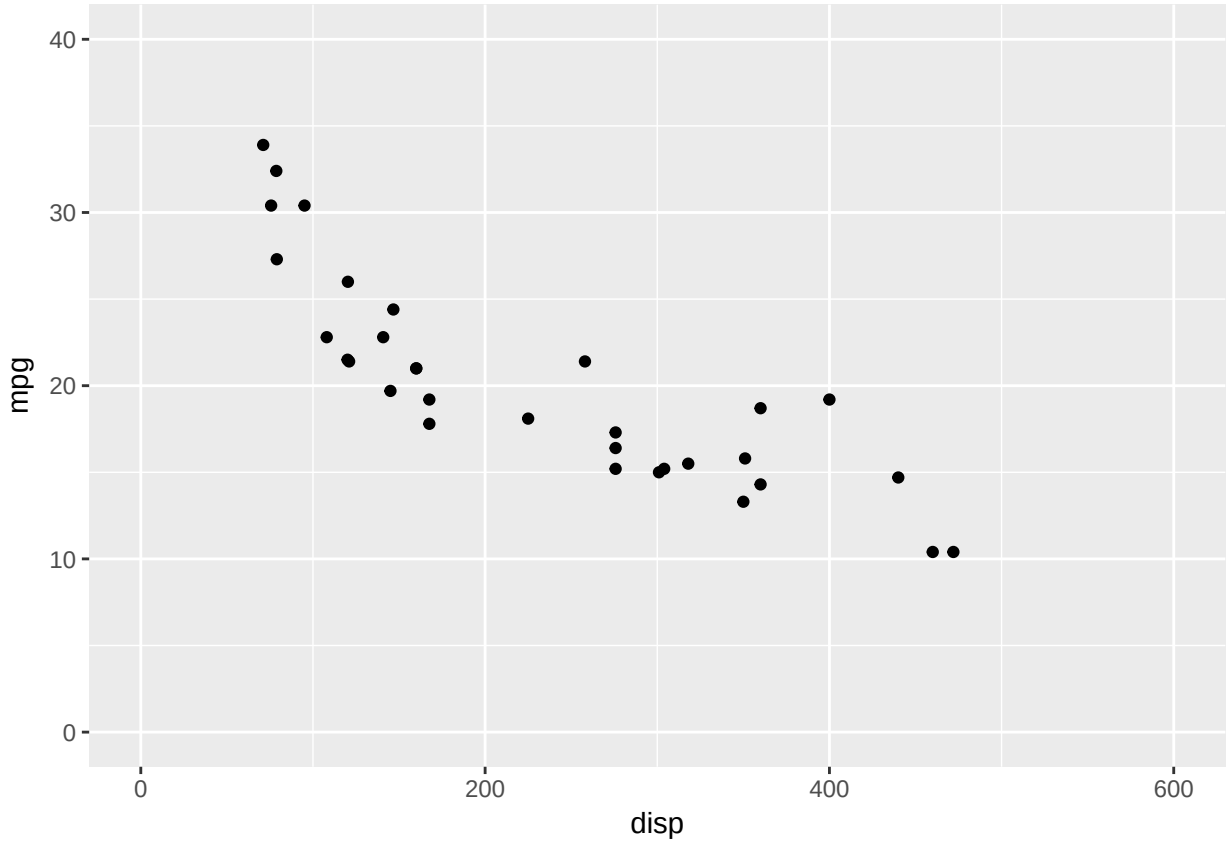
The range of the Y axis will be modified using `ylim()`. The lower limit will be 0 and the upper limit will be 40.

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  ylim(c(0, 40))
```



Instead of using `xlim()` and `ylim()`, we can use `expand_limits()` to modify the range of both the X and Y axis.

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  expand_limits(x = c(0, 600), y = c(0, 40))
```



Data Visulaization - Scatter Plots

Introduction

In the previous chapter, we learnt about text annotations. In this chapter, we will build scatter plots by applying everything we have learnt so far.

- build scatter plots
- modify point
  - color
  - fill
  - alpha
  - shape
  - size
- fit regression line

Libraries, Code & Data

We will use the following libraries in this chapter:

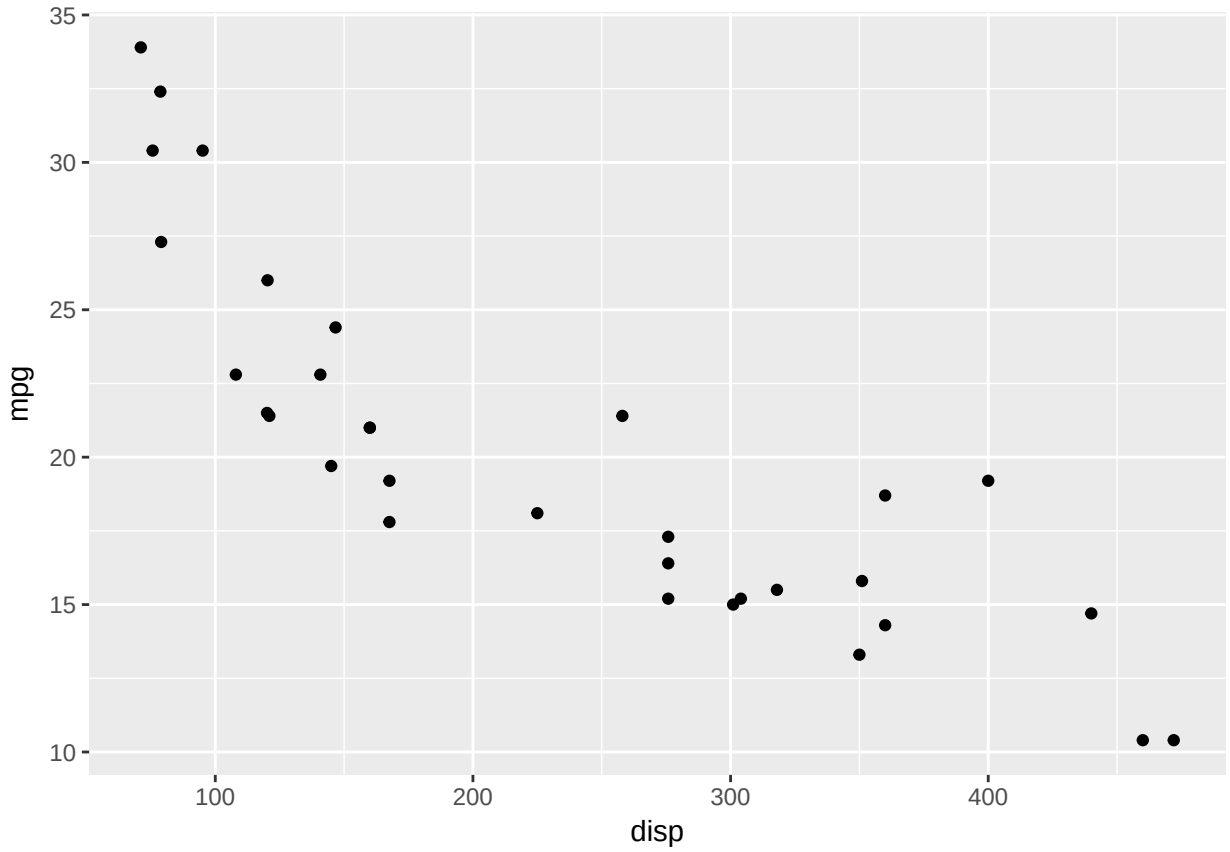
- readr
- ggplot2

All the data sets used in this chapter can be found here and code can be downloaded from here.

Basic Plot

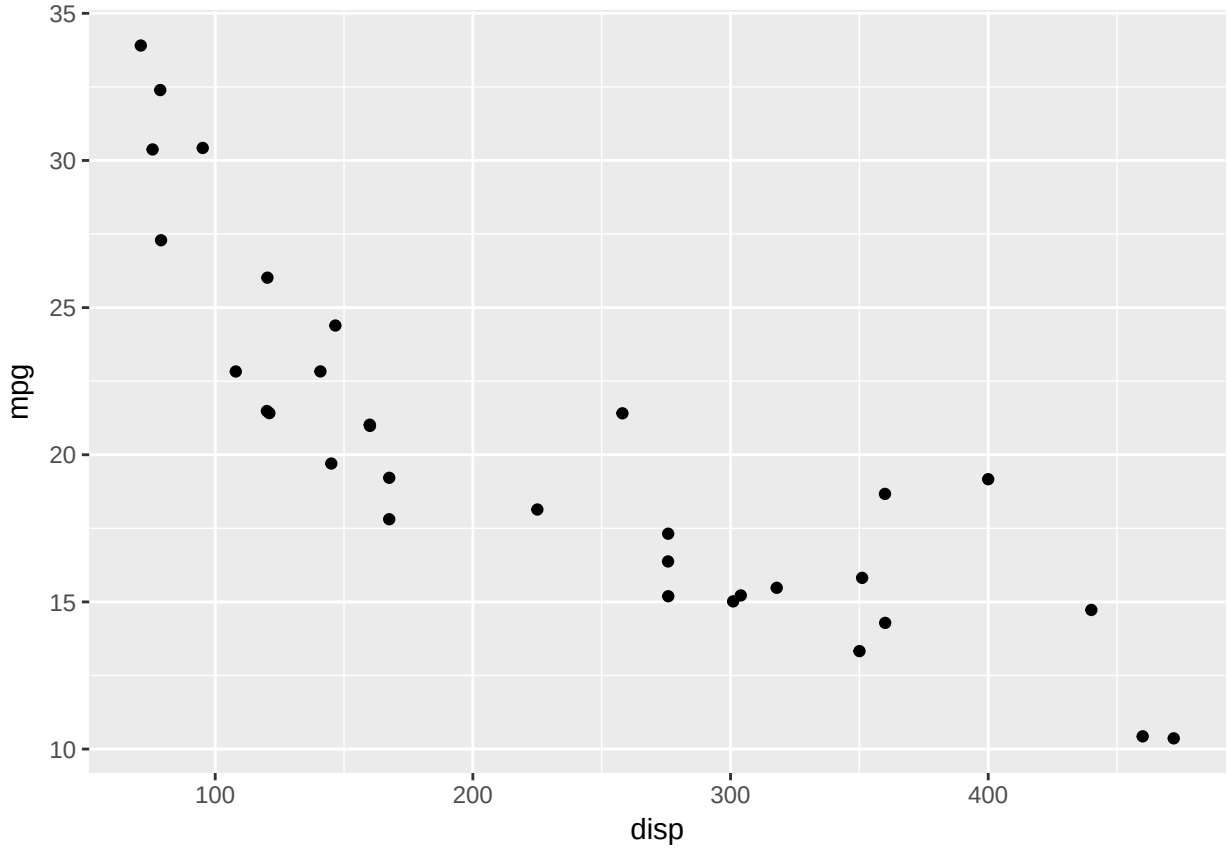
Let us build a simple scatter plot using the `mtcars` data to examine the relationship between `disp` (diaplacement) and `mpg` (miles per gallon).

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg))
```



Jitter

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg), position = 'jitter')
```



Aesthetics

Now let us modify the aesthetics of the points. There are two ways:

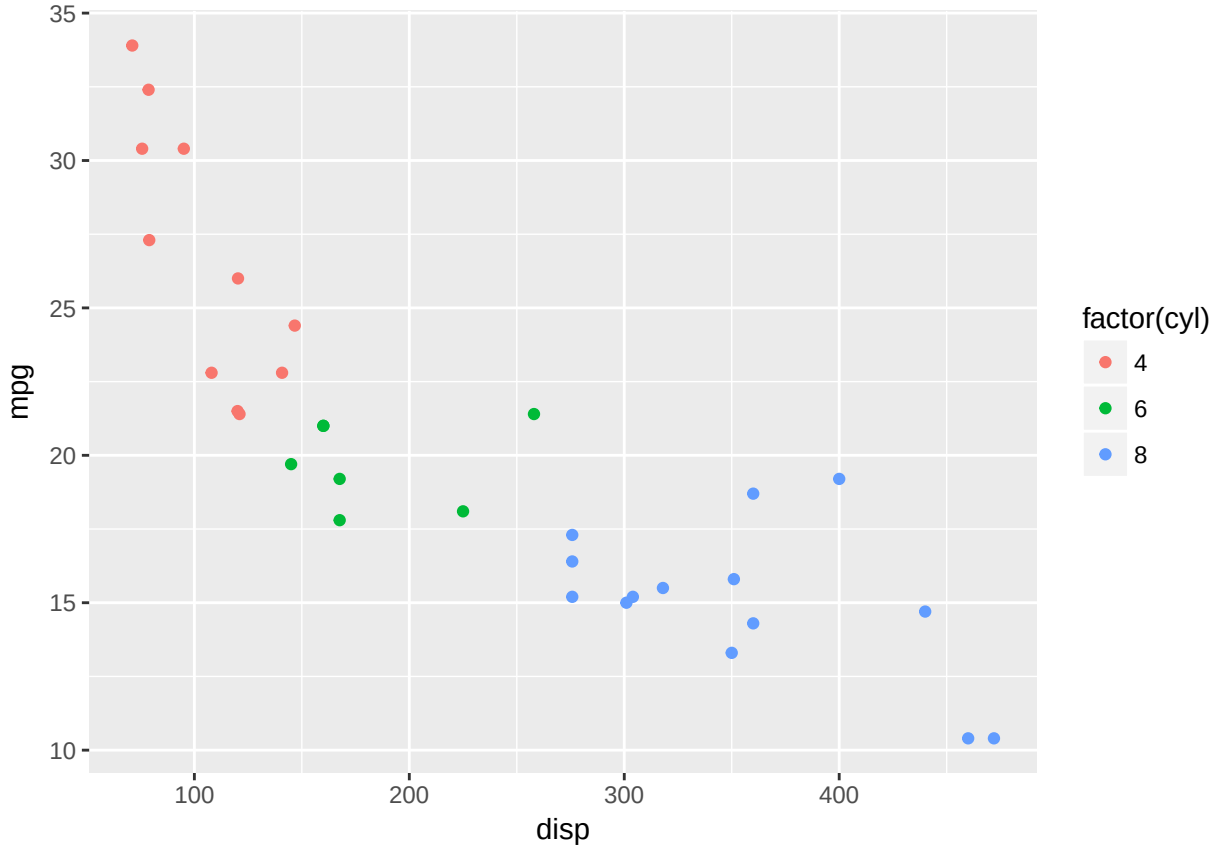
- map them to variables using the `aes()` function
- or specify values

In the next 4 examples, we will

- map color to a categorical variable
- map color to a continuous variable
- specify value for color
- specify value for color opacity

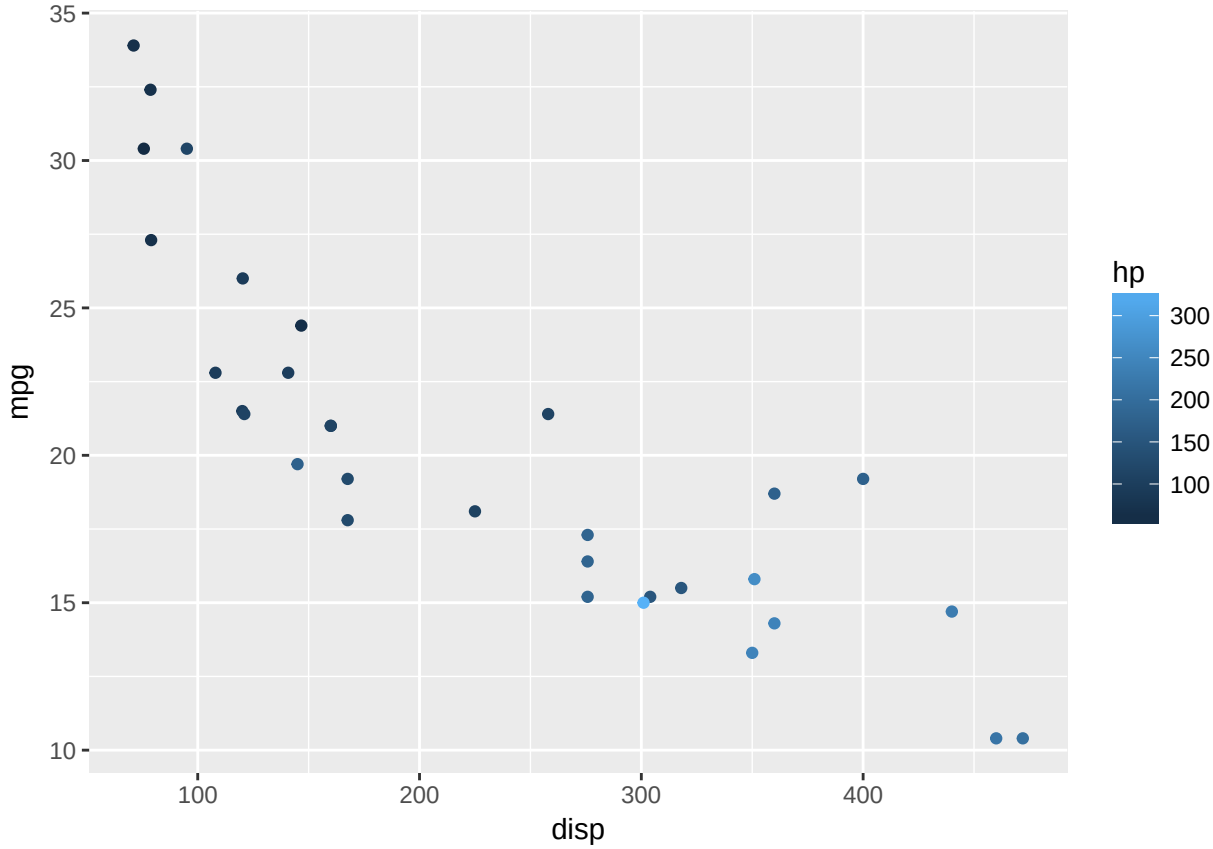
Map Color to Variable (Categorical)

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg, color = factor(cyl)))
```



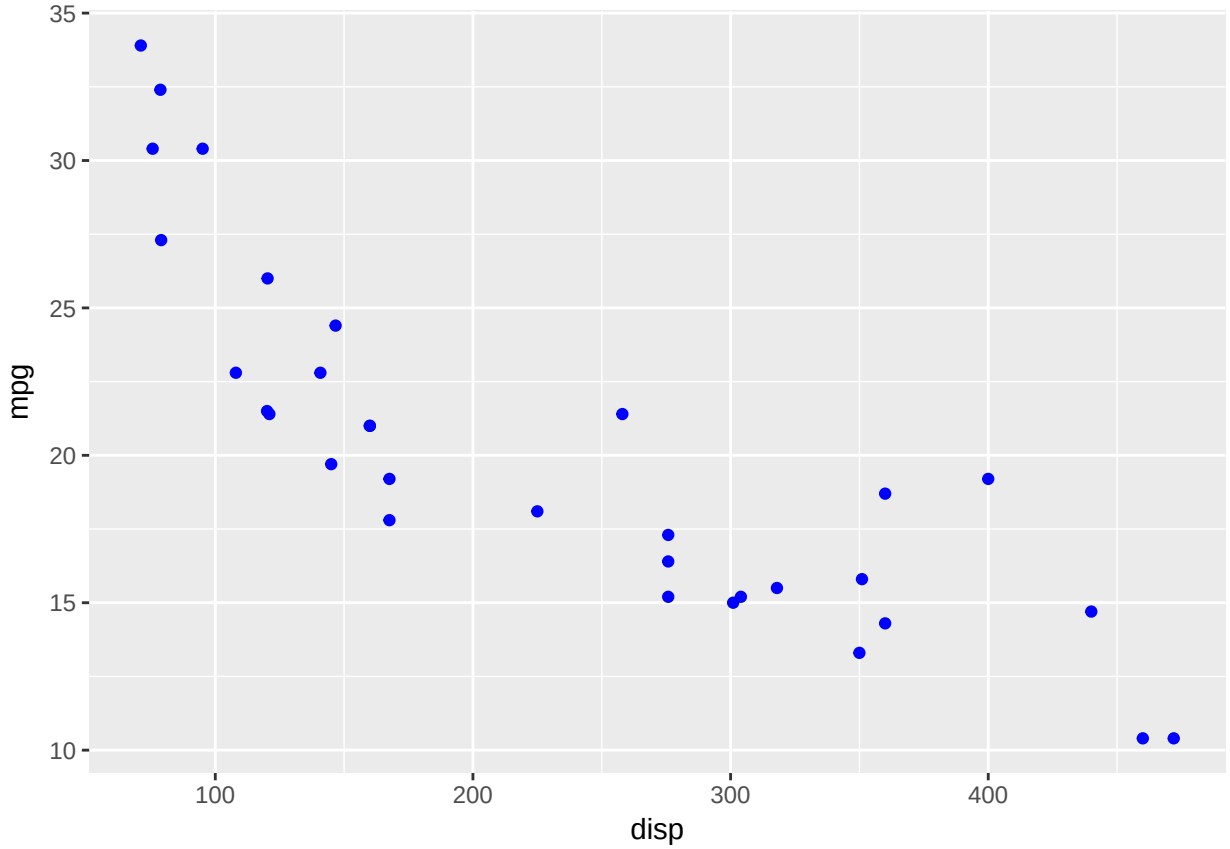
Map Color to Variable (Continuous)

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg, color = hp))
```



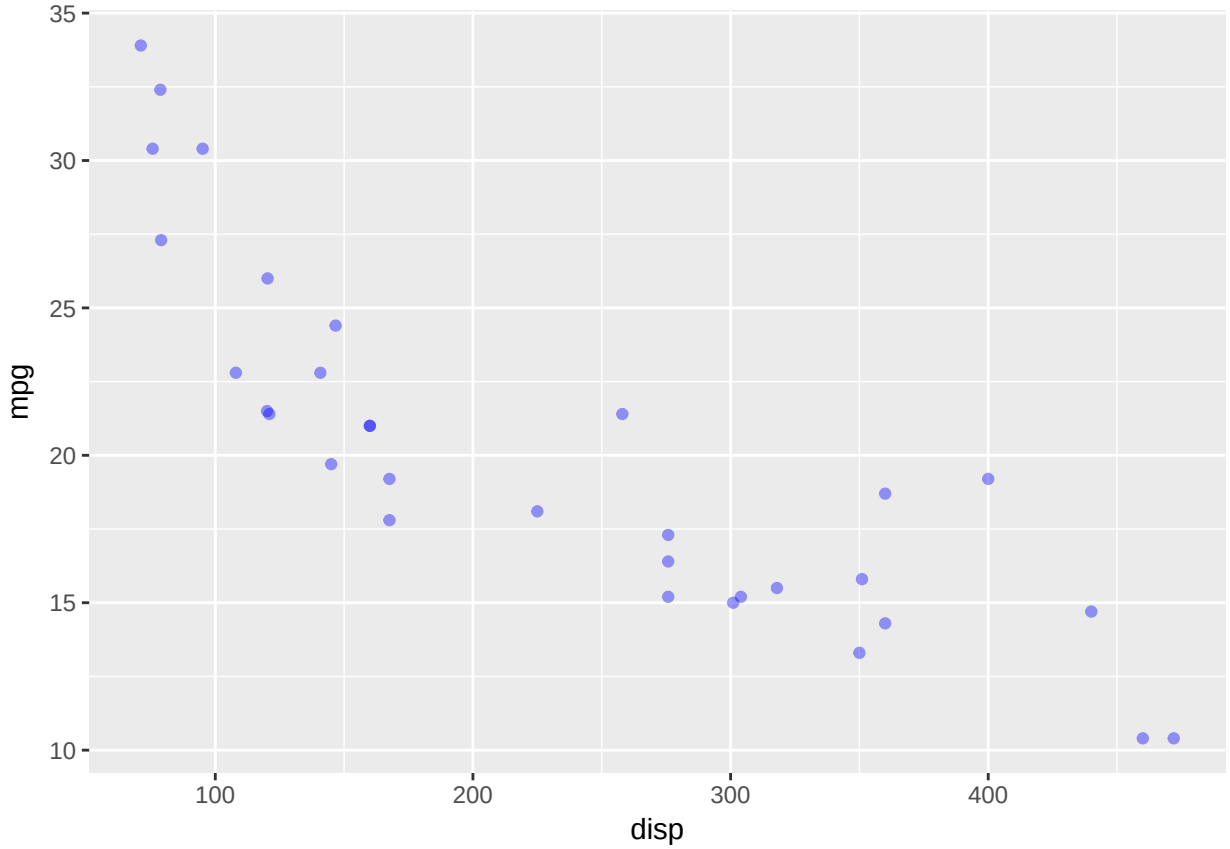
Specify Value for Color

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg), color = 'blue')
```



Specify Value for Alpha

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg), color = 'blue', alpha = 0.4)
```

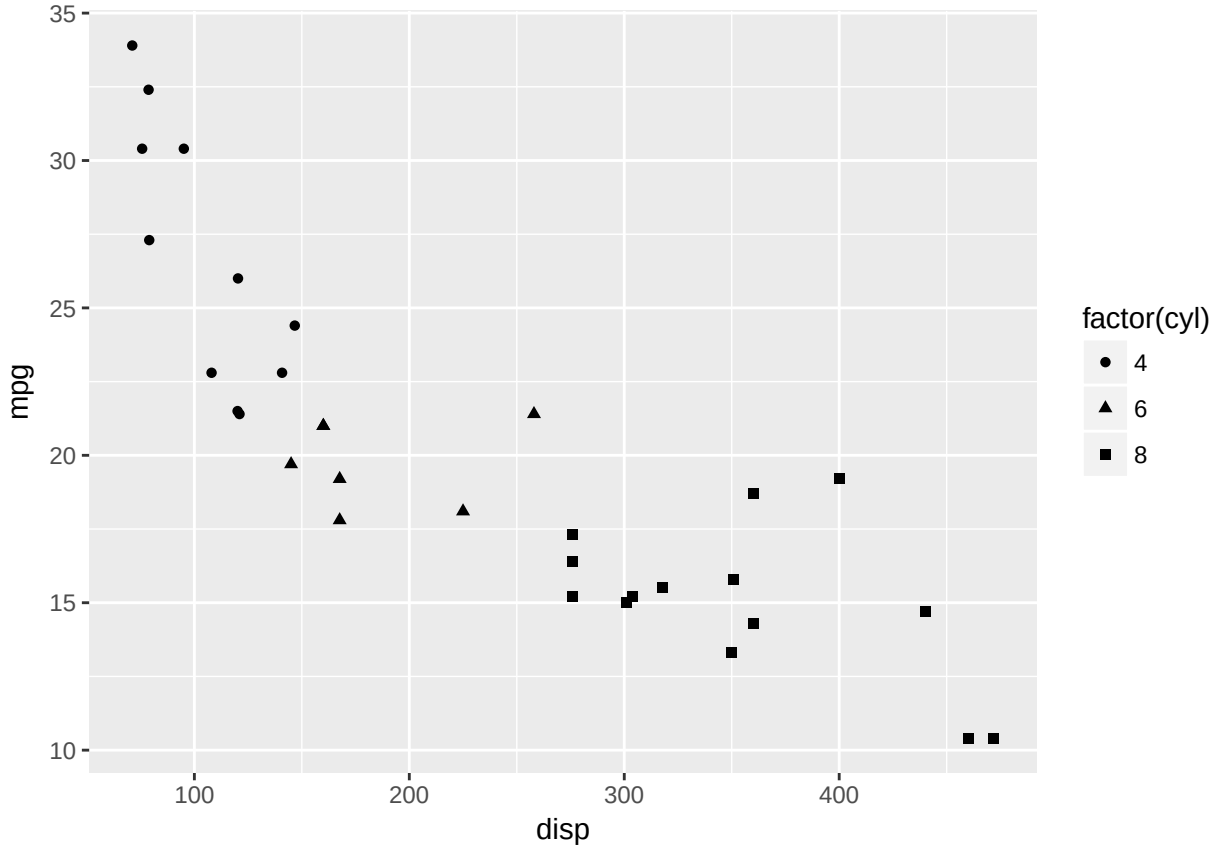


In the next 2 examples, we will

- map shape to a variable
- specify value for shape

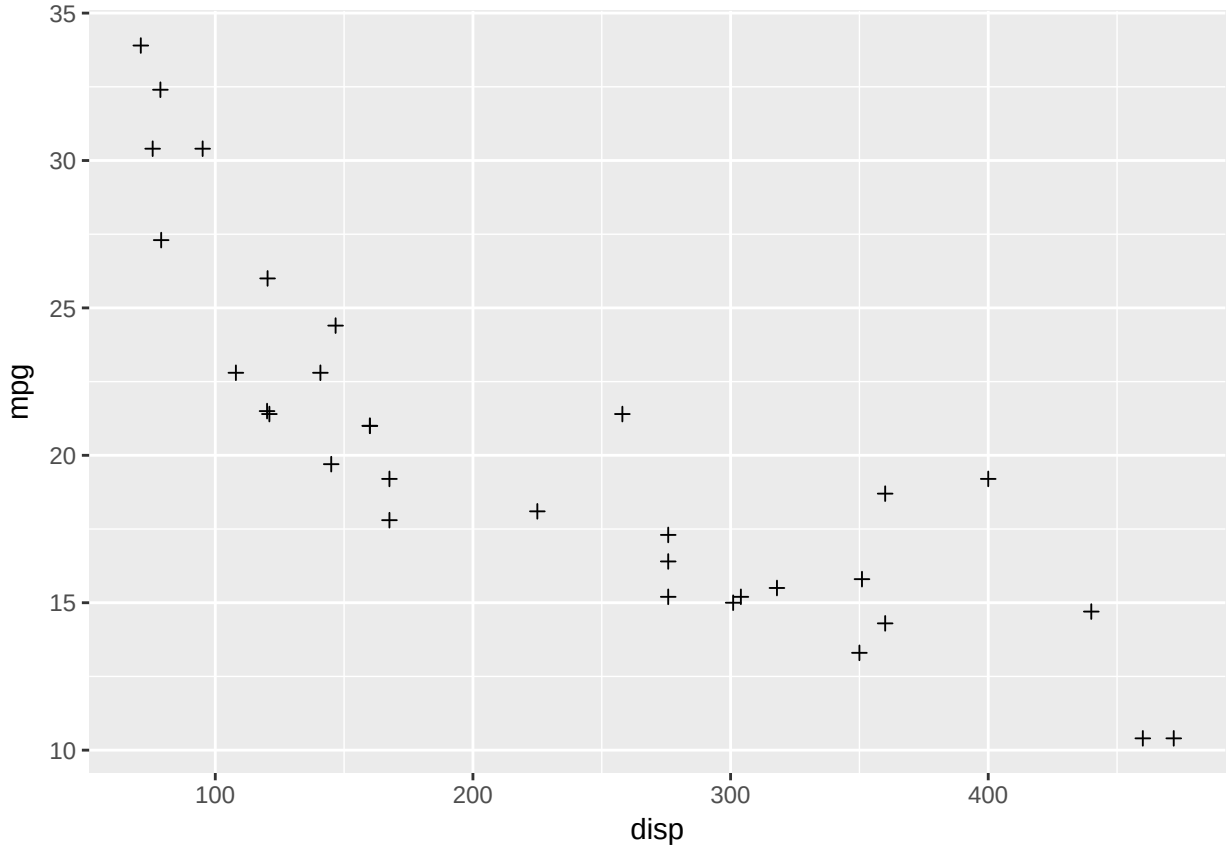
Map Shape to Variable

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg, shape = factor(cyl)))
```



Specify Value for Shape

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg), shape = 3)
```

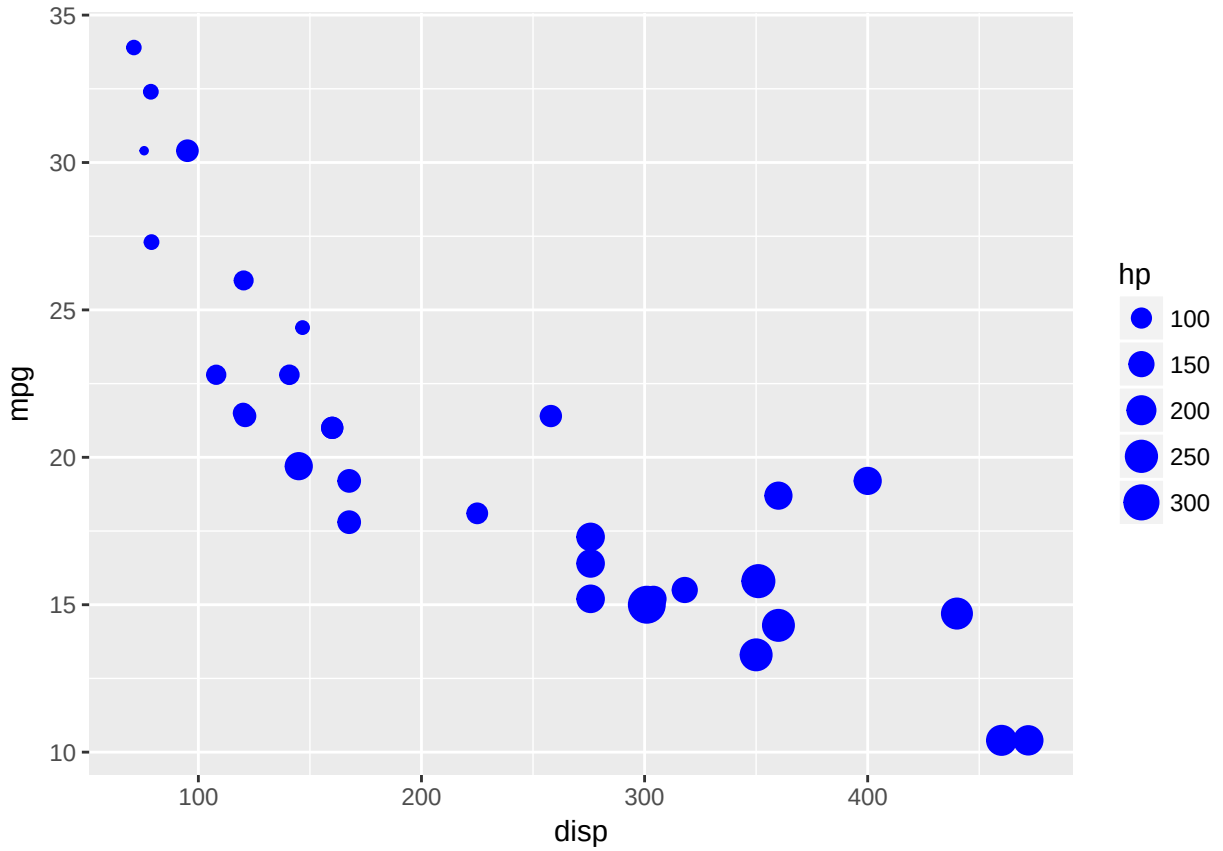


In the next 2 examples, we will

- map size to a variable
- specify value for size

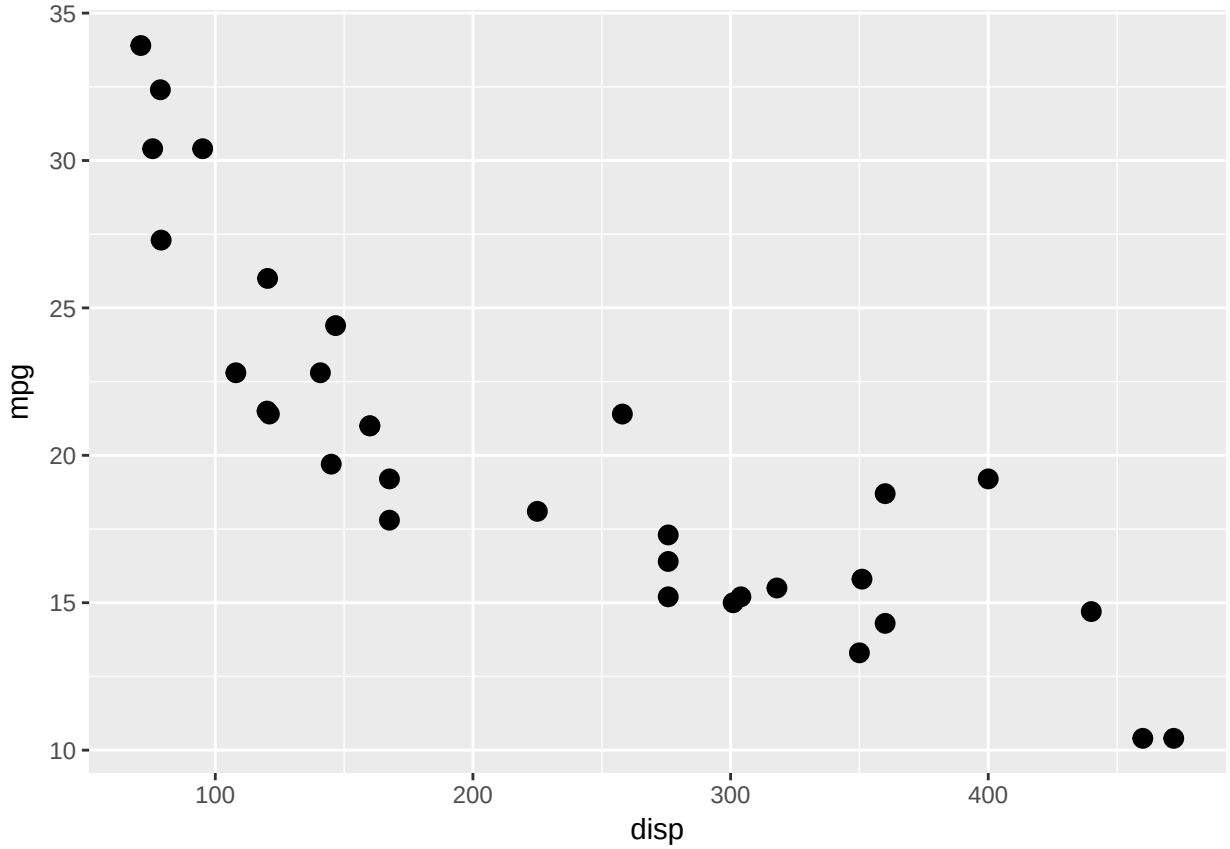
Map Size to Variable

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg, size = hp), color = 'blue')
```



Specify Value for Size

```
ggplot(mtcars) +
  geom_point(aes(disp, mpg), size = 3)
```



Regression Line

Most often, after building a scatter plot to examine the relationship between two variables, we fit a regression line. In this section, we will learn to fit a line to the scatter plot using:

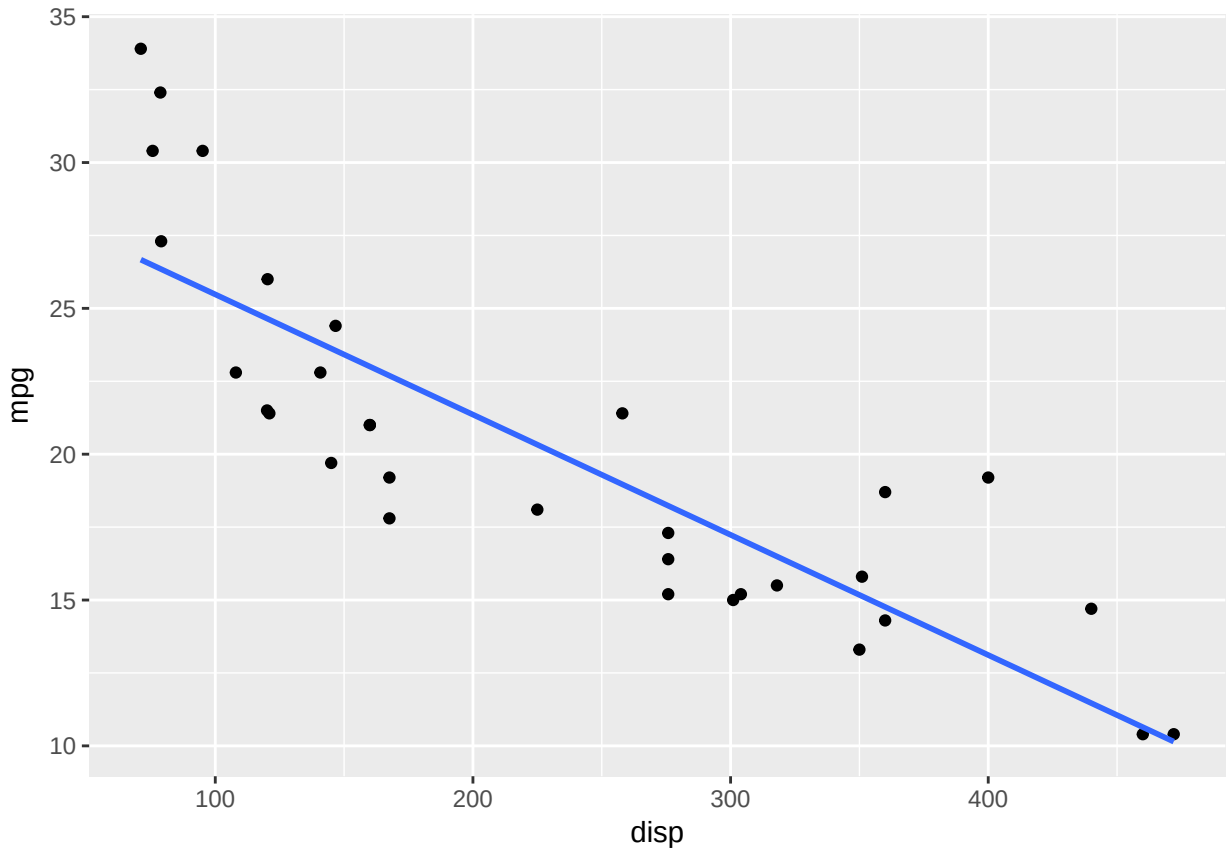
- `geom_smooth()`
- `geom_abline()`

In the below example, we fit a regression line using `geom_smooth()`. It takes two arguments:

- `method`
- `se`

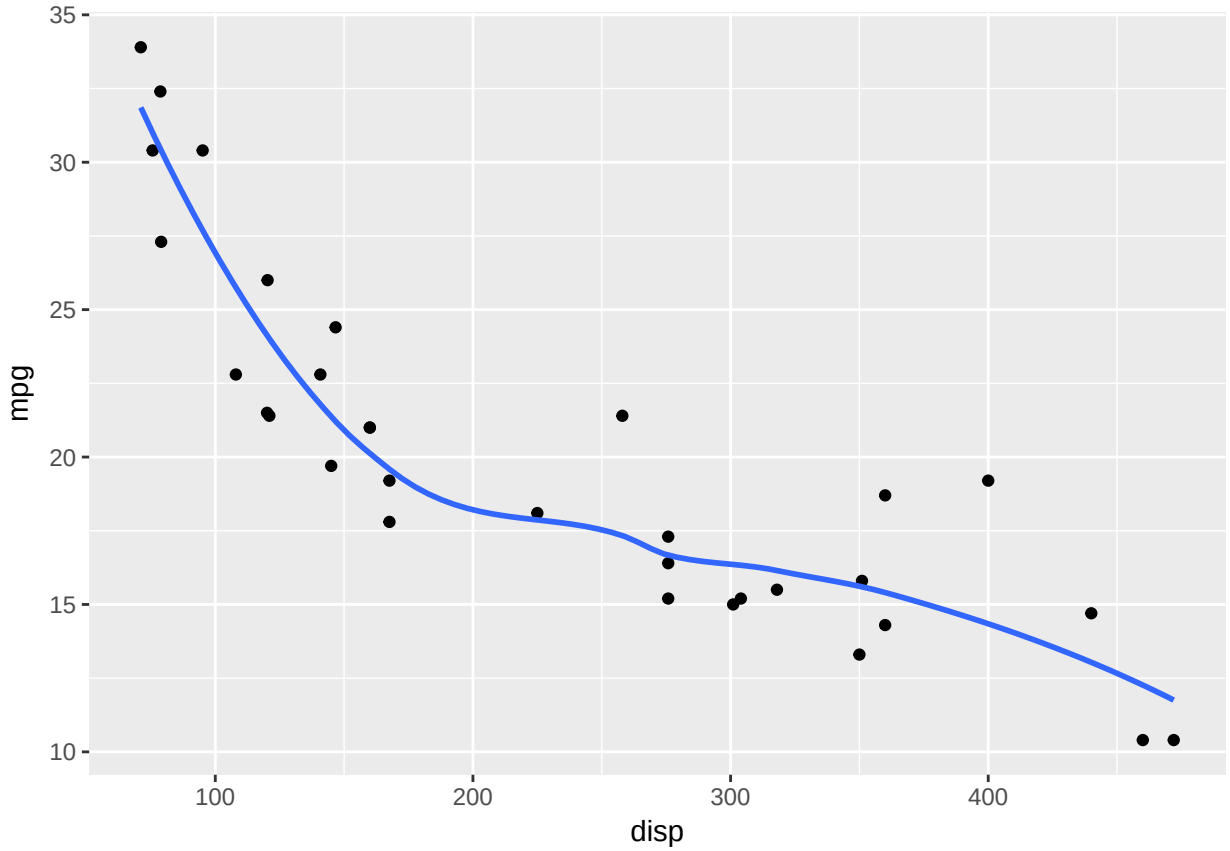
The `method` argument is used to specify the model type i.e. `lm` or `loess`.

```
ggplot(mtcars, aes(disp, mpg)) +
  geom_point() +
  geom_smooth(method = 'lm', se = FALSE)
```



Regression Line - Loess Method

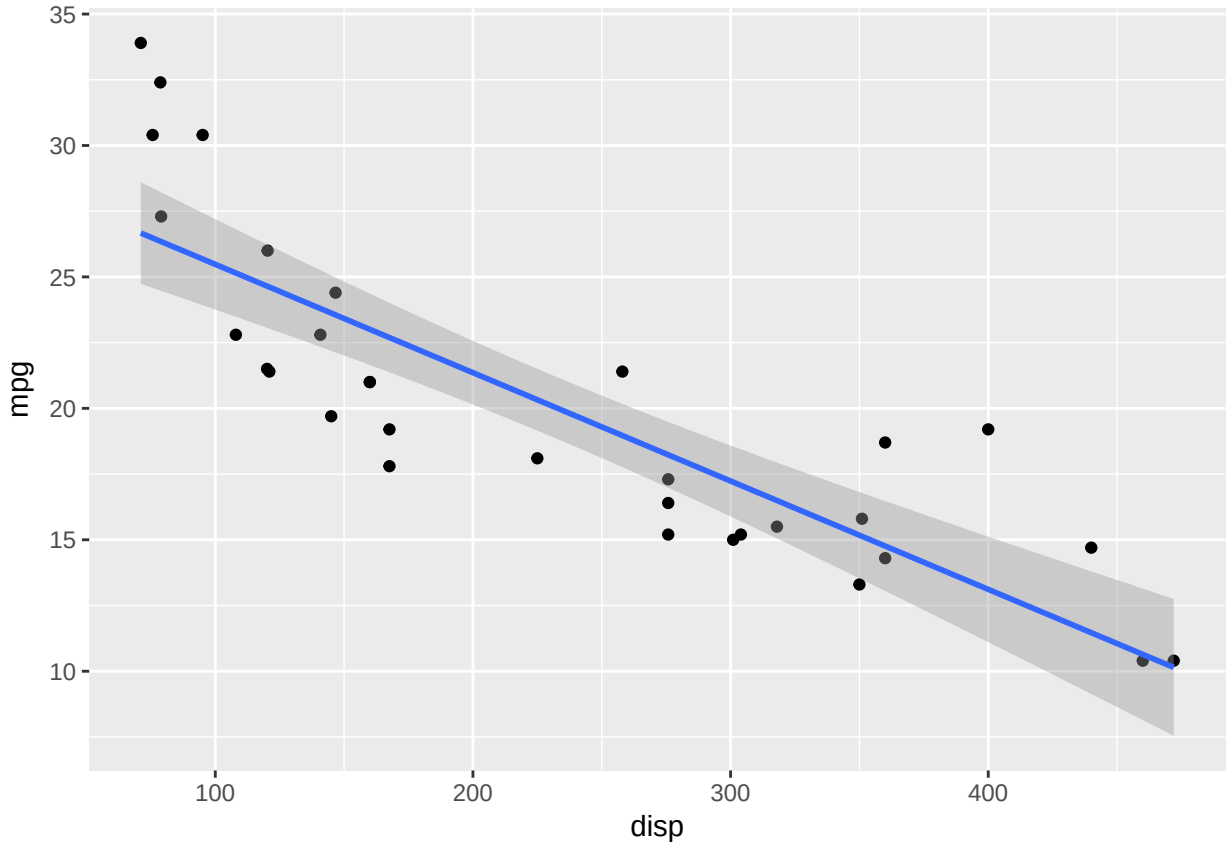
```
ggplot(mtcars, aes(disp, mpg)) +
  geom_point() +
  geom_smooth(method = 'loess', se = FALSE)
```



Regression Line - Conf. Interval

The `se` argument takes logical values i.e. `TRUE` or `FALSE`. If set to `TRUE`, the confidence band for the regression line is drawn.

```
ggplot(mtcars, aes(disp, mpg)) +
  geom_point() +
  geom_smooth(method = 'lm', se = TRUE)
```



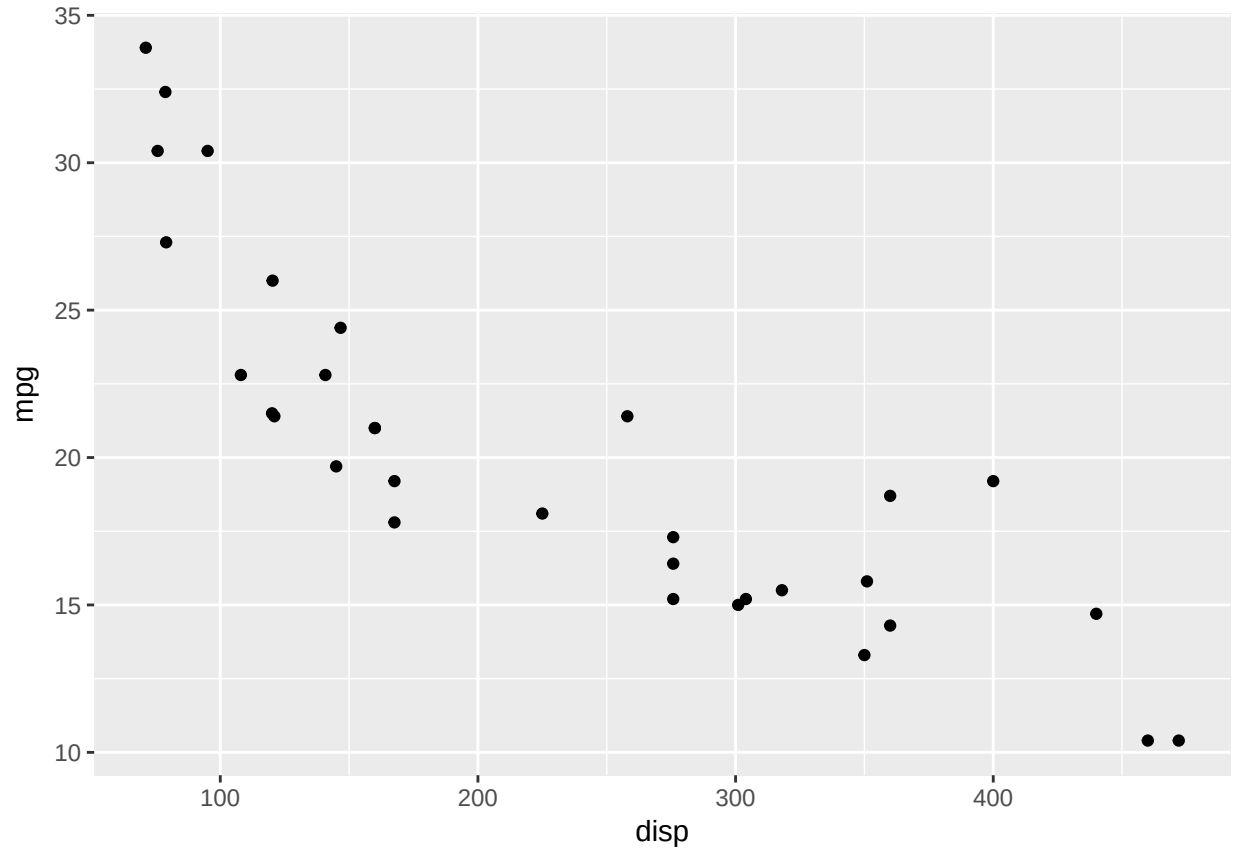
Fit Line - Intercept & Slope

We can fit a regression line using `geom_abline()` as well. It takes 2 arguments:

- `slope`
- `intercept`

We can get the `slope` and `intercept` by building a model using `lm()`.

```
ggplot(mtcars, aes(disp, mpg)) +
  geom_point() +
  geom_abline(slope = 29.59985, intercept = -0.04122)
```



Summary

In this chapter, we learnt to:

- build scatter plots
- map aesthetics to variables
- modify axis and legend
- fit regression line

Up Next..

In the next chapter, we will learn to build line charts.

Data Visulaization - Line Charts

Introduction

In the previous chapter, we learnt to build scatter plots. In this chapter, we will learn to

- build
  - simple line chart
  - grouped line chart
- map aesthetics to variables
- modify line
  - color
  - type
  - size

Libraries, Code & Data

We will use the following libraries in this chapter:

- readr
- ggplot2

All the data sets used in this chapter can be found here and code can be downloaded from here.

Case Study

We will use a data set related to GDP growth rate. You can download it from here. It contains GDP (Gross Domestic Product) growth data for the BRICS (Brazil, Russia, India, China, South Africa) for the years 2000 to 2005.

Data

```
gdp <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/gdp.csv')
```

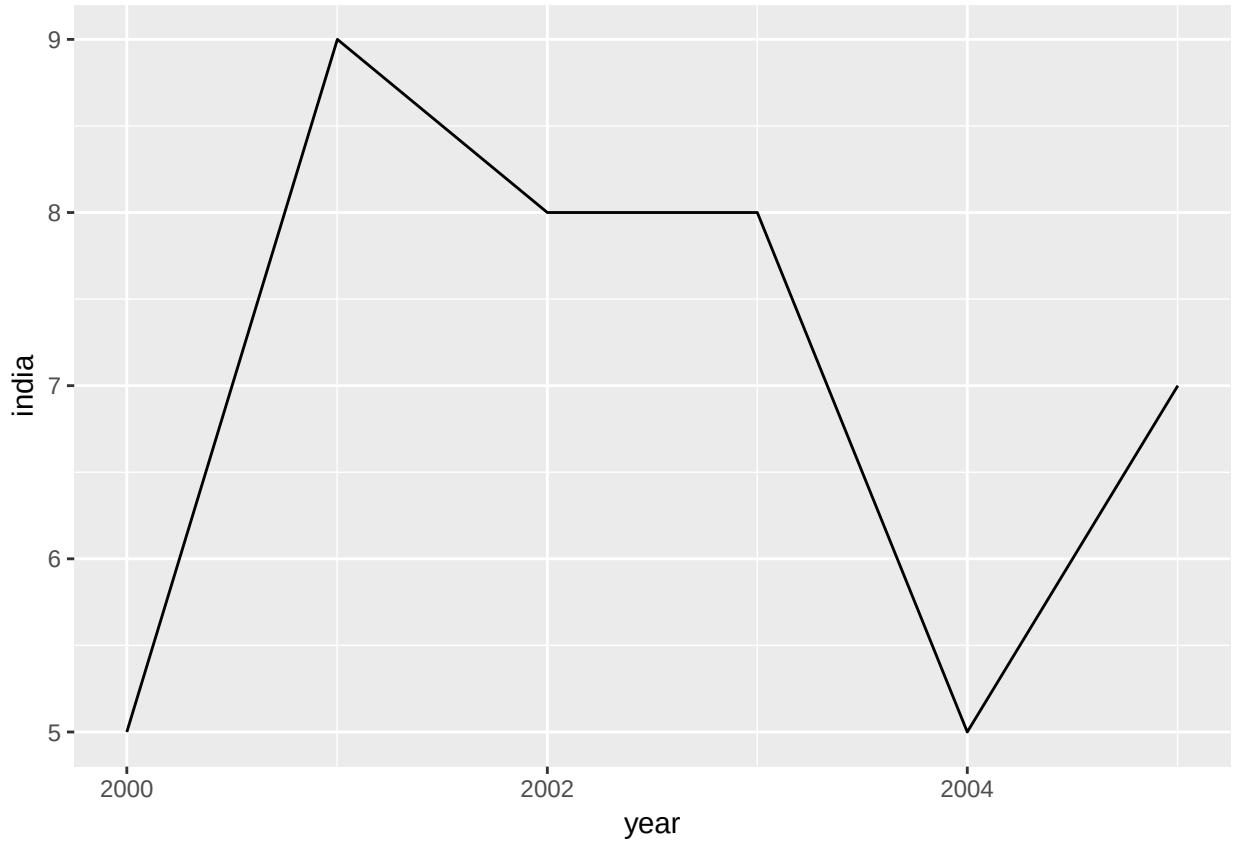
```
## Warning: Missing column names filled in: 'X1' [1]
```

```
gdp
```

```
## # A tibble: 6 x 6
##      X1     X year       growth india china
##   <int> <int> <date>      <int> <int> <int>
## 1     1     1 2000-01-01      6     5     8
## 2     2     2 2001-01-01      9     9     5
## 3     3     3 2002-01-01      8     8     6
## 4     4     4 2003-01-01      9     8     8
## 5     5     5 2004-01-01      9     5     9
## 6     6     6 2005-01-01      8     7     8
```

Line Chart

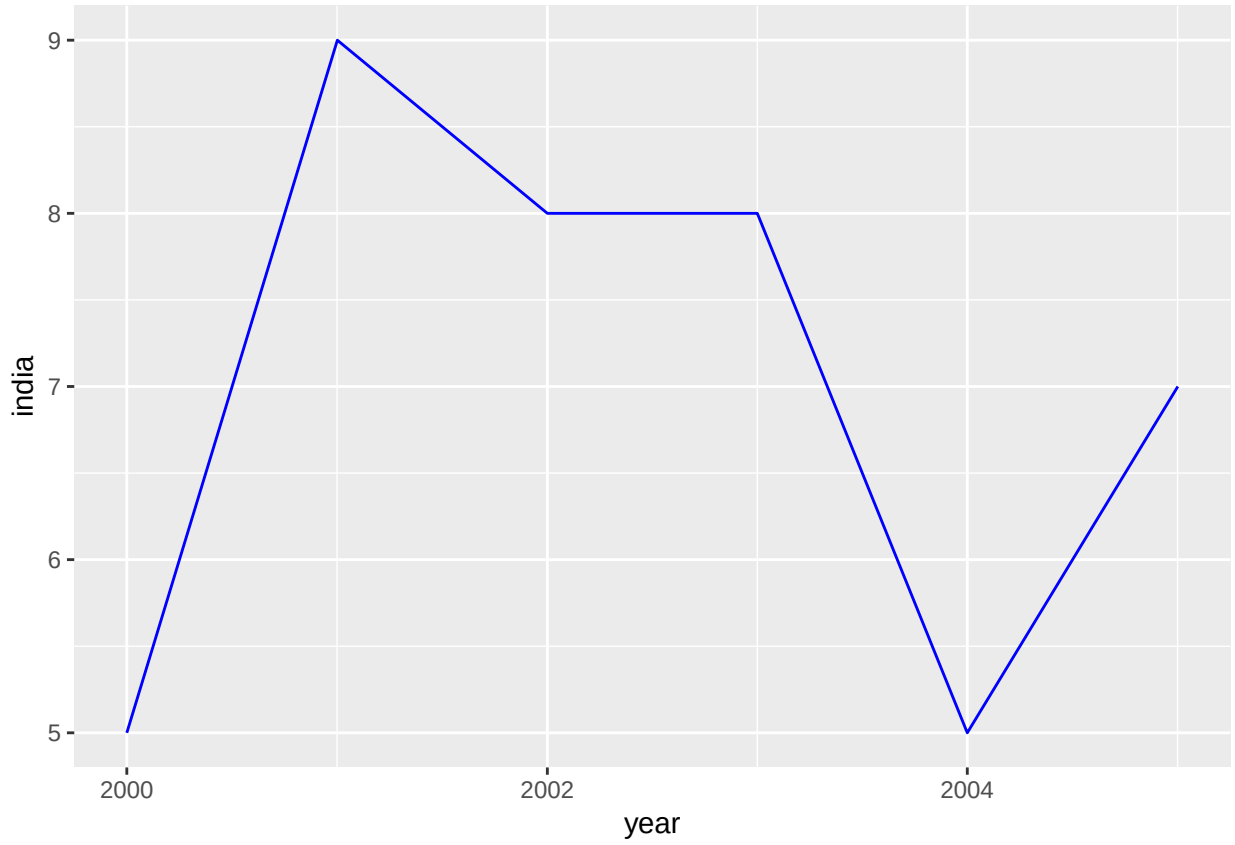
```
ggplot(gdp, aes(year, india)) +
  geom_line()
```



Line Color

The color of the line can be modified using the `color` argument in `geom_lin()`.

```
ggplot(gdp, aes(year, india)) +
  geom_line(color = 'blue')
```



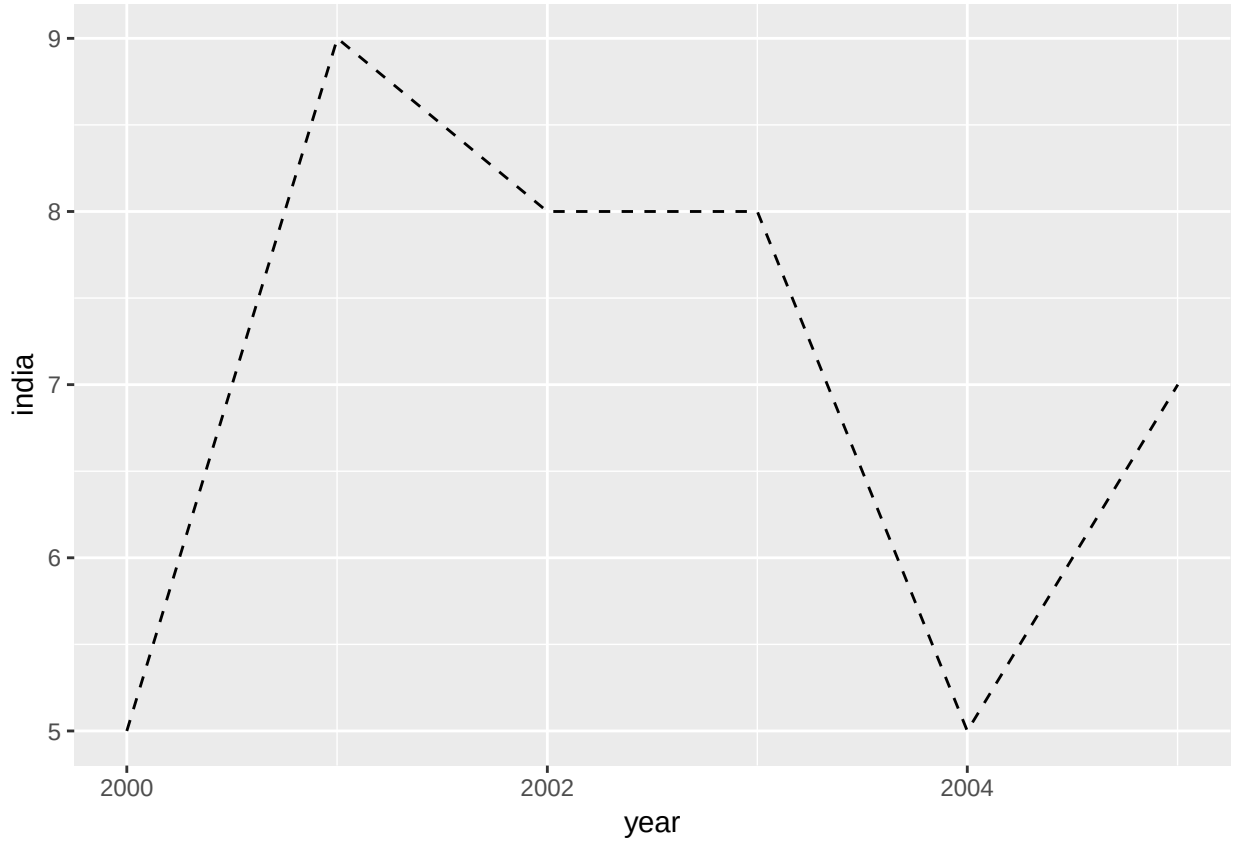
Line Type

- 0 : blank
- 1 : solid
- 2 : dashed
- 3 : dotted
- 4 : dotdash
- 5 : longdash
- 6 : twodash

Line Type

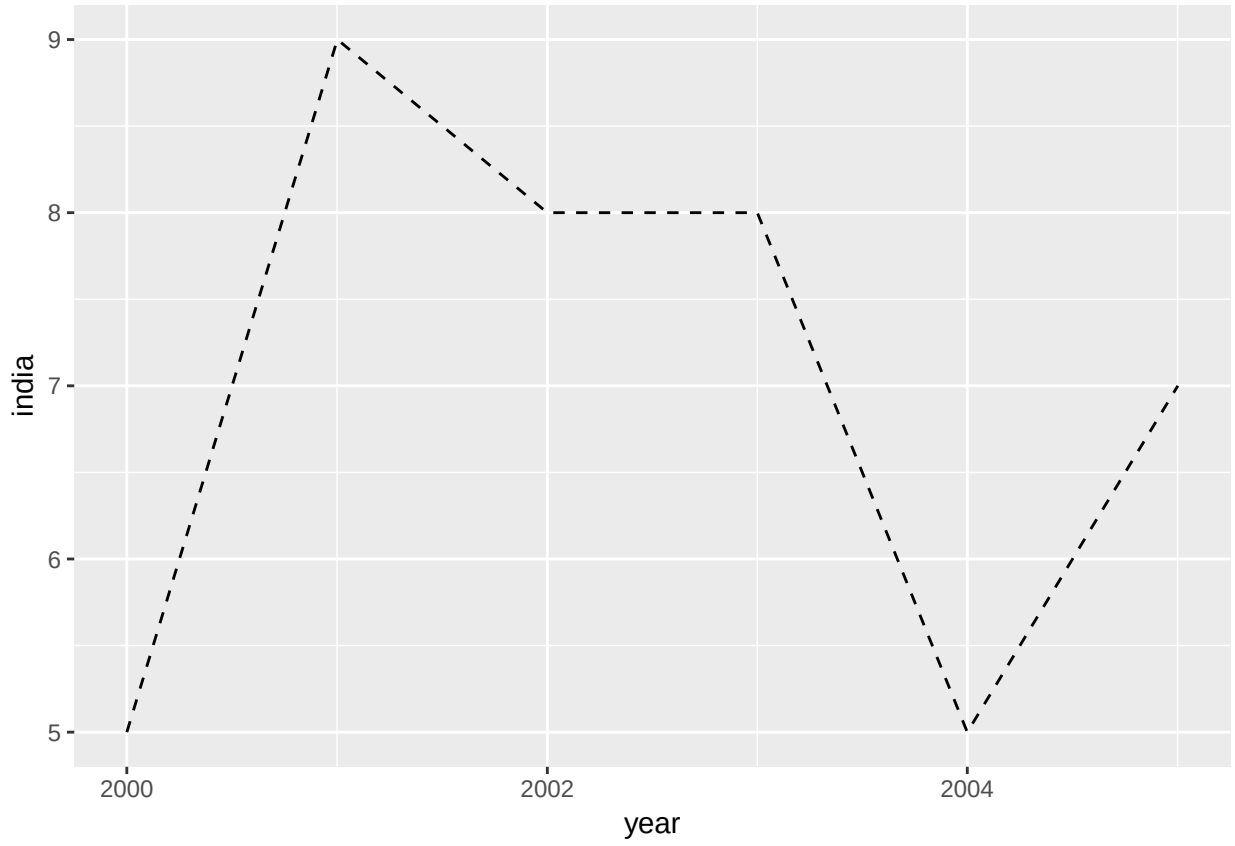
The type of line can be specified using `linetype` argument.

```
ggplot(gdp, aes(year, india)) +
  geom_line(linetype = 2)
```



Line Type (Dashed)

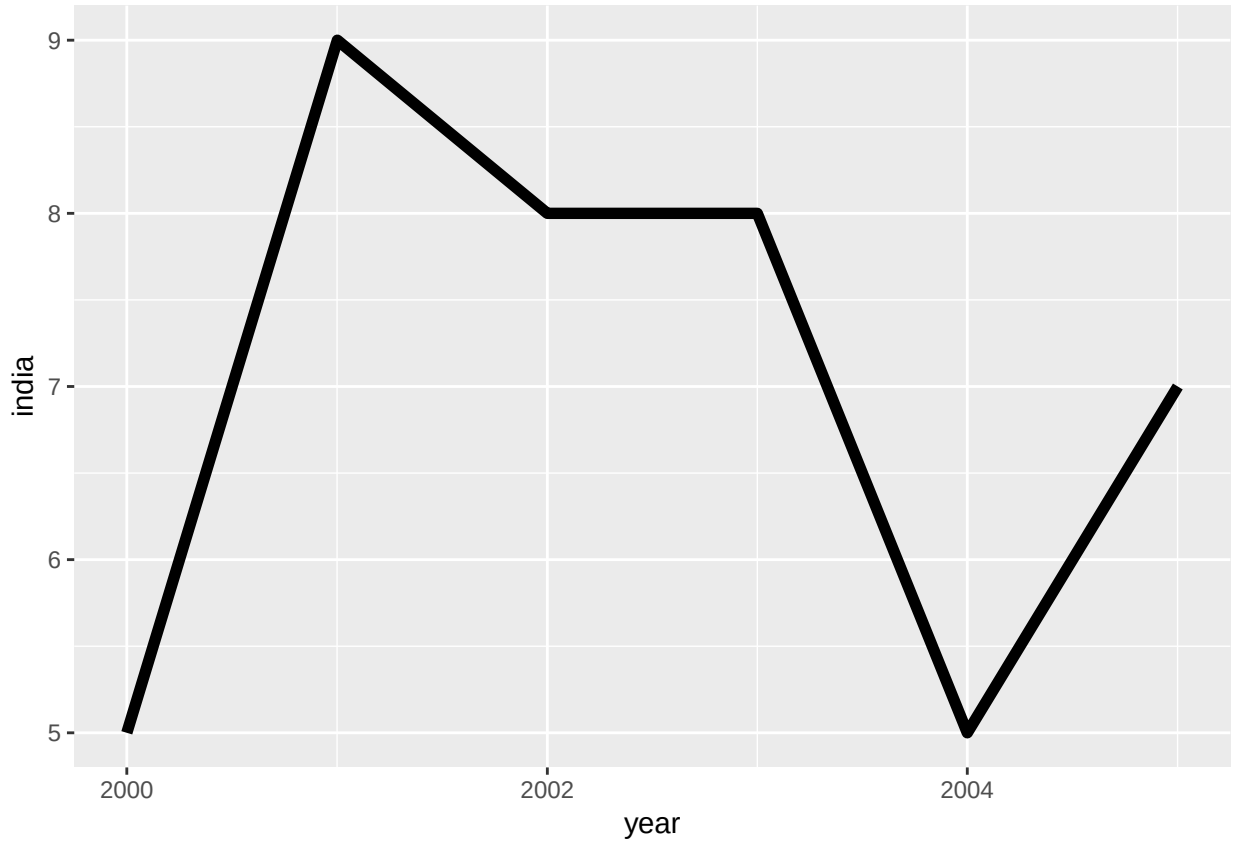
```
ggplot(gdp, aes(year, india)) +
  geom_line(linetype = 'dashed')
```



Line Size

The width of the line can be specified using the `size` argument.

```
ggplot(gdp, aes(year, india)) +
  geom_line(size = 2)
```



Modify Data

Now let us map the aesthetics to the variables. The data used in the above example cannot be used as we need a variable with country names. We will use `gather()` function from the `tidyr` package to reshape the data.

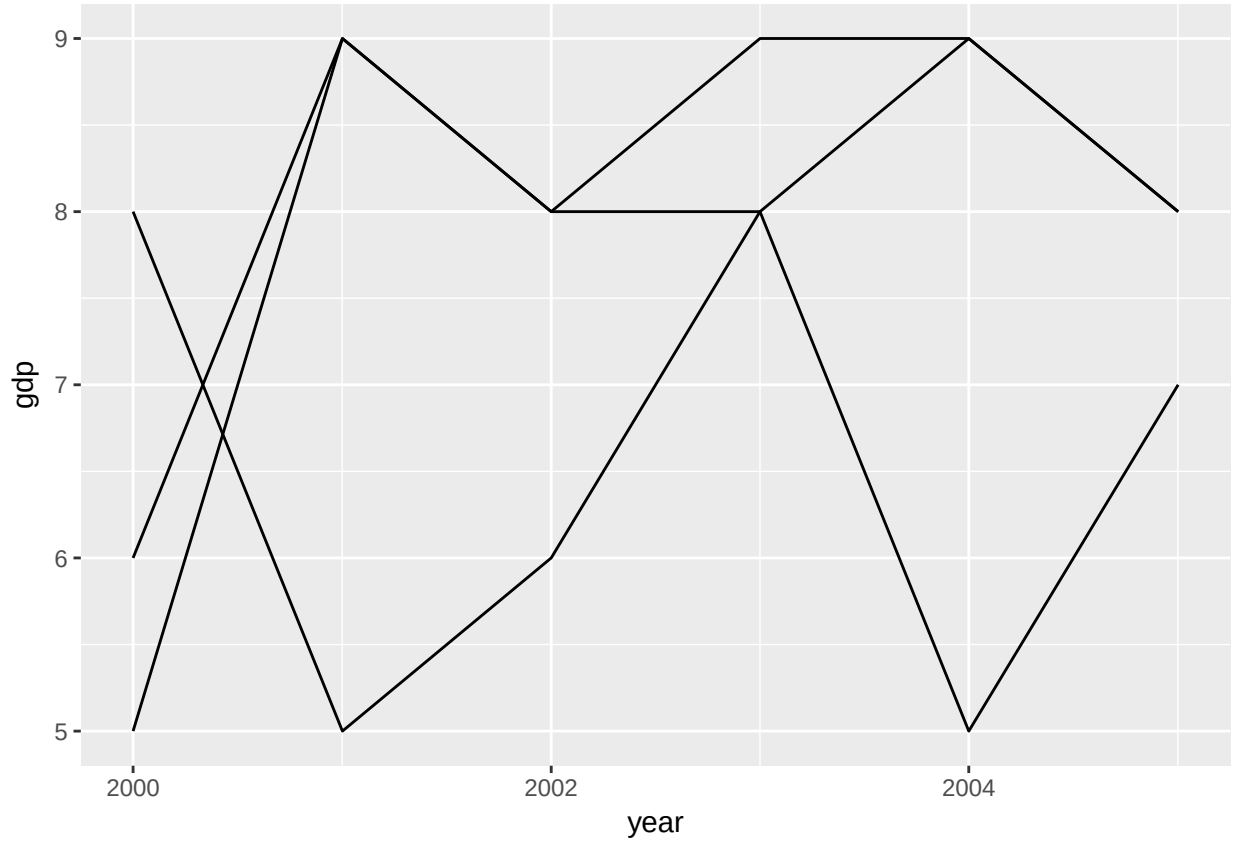
```
gdp2 <-
  gdp %>%
  select(year, growth, india, china) %>%
  gather(key = country, value = gdp, -year)
gdp2
```

```
## # A tibble: 18 x 3
##    year       country   gdp
##    <date>     <chr>   <int>
##  1 2000-01-01 growth      6
##  2 2001-01-01 growth      9
##  3 2002-01-01 growth      8
##  4 2003-01-01 growth      9
##  5 2004-01-01 growth      9
##  6 2005-01-01 growth      8
##  7 2000-01-01 india       5
##  8 2001-01-01 india       9
##  9 2002-01-01 india       8
## 10 2003-01-01 india       8
## 11 2004-01-01 india       5
## 12 2005-01-01 india       7
## 13 2000-01-01 china       8
## 14 2001-01-01 china       5
## 15 2002-01-01 china       6
## 16 2003-01-01 china       8
## 17 2004-01-01 china       9
## 18 2005-01-01 china       8
```

Grouped Line Chart

To create multiple lines, we can use the `group` argument and map it to a categorical variable. In the below example, we want to visualize the trend in GDP of different countries. Instead of using `geom_line()` multiple times, we map `group` argument to the variable `country` and we can visualize the GDP trend for all the countries at once.

```
ggplot(gdp2, aes(year, gdp, group = country)) +
  geom_line()
```



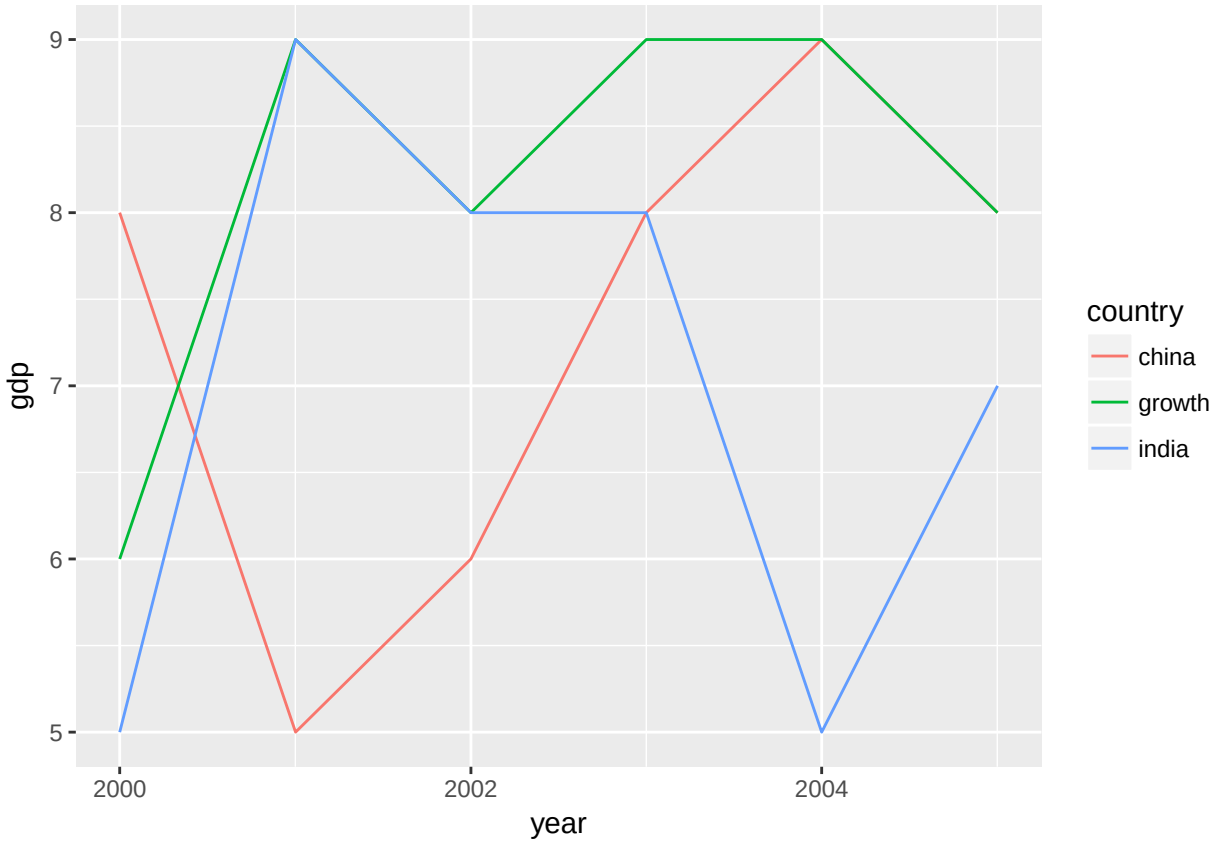
We can map aesthetics such as:

- color
- line type
- and line width

to categorical variables. In the next 3 examples, we map color, line type and line width to the `country` variable.

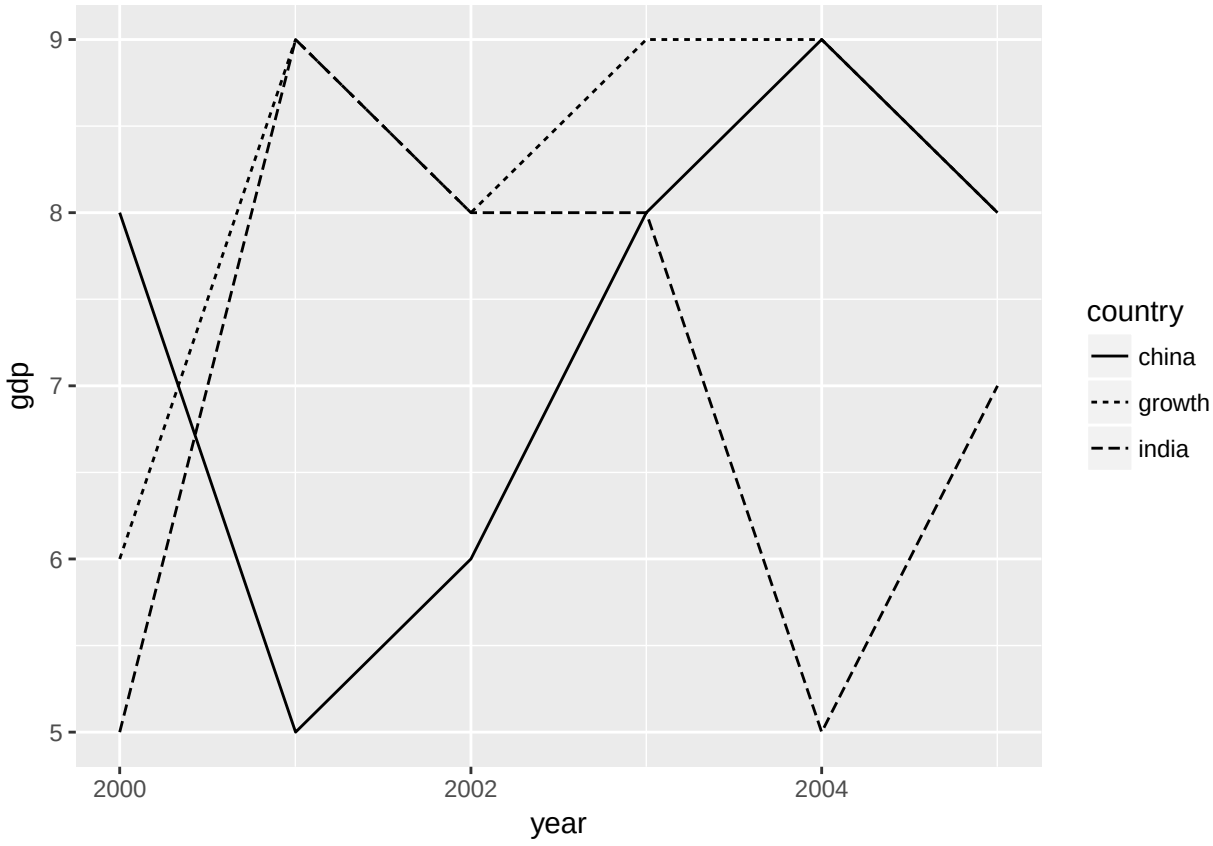
Map Color to Country

```
ggplot(gdp2, aes(year, gdp, group = country)) +
  geom_line(aes(color = country))
```



Map Line Type to Country

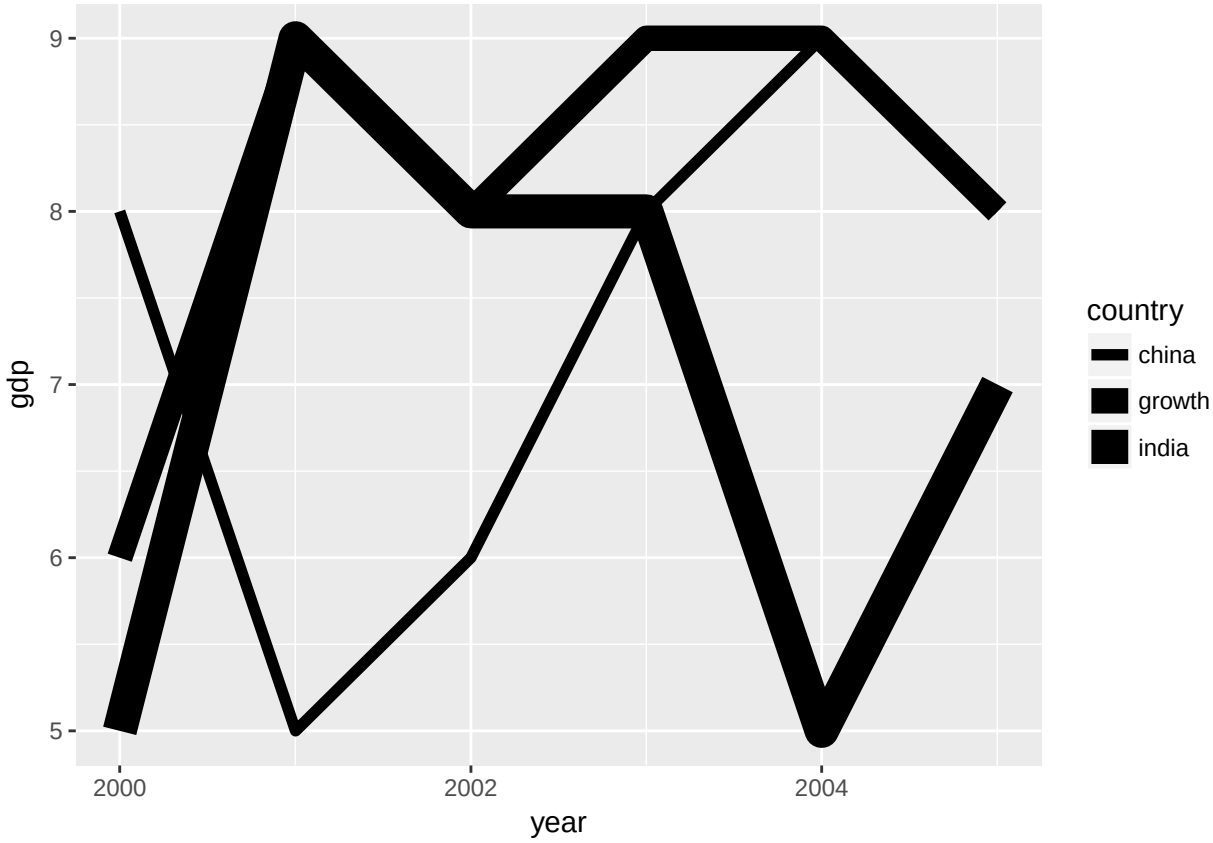
```
ggplot(gdp2, aes(year, gdp, group = country)) +
  geom_line(aes(linetype = country))
```



Map Line Width to Country

```
ggplot(gdp2, aes(year, gdp, group = country)) +
  geom_line(aes(size = country))
```

```
## Warning: Using size for a discrete variable is not advised.
```



Summary

In this chapter, we learnt to:

- build
  - simple line chart
  - grouped line chart
- map aesthetics to variables
- modify line
  - color
  - type
  - size

Up Next..

In the next chapter, we will learn to build bar plots.

Data Visulaization - Bar Plots

Introduction

In the previous chapter, we learnt to build line charts. In this chapter, we will learn to:

- build
  - simple bar plot
  - stacked bar plot
  - grouped bar plot
  - proportional bar plot
- map aesthetics to variables
- specify values for
  - bar color
  - bar line color
  - bar line type
  - bar line size

Libraries, Code & Data

We will use the following libraries in this chapter:

- readr
- ggplot2

All the data sets used in this chapter can be found here and code can be downloaded from here.

Data

```
ecom <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/web.csv')
ecom
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     4 bing     tablet false          3   18.0     468   China
##  5     5 yahoo    mobile true           9    1.00    955   Poland
##  6     6 yahoo    laptop false          5    5.00    135   South Africa
##  7     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  8     8 direct   mobile true          10    1.00    908   Indonesia
##  9     9 bing     mobile false          3   19.0     209   Netherlands
## 10    10 google   mobile true           6    1.00    208   Czech Republic
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Data Dictionary

- id: row id
- referrer: referrer website/search engine
- os: operating system

- browser: browser
- device: device used to visit the website
- n_pages: number of pages visited
- duration: time spent on the website (in seconds)
- repeat: frequency of visits
- country: country of origin
- purchase: whether visitor purchased
- order_value: order value of visitor (in dollars)

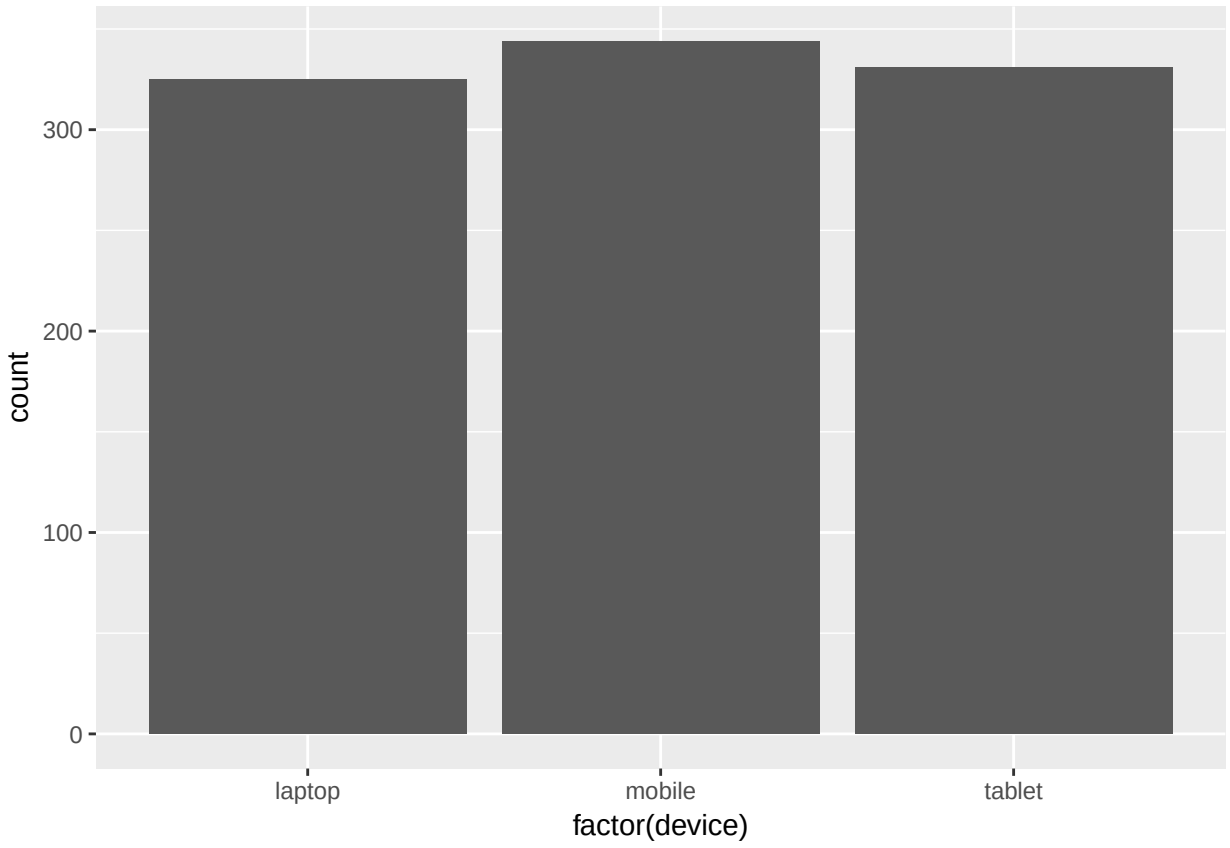
Aesthetics

- `fill`
- `color`
- `linetype`
- `size`
- `position`

Simple Bar Plot

Let us start by building a simple bar plot using `geom_bar()`. We will look at the distribution of devices that drive traffic to a fictional website.

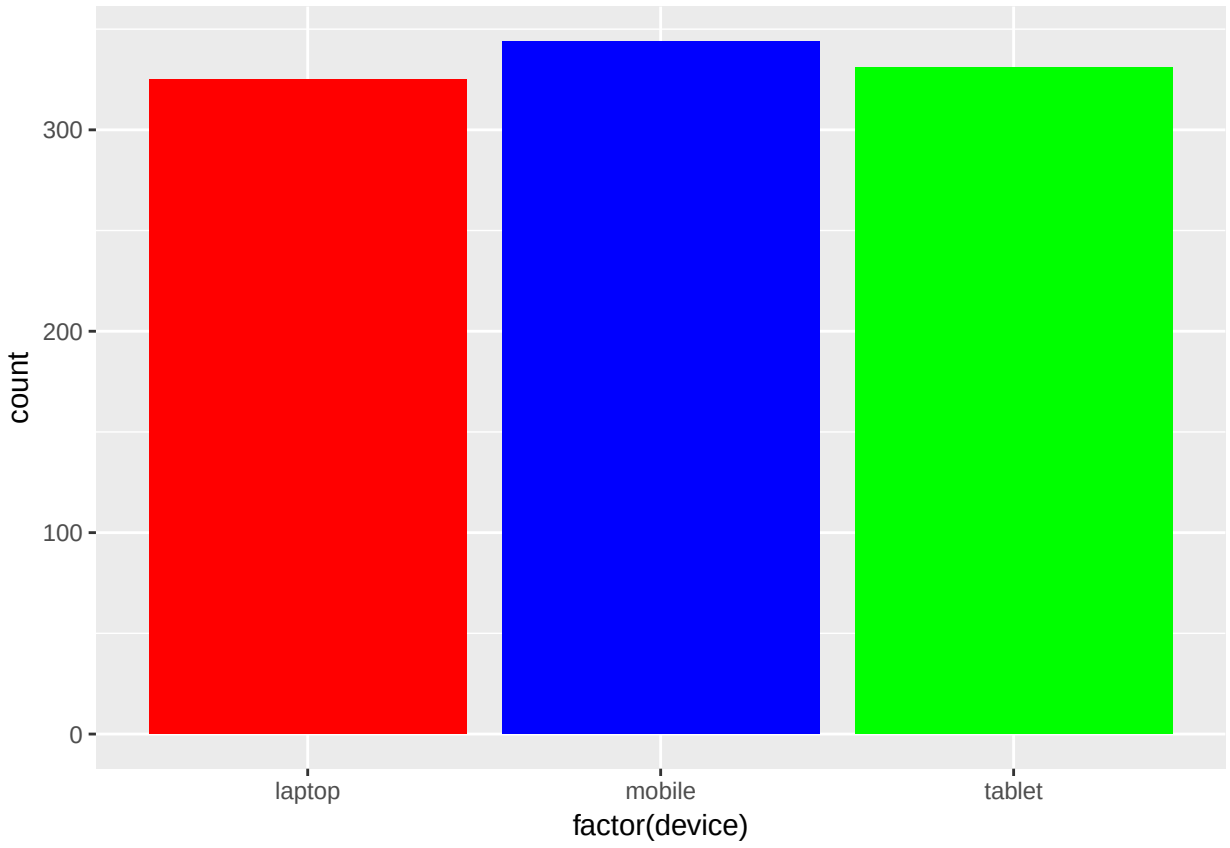
```
ggplot(ecom) +
  geom_bar(aes(factor(device)))
```



Bar Color

Use the `fill` argument to modify the color of the bars.

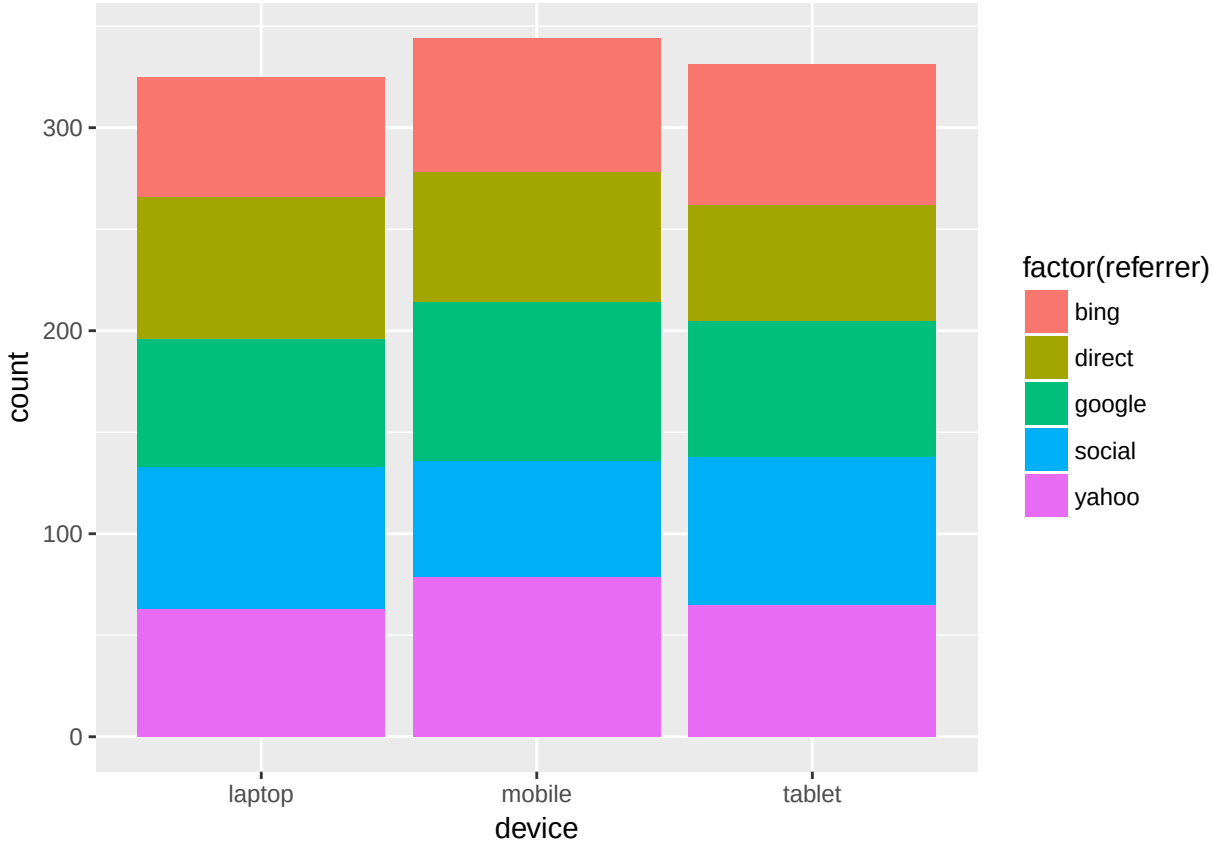
```
ggplot(ecom) +
  geom_bar(aes(factor(device)),
           fill = c('red', 'blue', 'green'))
```



Stacked Bar Plot

To build a stacked bar plot, map the `fill` argument to another categorical variable. In the below example, we map `fill` to `referrer`.

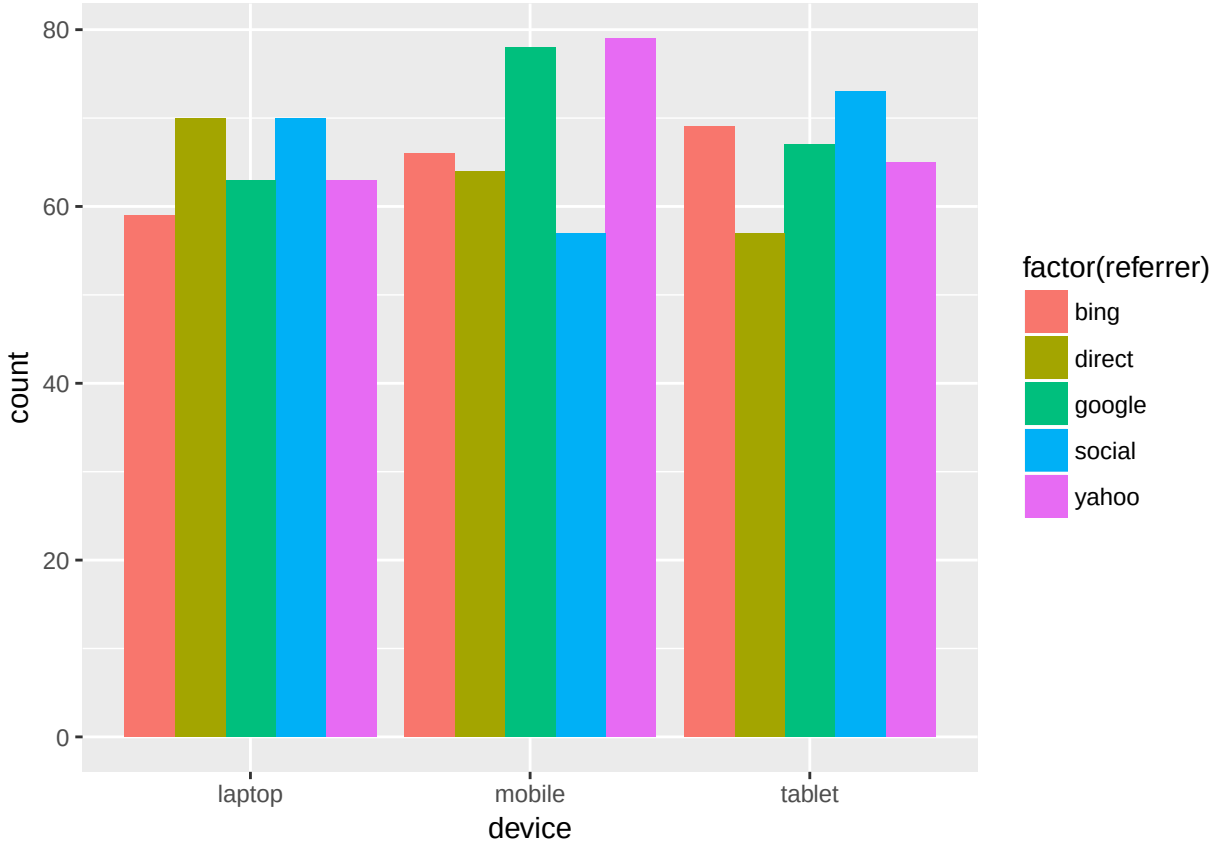
```
ggplot(ecom) +
  geom_bar(aes(device, fill = factor(referrer)))
```



Grouped Bar Plot

Use the `position` argument to create a grouped bar plot. Assign the value `dodge` to `position` argument. Instead of stacking the bars on top of one another, the bar are placed next to each other.

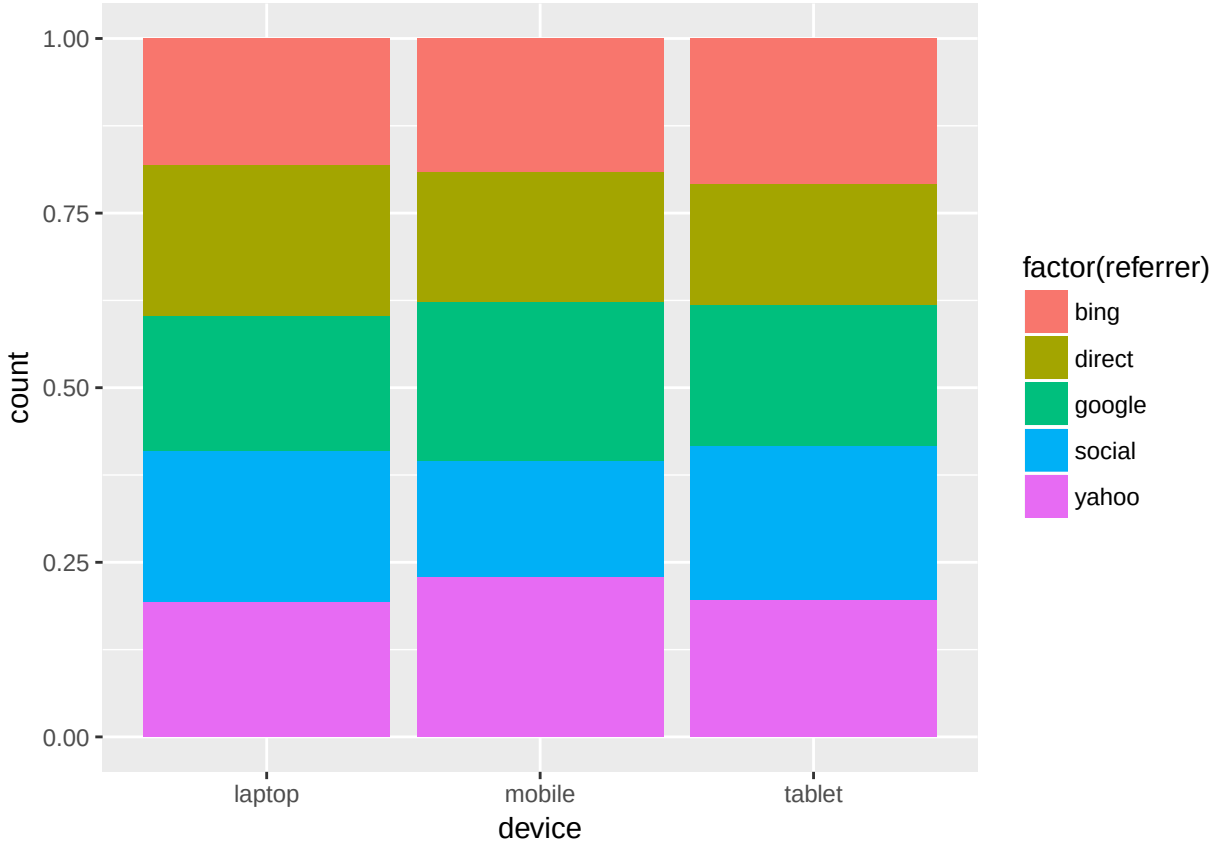
```
ggplot(ecom) +
  geom_bar(aes(device, fill = factor(referrer)), position = 'dodge')
```



Proportional Bar Plot

Proportional bar plots can be created by assigning the value `fill` to the `position` argument. In a proportional bar plot, the height of all the bars is same.

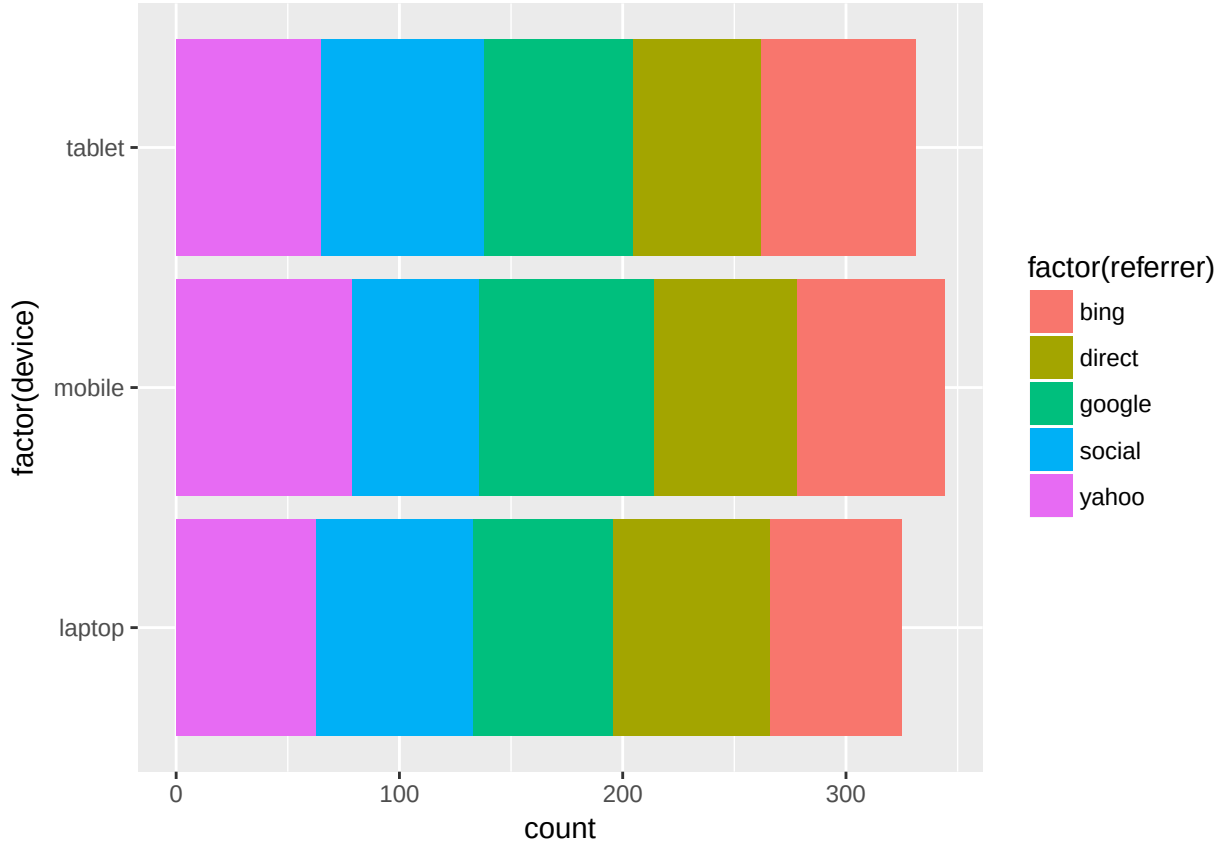
```
ggplot(ecom) +
  geom_bar(aes(device, fill = factor(referrer)), position = 'fill')
```



Horizontal Bar Plot

Use `coord_flip()` to create a horizontal bar plot.

```
ggplot(ecom) +
  geom_bar(aes(factor(device), fill = factor(referrer))) +
  coord_flip()
```



Summary

In this chapter, we learnt to:

- build
  - simple bar plot
  - stacked bar plot
  - grouped bar plot
  - proportional bar plot
- map aesthetics to variables
- specify values for
  - bar color
  - bar line color
  - bar line type
  - bar line size

Up Next..

In the next chapter, we will learn to build box plots.

Data Visulaization - Box Plots

Introduction

In the previous chapter, we learnt how to build bar charts. In this chapter, we will

- build box plots
- modify box
  - color
  - fill
  - alpha
  - line size
  - line type
- modify outlier
  - color
  - shape
  - size
  - alpha

Libraries, Code & Data

We will use the following libraries in this chapter:

- readr
- ggplot2

All the data sets used in this chapter can be found here and code can be downloaded from here.

Data

```
ecom <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/web.csv')
ecom
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     4 bing     tablet false          3   18.0     468   China
##  5     5 yahoo    mobile true           9    1.00    955   Poland
##  6     6 yahoo    laptop false          5    5.00    135   South Africa
##  7     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  8     8 direct   mobile true          10    1.00    908   Indonesia
##  9     9 bing     mobile false          3   19.0     209   Netherlands
## 10    10 google   mobile true           6    1.00    208   Czech Republic
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Data Dictionary

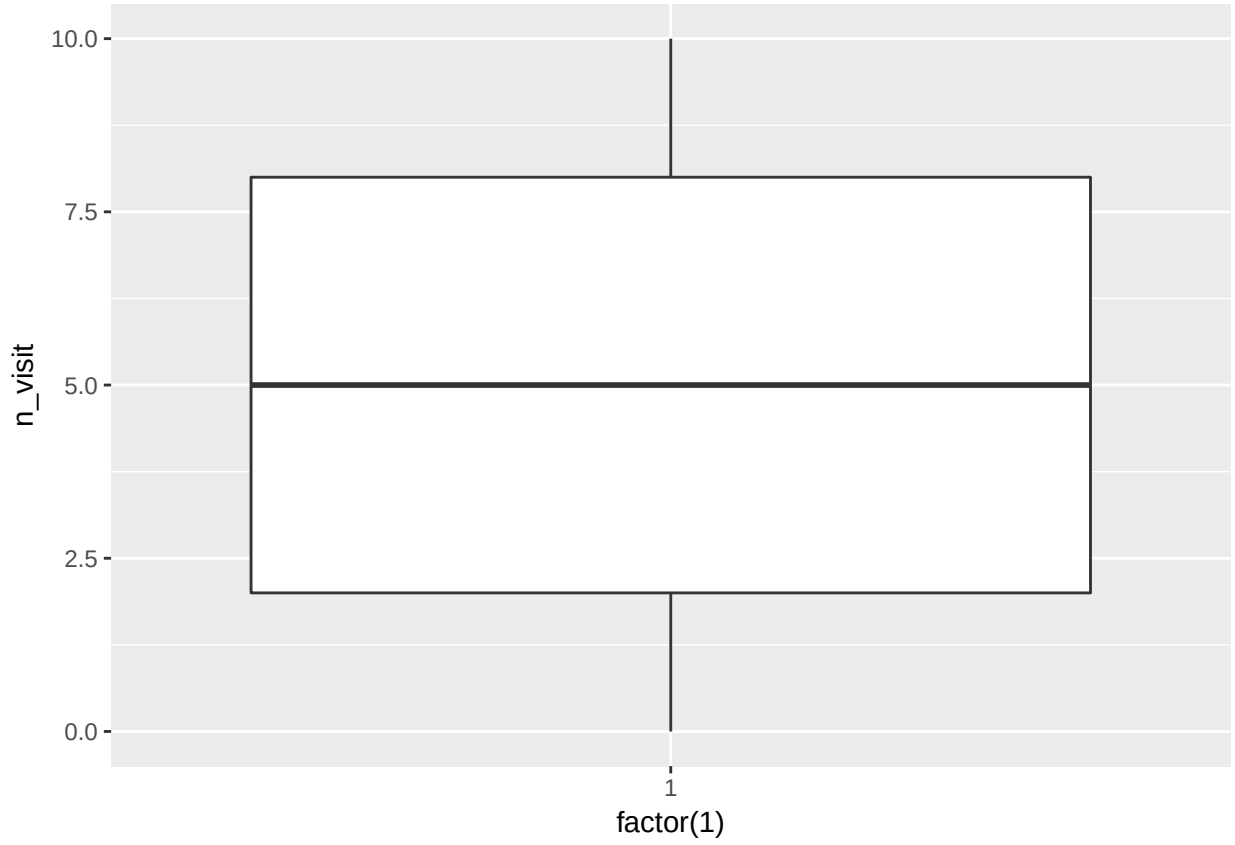
- id: row id
- referrer: referrer website/search engine

- os: operating system
- browser: browser
- device: device used to visit the website
- n_pages: number of pages visited
- duration: time spent on the website (in seconds)
- repeat: frequency of visits
- country: country of origin
- purchase: whether visitor purchased
- order_value: order value of visitor (in dollars)

Univariate Box Plot

Let us create a univariate box plot i.e. we are not comparing the distribuion of the variable across groups. In `geom_boxplot()`, we must map the `x` aesthetic to a variable else it will return an error. In order to create the box plot, we will assing the value `factor(1)` to the `x` aesthetic and the variable whose distribution we are examining to the `y` aesthetic.

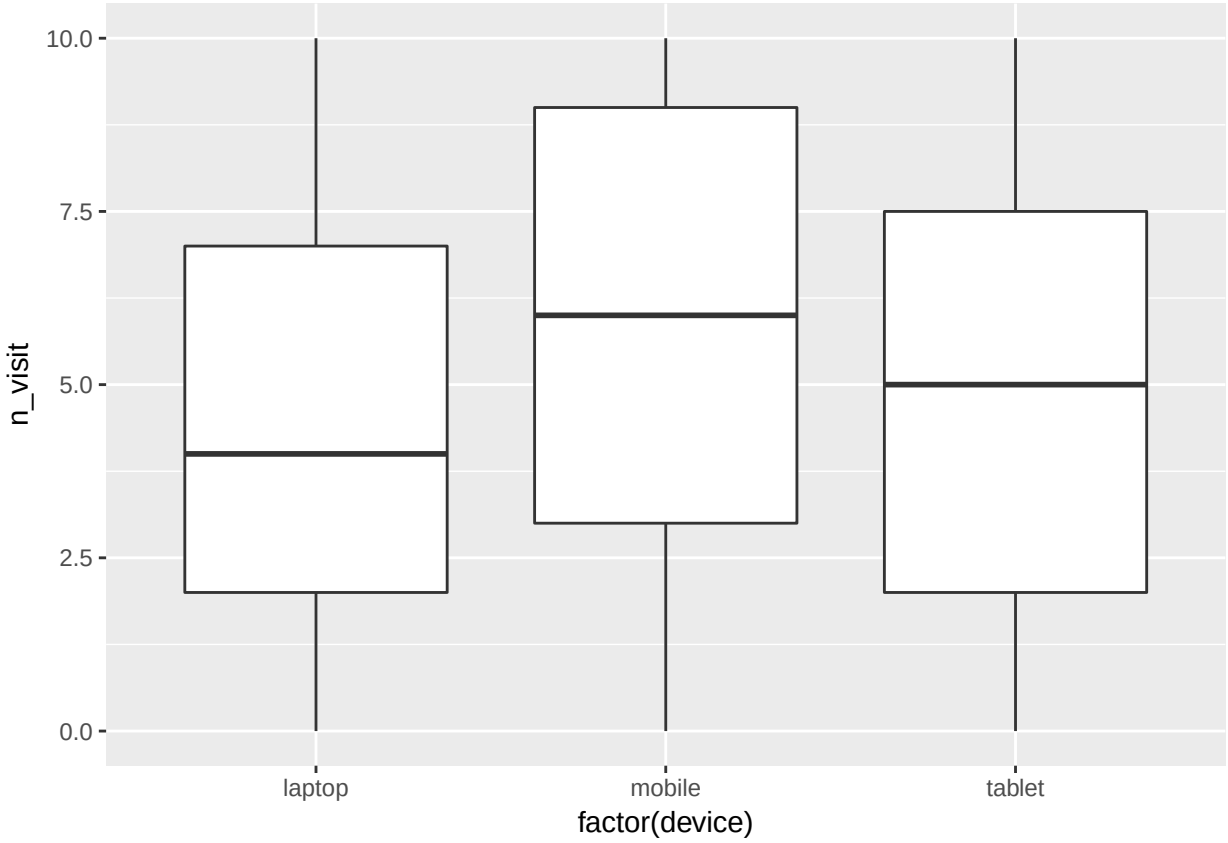
```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(1), y = n_visit))
```



Box Plot

If we are comparing the distribution of a variable across groups, we can map the `x` aesthetic to a categorical variable. In the below example, we are comparing the distribution of `n_visit` across different device types.

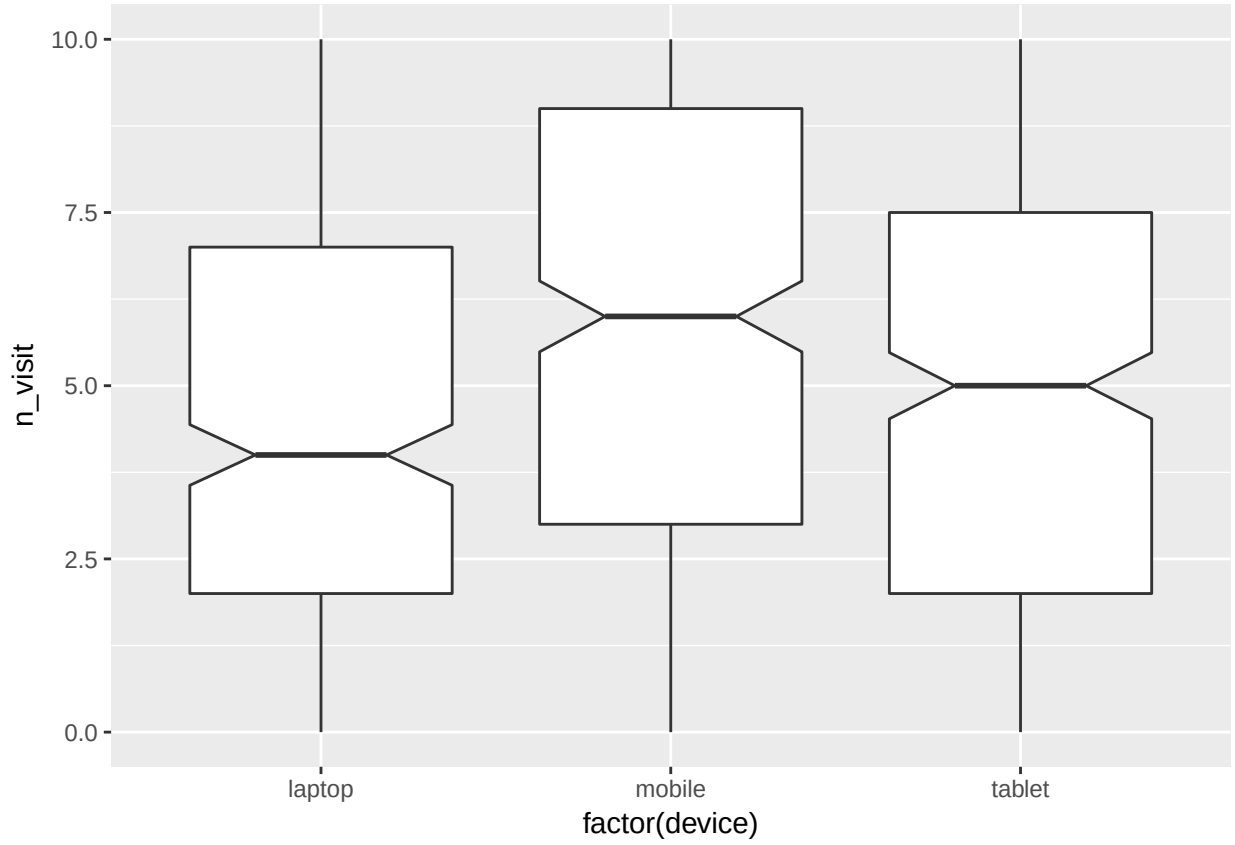
```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = n_visit))
```



Notch

If we want to test whether the medians of the different groups differ, we can use the `notch` argument and set it to `TRUE`. A notch is drawn in each side of the boxes and if the notches of the plots do not overlap, it is strong evidence that the medians differ.

```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = n_visit),
               notch = TRUE)
```



Outliers

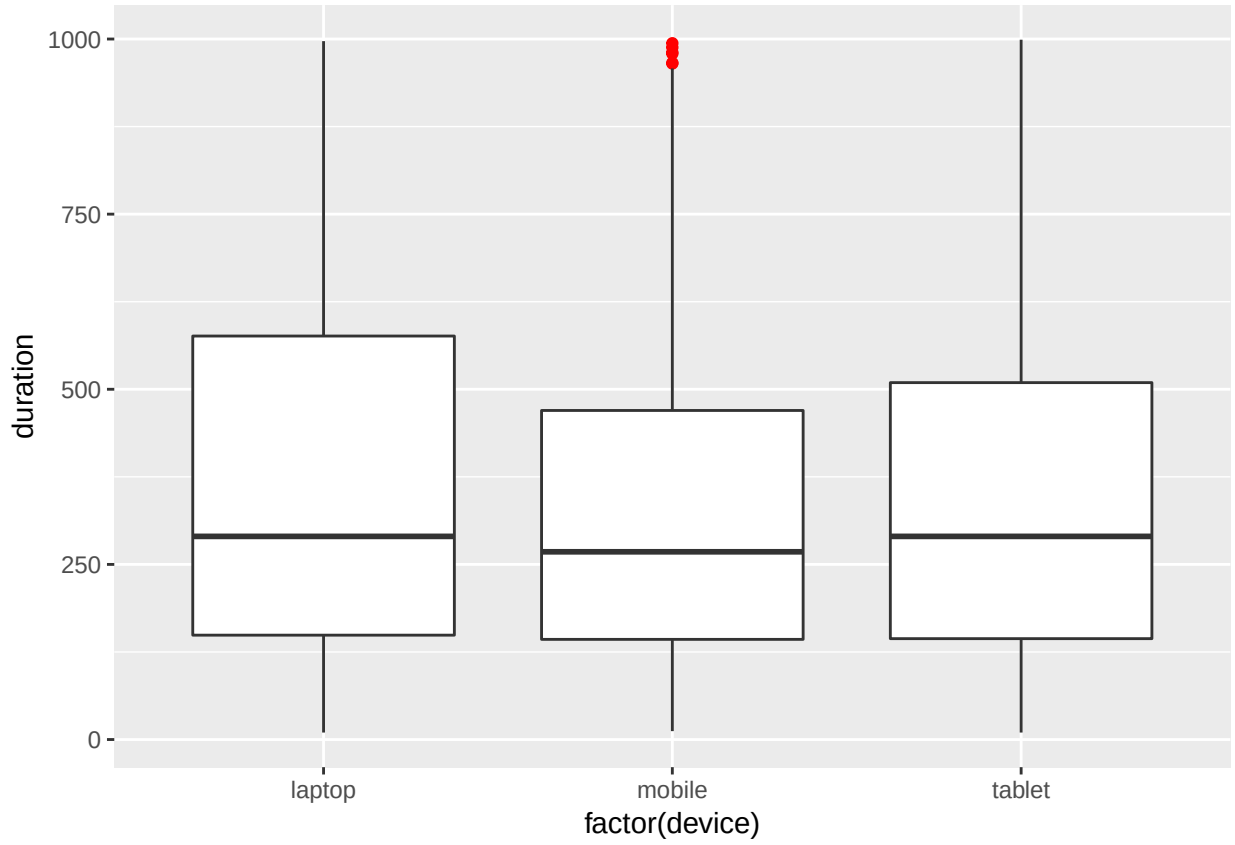
The box plot is useful in detecting outliers in the data. In this section, we will learn to modify the appearance of the outlier using:

- `outlier.color`
- `outlier.shape`
- `outlier.size`
- `outlier.opacity`

In the next 4 examples, we will modify the appearance of the outlier.

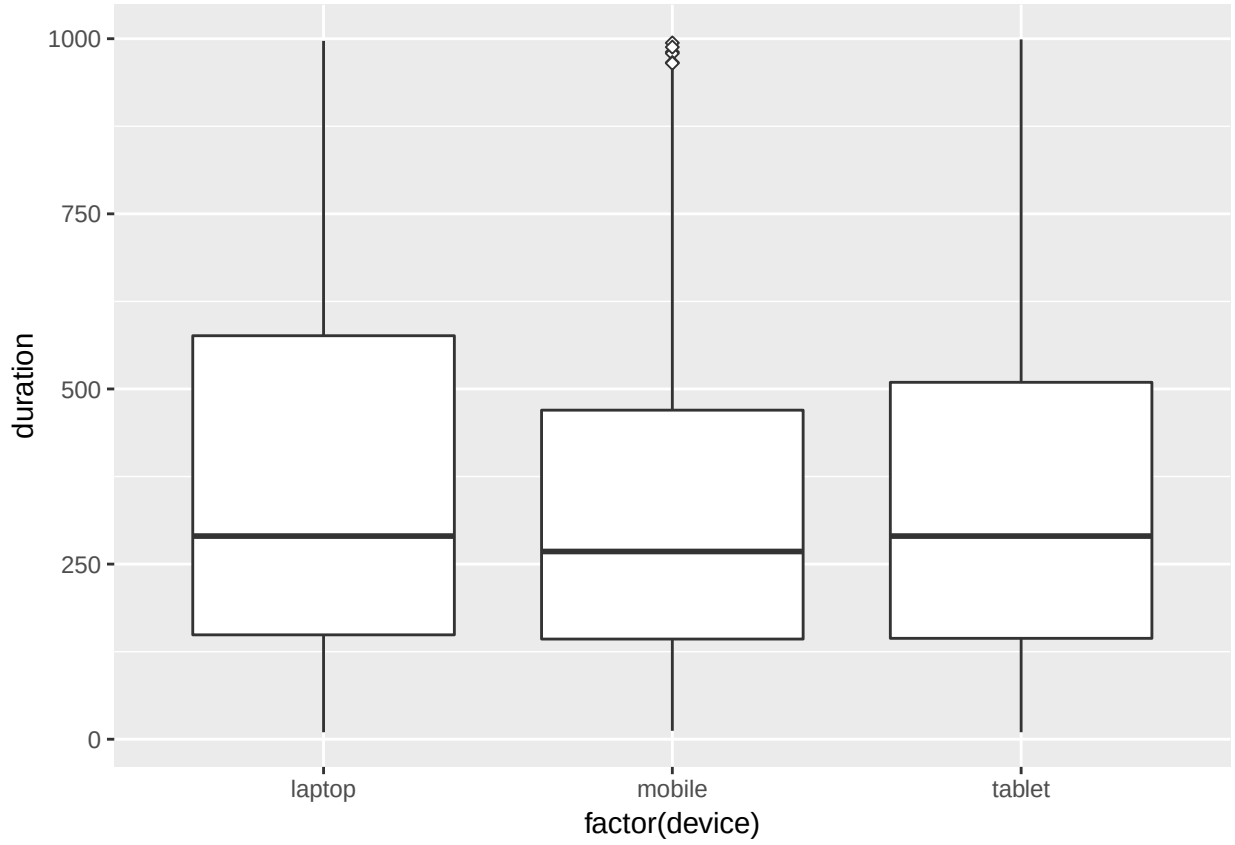
Outlier Color

```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = duration),
               outlier.color = 'red')
```



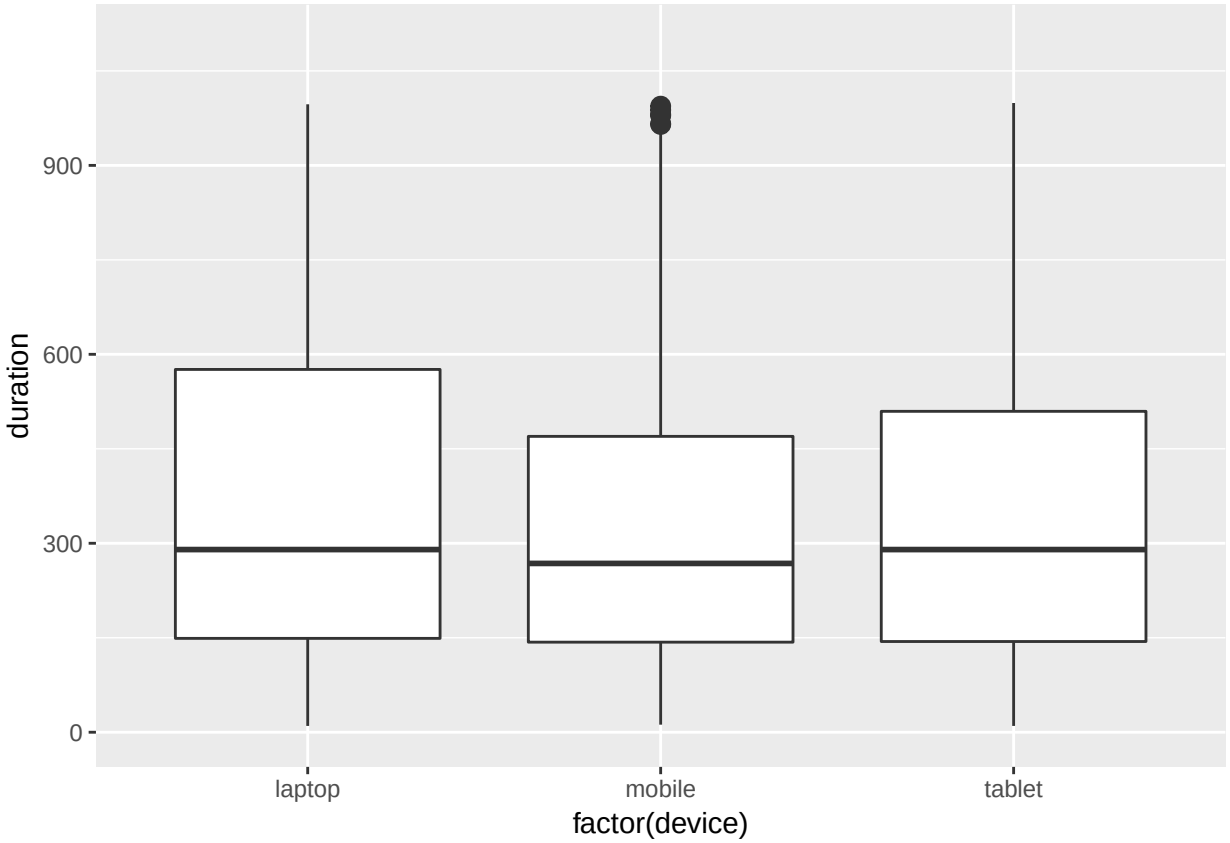
Outlier Shape

```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = duration),
               outlier.shape = 23)
```



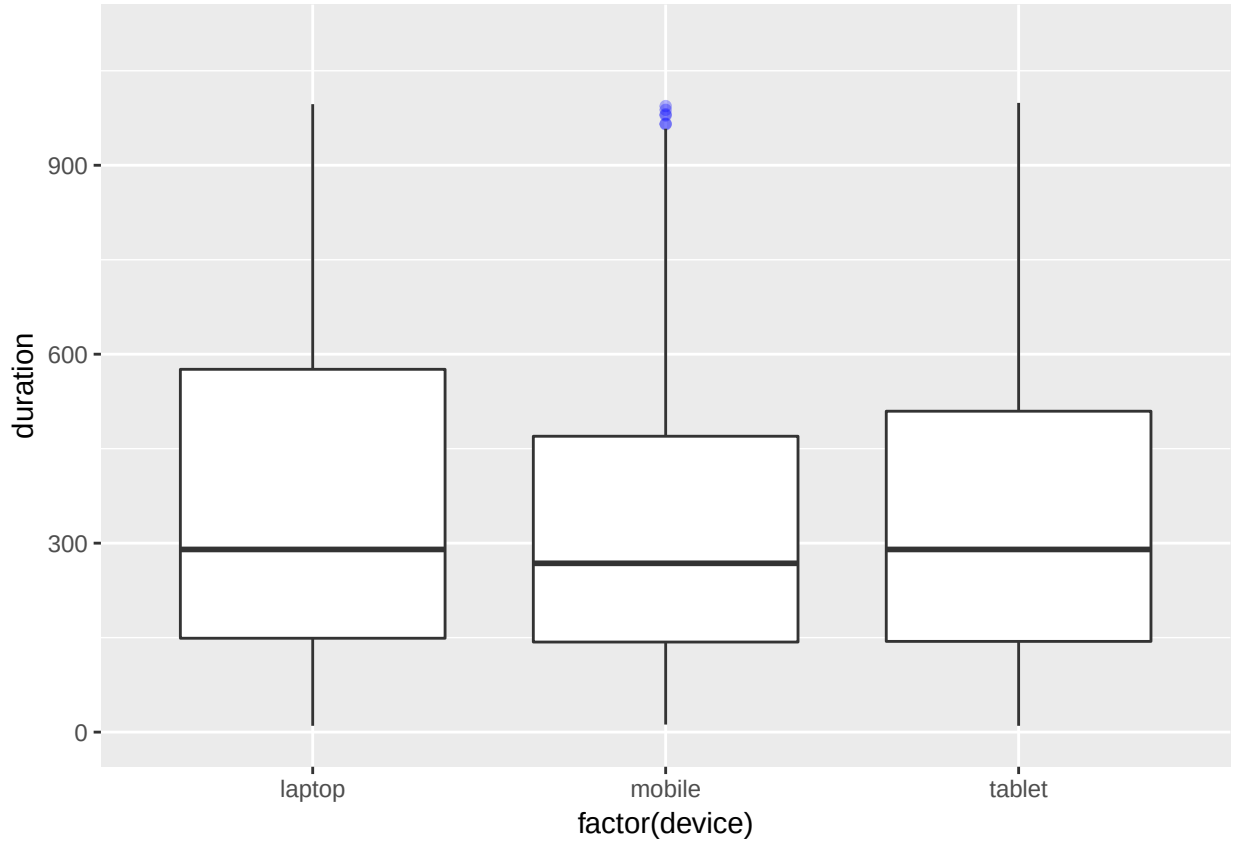
Outlier Size

```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = duration), outlier.size = 3) +
  expand_limits(y = c(0, 1100))
```



Outlier Alpha

```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = duration),
               outlier.color = 'blue', outlier.alpha = 0.3) +
  expand_limits(y = c(0, 1100))
```



Box Aesthetics

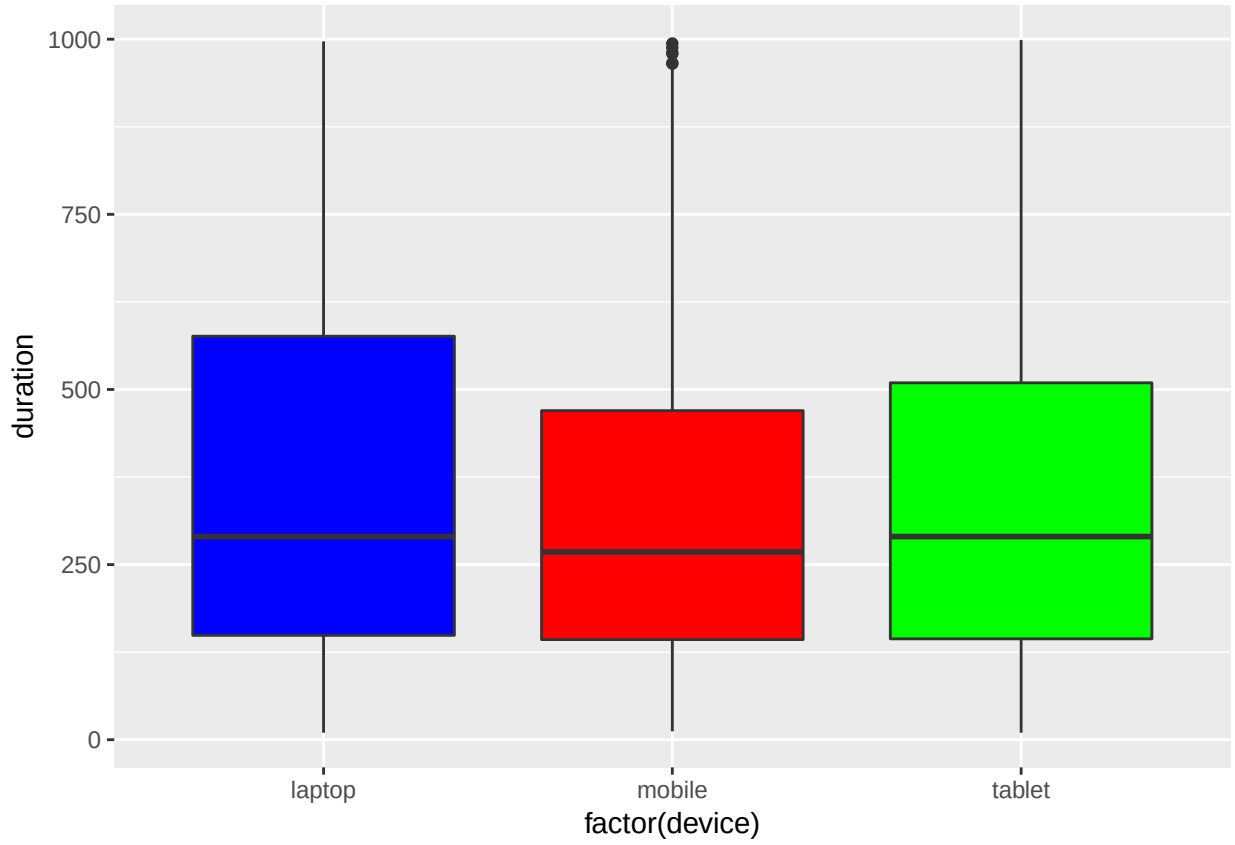
In this section, we will learn to modify the appearance of the box.

Background Color

The background color of the box can be modified using the `fill` argument. We can either map it to variables or specify values for each box in the plot.

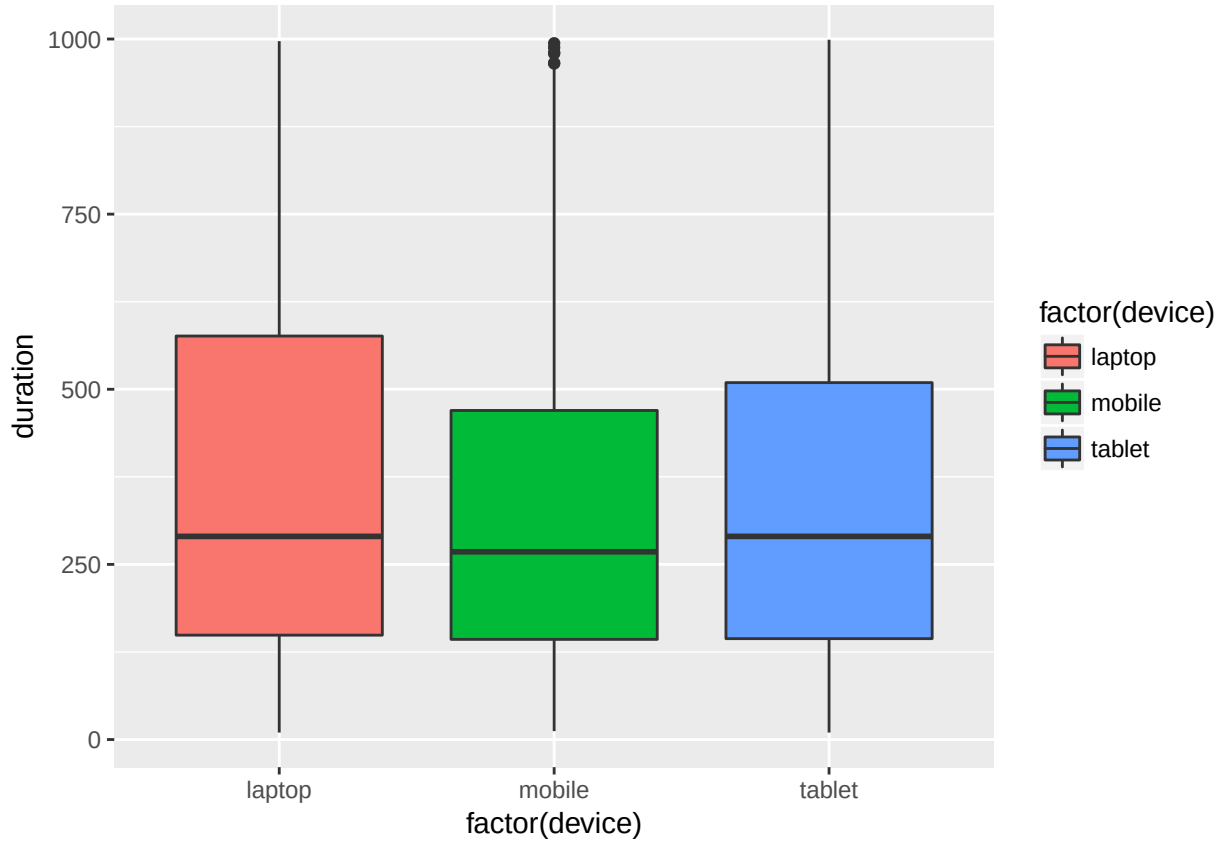
Specify Values for Fill

```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = duration),
               fill = c('blue', 'red', 'green'))
```



Map Fill to Variable

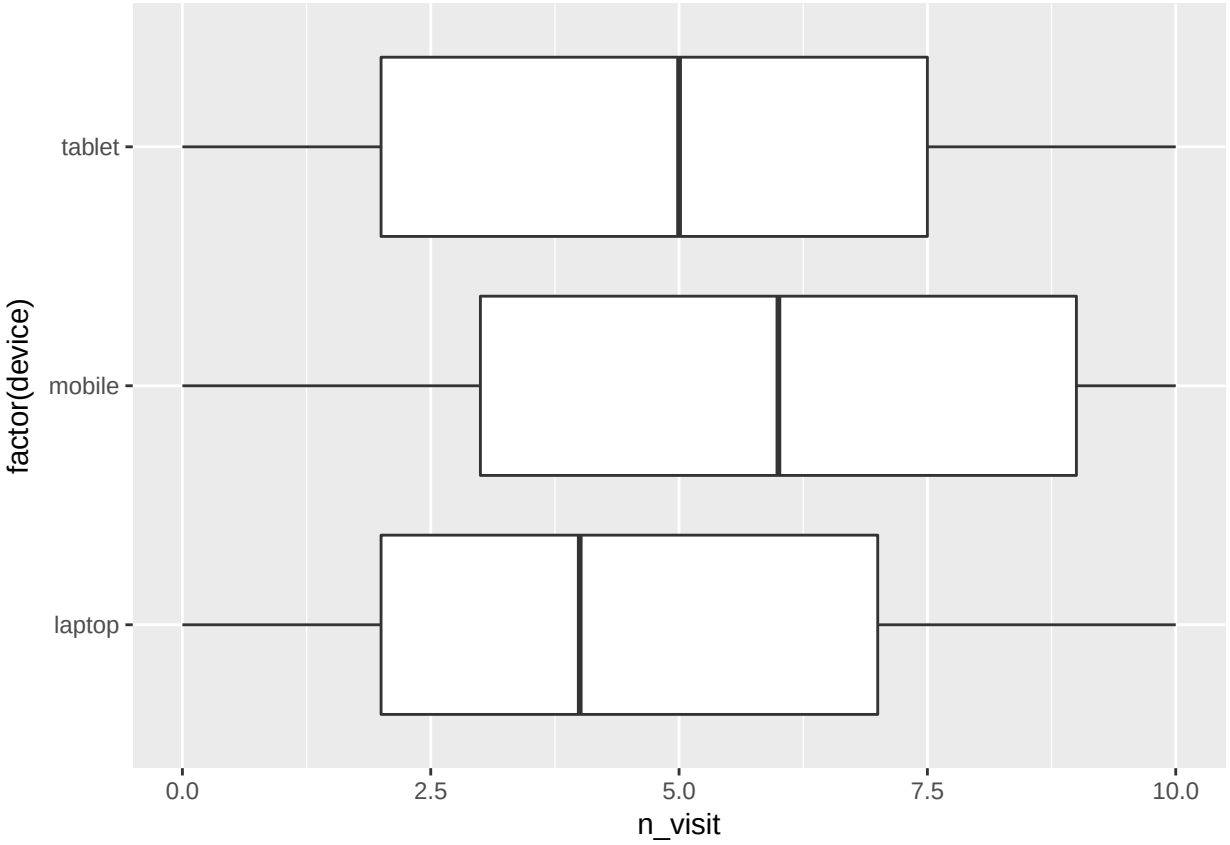
```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = duration,
                   fill = factor(device)))
```



Horizontal Box Plot

Use `coord_flip()` to create a horizontal box plot.

```
ggplot(ecom) +
  geom_boxplot(aes(x = factor(device), y = n_visit)) +
  coord_flip()
```



Summary

In this chapter, we learnt to:

- build box plots
- modify outlier color, shape, size etc.
- modify box color

Up Next..

In the next chapter, we will learn to build histograms.

Data Visulaization - Histograms

Introduction

In the previous chapter, we learnt to build box plots. In this chapter, we will learn to

- build histogram
- specify bins
- modify
  - color
  - fill
  - alpha
  - bin width
  - line type
  - line size

A histogram is a plot that can be used to examine the shape and spread of continuous data. It looks very similar to a bar graph and can be used to detect outliers and skewness in data. The histogram graphically shows the following:

- center (location) of the data
- spread (dispersion) of the data
- skewness
- outliers
- presence of multiple modes

To construct a histogram, the data is split into intervals called bins. The intervals may or may not be equal sized. For each bin, the number of data points that fall into it are counted (frequency). The Y axis of the histogram represents the frequency and the X axis represents the variable.

Libraries, Code & Data

We will use the following libraries in this chapter:

- readr
- ggplot2

All the data sets used in this chapter can be found here and code can be downloaded from here.

Data

```
ecom <- readr::read_csv('https://raw.githubusercontent.com/rsquaredacademy/datasets/master/web.csv')
ecom
```

```
## # A tibble: 1,000 x 11
##       id referrer device bouncers n_visit n_pages duration country
##    <int> <chr>    <chr>  <chr>      <int>   <dbl>    <dbl> <chr>
##  1     1 google   laptop true          10    1.00    693   Czech Republic
##  2     2 yahoo    tablet true           9    1.00    459   Yemen
##  3     3 direct   laptop true           0    1.00    996   Brazil
##  4     4 bing     tablet false          3   18.0     468   China
##  5     5 yahoo    mobile true           9    1.00    955   Poland
##  6     6 yahoo    laptop false          5    5.00    135   South Africa
##  7     7 yahoo    mobile true          10    1.00     75.0 Bangladesh
##  8     8 direct   mobile true          10    1.00    908   Indonesia
```

```
##  9     9 bing     mobile false          3   19.0     209   Netherlands
## 10    10 google   mobile true           6    1.00    208   Czech Republic
## # ... with 990 more rows, and 3 more variables: purchase <chr>,
## #   order_items <dbl>, order_value <dbl>
```

Data Dictionary

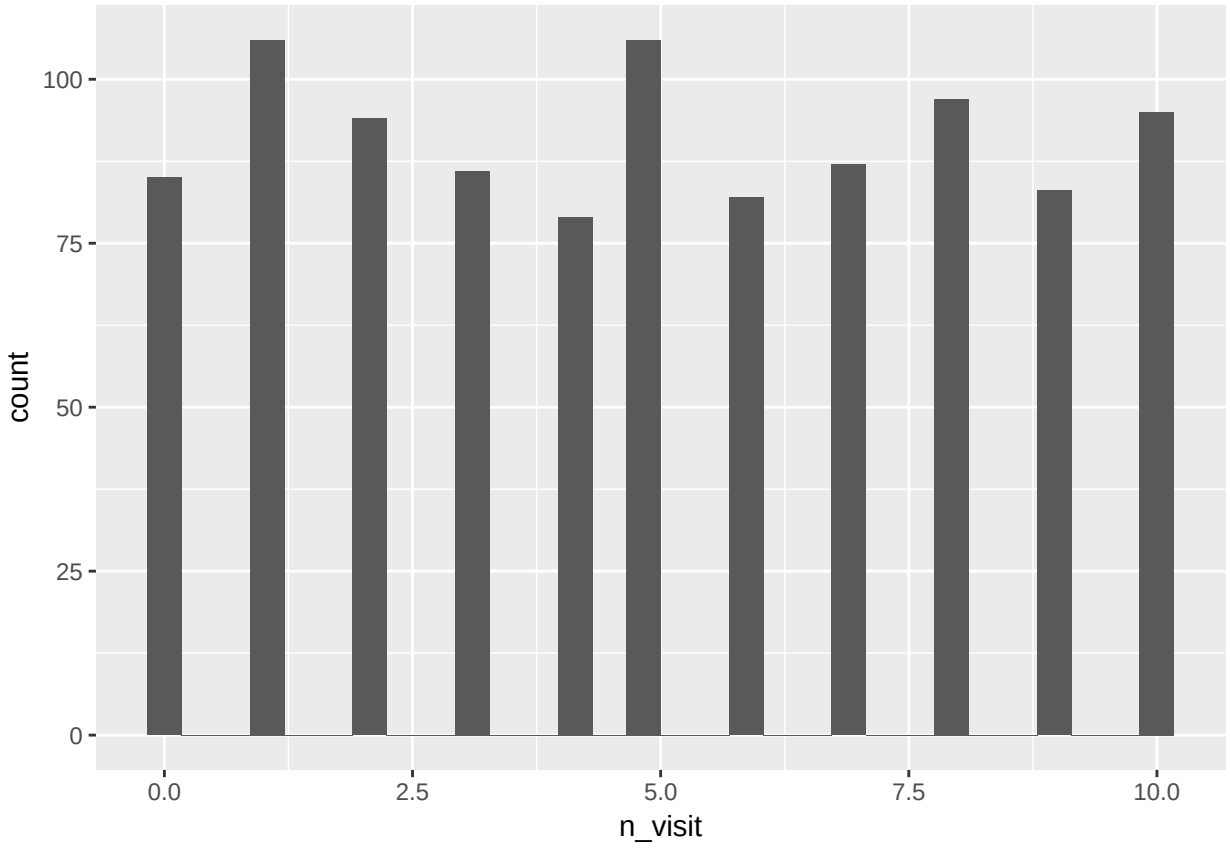
- id: row id
- referrer: referrer website/search engine
- os: operating system
- browser: browser
- device: device used to visit the website
- n_pages: number of pages visited
- duration: time spent on the website (in seconds)
- repeat: frequency of visits
- country: country of origin
- purchase: whether visitor purchased
- order_value: order value of visitor (in dollars)

Histogram

In ggplot2, a histogram is created using `geom_histogram()`. In the below example, we build the histogram of `n_visit` from the `ecom` data.

```
ggplot(ecom) +
  geom_histogram(aes(n_visit))
```

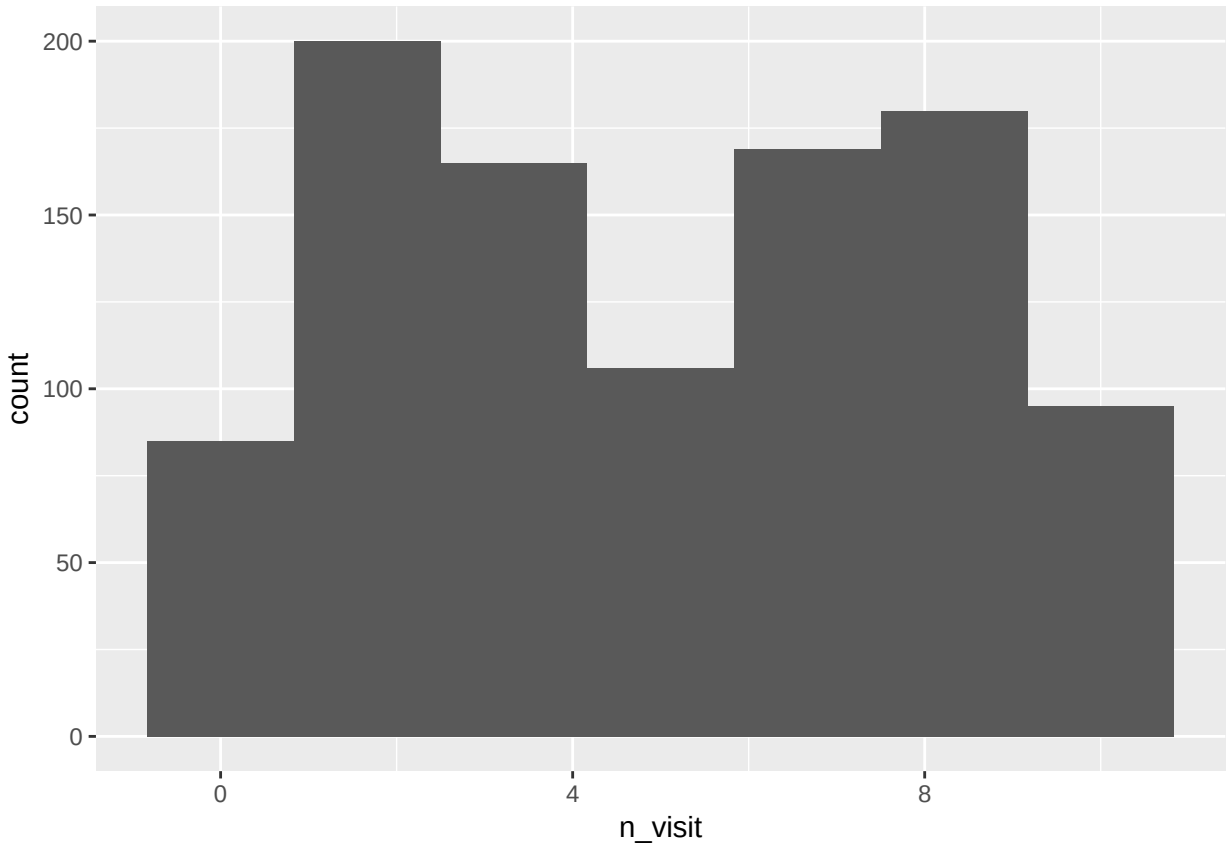
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Specify Bins

The default number of bins used by `geom_histogram()` is 30 and may not be always useful. Let us specify the number of bins using the `bins` argument.

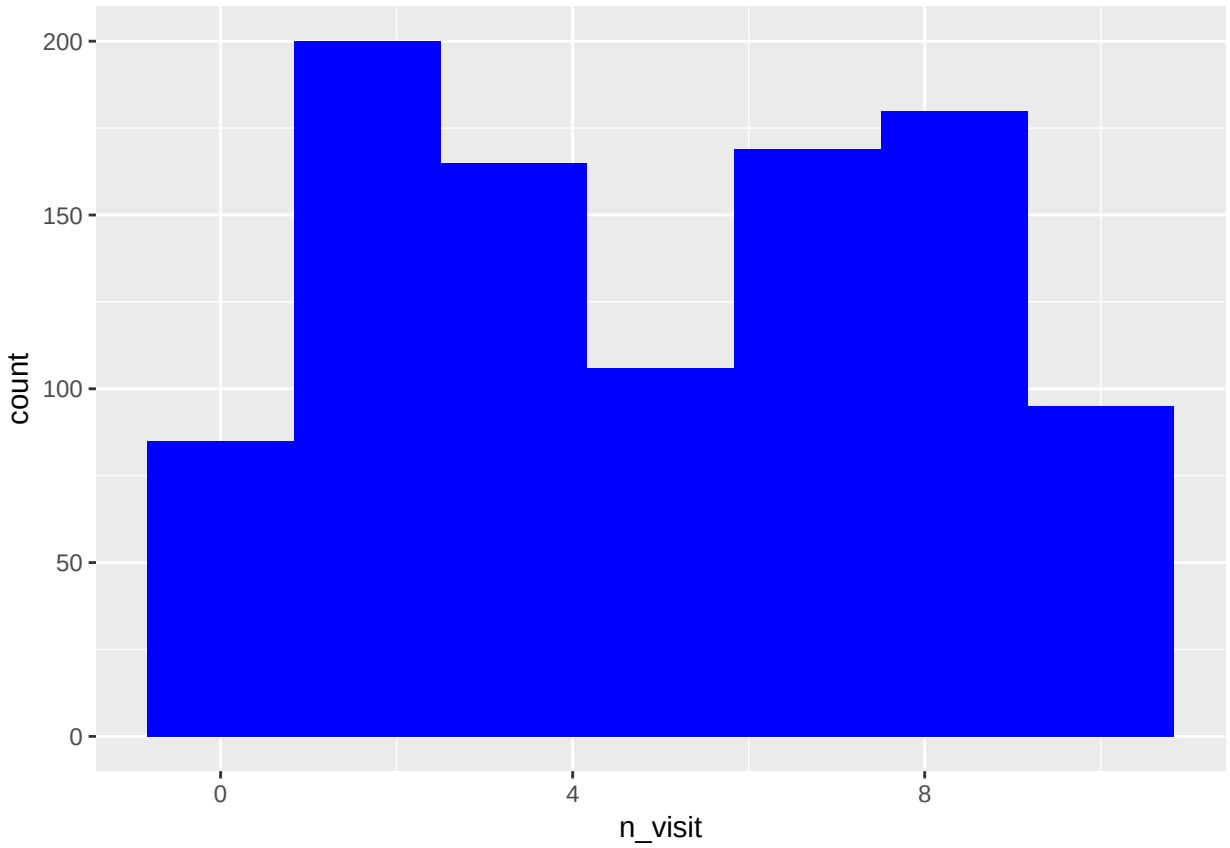
```
ggplot(ecom) +
  geom_histogram(aes(n_visit), bins = 7)
```



Fill

Use the `fill` argument to modify the background color of the histogram. In the below example, we set the background color to blue.

```
ggplot(ecom) +
  geom_histogram(aes(n_visit), bins = 7, fill = 'blue')
```



Summary

In this chapter, we learnt to:

- build histogram
- specify bins
- modify
  - color
  - fill