POLS6382 Methods III: MLE

Ling Zhu

University of Houston

August 27, 2025

Lab 1: Getting Started with R

1. An introduction to R and RStudio

1.1 About R/R Studio

- `R` is a programming language developed by statisticians for data analysis and visualization
- Open-source and free
- RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, tracking data-analysis history, and workspace management.To participate in this workshop, you need to install R and RStudio according to your operating system.
- Installing R
- Installing RStudio

1.2 Additional fascinating advantages of R

- Compatible with LaTeX
- Write dynamic documents with `R Markdown`
- Build your own personal website with `Blogdown`

1.3 A few notes on using R

- Equal sign: <-, =, ==
- Escape sign: \
- Pound sign: #
- Semicolon sign: ;
- Colon sign: :
- Question mark: ?
- Install packages: `install.packages("")`
- Load packages: `library()` or `require()`
- Execute codes: `Run` button or shortcut key ("Command+Enter" on Mac and "Control+Enter" on Windows)
- Fold code chunks: "Command+Option+L"
- Clear the console: "Control+L"
- Clear the environment: `rm(list = ls())`

1.4 About RStudio

RStudio is an integrated developed environment (IDE) for `R`. The interface of RStudio is composed of four panes:

- *Environment/History/Connections*: display all scales, vectors, data sets, and data frames
- *Files/Plots/Packages/Help/View*: present plots and help documentation
- *Console*: show R outputs and messages
- *Script*: the place where you write an R script (and other scripts)

2. Workflow

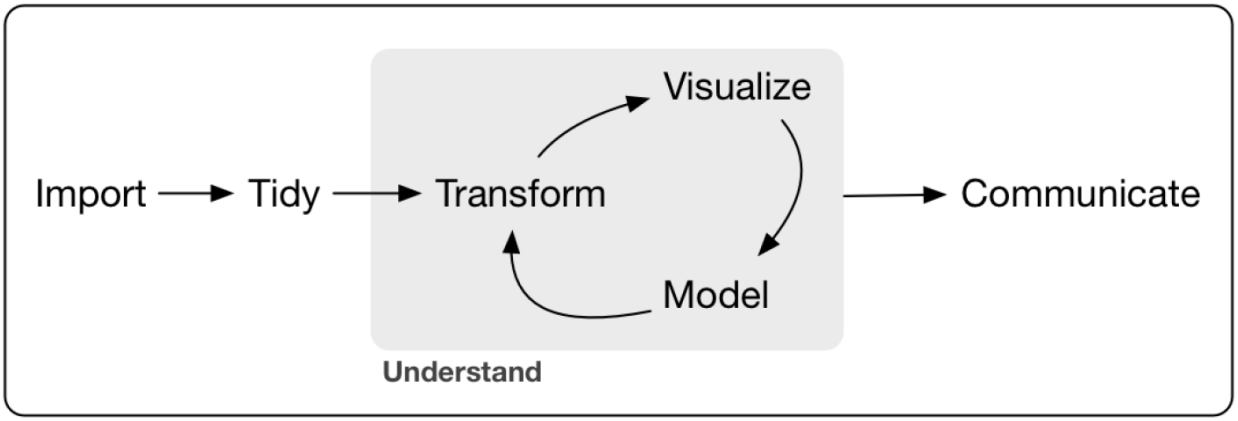


Figure 1: avatar

Figure 1. A typical workflow in R suggested by Wickham and Grolemund (2017)

(1) Set up a folder for your project and put relevant files in that folder
(2) Create or open an R script in RStudio
(3) Set a work directory in accordance with the location of the previous folder using the `setwd()` command (forward sign / on Mac and two backslash signs on Windows \\)
(4) Read in data
(5) Data cleaning, manipulation, and analysis
(6) Save the R script and data file (if necessary)
(7) Run the R script from beginning to the end to make sure all results are replicable.

```
rm(list = ls())
# clear the environment
setwd("/Users/lingzhu/Dropbox/UH Teaching/POLS6382_2025/2025 Labs/Lab 1")
# set work directory
lab1data <- read.csv("Lab 1.csv", stringsAsFactors = FALSE)
# import the data set
lab1data$surveydate <- as.Date(lab1data$surveydate, format = '%m/%d/%y')
# data cleaning
lab1data$female <- lab1data$gender=="female"
lab1data$gender <- as.factor(lab1data$gender)
# data manipulation
lmreg <- lm(lab1data$income ~ lab1data$education + lab1data$age, data = lab1data)
# data analysis
write.csv(lab1data, file = "workflow.csv")
# save the data file
```

3. Data Structure in R

- Scale: a single value
- Vector: a set of values that appear in the form of number, character, date, factor, or boolean values

- Matrix: a two-dimension array composed of numeric vectors with equal number of elements
- List: a mix of different kinds of vector (key-value pairs)
- Data frame: a special list with equal number of elements for each vector

```
value <- 8; value
```

```
## [1] 8
```

```
# this is a scale
```

```
vector1 <- c(1, 2, 3); vector1 # c() refers to the concatenate() function
```

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

```
# this is a numeric vector
```

```
vector2 <- c("apple", "orange", "banana"); vector2
```

```
## [1] "apple"  "orange" "banana"
```

```
# this is a character vector
```

```
matrix <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), nrow = 3,
                 byrow = T, dimnames = list(c("a", "b", "c"),
                                            c("var1", "var2", "var3", "var4"))); matrix
```

```
##   var1 var2 var3 var4
## a    1    2    3    4
## b    5    6    7    8
## c    9   10   11   12
```

```
# this is a matrix
```

```
list <- list(a=c(1,3,5), b=c("China", "Finland", "UK", "US")); list
```

```
## $a
## [1] 1 3 5
##
## $b
## [1] "China"   "Finland" "UK"      "US"
```

```
# this is a list
```

```
dataframe <- as.data.frame(matrix); dataframe
```

```
##   var1 var2 var3 var4
## a    1    2    3    4
## b    5    6    7    8
## c    9   10   11   12
```

```
# this is a dataframe
```

4. Calling Out Elements and Variables

4.1 Subscript within square brackets

- var[i]: select the i_{th} element in var
- var[c(1, 3)]: select the 1^{st} and 3^{rd} element in var
- dataframe[i, j]: select from a dataframe the element in the i_{th} row and j_{th} column
- dataframe[, m:k]: select columns from m to k
- dataframe[c:f,]: select rows from c to f

```r
lab1data$income[10]
```

```
## [1] 60000
```

```r
# show the 10th person's income
lab1data$income[c(5, 7)]
```

```
## [1] 22000 40000
```

```r
# show the 5th and 7th person's income
lab1data[5, 7]
```

```
## [1] 176
```

```r
# show the element corresponding to the 7th column and 5th row
head(lab1data[ ,5:7], 10)
```

```
##    age      union height
## 1   59 used to be    170
## 2   32        yes    176
## 3   67 used to be    156
## 4   61 used to be    170
## 5   42         no    176
## 6   86 used to be    164
## 7   52        yes    158
## 8   81         no    150
## 9   38         no    160
## 10  49        yes    170
```

```r
# show the first 10 rows of age, union, and height
lab1data[105:115, ]
```

```
##      id income education gender age      union height weight surveydate female
## 105 105  40000        11   male  35         no    170    140 2015-10-03  FALSE
## 106 106  60000         7 female  78 used to be    149     96 2015-10-04   TRUE
## 107 107  20000         4 female  49 used to be    158    170 2015-10-01   TRUE
## 108 108  45000         7   male  70 used to be    169    135 2015-10-03  FALSE
## 109 109      0         6   male  52         no    168    156 2015-10-04  FALSE
## 110 110   3000         3   male  39         no    170    175 2015-10-01  FALSE
## 111 111   5000         4   male  28 used to be    175    128 2015-10-02  FALSE
## 112 112      0         4 female  48         no    162    160 2015-08-29   TRUE
## 113 113  30000         4 female  46         no    158    109 2015-10-02   TRUE
## 114 114  80000        10   male  40 used to be    171    134 2015-10-17  FALSE
## 115 115  84000        10   male  34        yes    180    166 2015-10-18  FALSE
```

```r
# show the values of all variables for the 105th to 115th persons
```

4.2 The dollar sign

```r
head(lab1data$income)
```

```
## [1] 30000 48000 23000 50000 22000 48000
```

```r
head(lab1data[ , 2])
```

```
## [1] 30000 48000 23000 50000 22000 48000
```

4.3 The attach() function

```
attach(lab1data)
head(income)
```

```
## [1] 30000 48000 23000 50000 22000 48000
```

5. Generating and Importing Data

5.1 Random sampling

```
head(rnorm(100, mean = 10, sd = 2.5))
```

```
## [1] 11.616905  7.737690  6.532596  9.837589 12.338205  7.447037
```

5.2 Customized data

```
values <- c(1, 3, 5); values
```

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

```
vector1 <- seq(5, by=5, length=20); vector1
```

```
##  [1]   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90  95
## [20] 100
```

```
vector2 <- rep(values, times = 3); vector2
```

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

```
vector3 <- rep(values, c(1, 2, 3)); vector3
```

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

5.3 Import an external dataset

- read.csv()
- read.xlsx(): need to install and load the `openxlsx` package
- read.dta()
- read.dta13(): applicable to dta files generated from `Stata` in a version above 12.0. The `readstata13` package should be installed and loaded before using this command.
- read.table()

```
csvdata <- read.csv("Lab 1.csv")
library(readstata13)
csvdata <- read.dta13("cgss2015.dta")
```

```
## Warning in read.dta13("cgss2015.dta"):
##    Factor codes of type double or float detected in variables
##
##    income
##
##    No labels have been assigned.
##    Set option 'nonint.factors = TRUE' to assign labels anyway.
```

```
library(haven)
csvdata <- read_dta("cgss2015.dta")
```

6. Data Manipulation

6.1 Add variables to a dataframe

- dataframe$newvar: make sure this variable name doesn't exist in the data set, or the original variable will be overwritten by the new variable

```
names(lab1data)
```

```
##  [1] "id"         "income"     "education"  "gender"     "age"
##  [6] "union"      "height"     "weight"     "surveydate" "female"
```

```
# display variable names in the existing dataset, 10 variables in total
lab1data$year <- 2015 ; names(lab1data)
```

```
##  [1] "id"         "income"     "education"  "gender"     "age"
##  [6] "union"      "height"     "weight"     "surveydate" "female"
## [11] "year"
```

```
# year was added to the dataset
```

6.2 Drop variables from a dataframe

- dataframe[, -i]
- dataframe[, -i:-j]

```
otherdata <- lab1data
# create a duplicate of the existing data set, and name it as other data
names(otherdata)
```

```
##  [1] "id"         "income"     "education"  "gender"     "age"
##  [6] "union"      "height"     "weight"     "surveydate" "female"
## [11] "year"
```

```
# show the variable names
otherdata <- otherdata[ , -10]
names(otherdata)
```

```
##  [1] "id"         "income"     "education"  "gender"     "age"
##  [6] "union"      "height"     "weight"     "surveydate" "year"
```

```
# drop the 10th variable (female)
otherdata <- otherdata[ , -6:-9]
names(otherdata)
```

```
## [1] "id"        "income"    "education" "gender"    "age"       "year"
```

```
# drop union, height, weight, and surveydate
```

6.3 Select a subset from the dataframe based on rows (observations)

- dataframe[which,]
- *which can be any condition in relation to observations*

```
union.no <- lab1data[lab1data$union == "no", ]; dim(union.no)
```

```
## [1] 8160   11
```

```
union.yes <- lab1data[lab1data$union == "yes", ]; dim(union.yes)
```

```
## [1] 965  11
```

6.4 Select a subset from the dataframe based on columns (variables)

- newdata <- subset(olddata, select = c(var1, var2, ...))
- newdata <- subset(olddata, select = - c(var1, var2, ...))

```
data1 <- subset(lab1data, select = c(id, income, education, gender)); head(data1)
```

```
##   id income education gender
## 1  1  30000         6   male
## 2  2  48000        12   male
## 3  3  23000         3 female
## 4  4  50000        12 female
## 5  5  22000         4   male
## 6  6  48000         3   male
```

```
data2 <- subset(lab1data, select = c(id, age, union)); head(data2)
```

```
##   id age      union
## 1  1  59 used to be
## 2  2  32        yes
## 3  3  67 used to be
## 4  4  61 used to be
## 5  5  42         no
## 6  6  86 used to be
```

```
data3 <- subset(lab1data, select = - c(age, union)); head(data3)
```

```
##   id income education gender height weight surveydate female year
## 1  1  30000         6   male    170    160 2015-10-04  FALSE 2015
## 2  2  48000        12   male    176    110 2015-10-02  FALSE 2015
## 3  3  23000         3 female    156    129 2015-10-17   TRUE 2015
## 4  4  50000        12 female    170    146 2015-10-06   TRUE 2015
## 5  5  22000         4   male    176    171 2015-09-07  FALSE 2015
## 6  6  48000         3   male    164    124 2015-09-14  FALSE 2015
```

6.5 Merge two datasets

- mergeddata <- merge(dataframe1, dataframe2, by = "id", all.x = T, all.y = T)
- mergeddata <- merge(dataframe1, dataframe2, by = c("id", "year"), all = T)

```
mergeddata <- merge(data1, data2, by = "id", all.x = T, all.y = T); dim(mergeddata)
```

```
## [1] 10242     6
```

```
# the mergeddata is totally the same as the "sum" of data1 and data2 (i.e. lab1data)
```

6.6 Append one dataset to the other

- appendeddata <- rbind(union1, union2)
- *Note that the two data frames should have exactly the same variables*

```
appendeddata <- rbind(union.no, union.yes); dim(appendeddata)
```

```
## [1] 9125   11
```

6.7 Reshape the data structure from wide to long

- library(reshape2)
- newdataframe <- melt(dataframe, id = c("var1", "var2", ...))

```
wide.grades <- as.data.frame(matrix(round(rnorm(12, 80, 5), 0), nrow = 4, byrow = T,
                                    dimnames = list(1:4, c("Chinese", "Math", "English"))))
wide.grades$Name <- c("Zhao", "Qian", "Sun", "Li")
head(wide.grades)
```

```
##   Chinese Math English Name
## 1      80   81      83 Zhao
## 2      76   79      77 Qian
## 3      80   74      73  Sun
## 4      79   82      67   Li
```

```
library(reshape2)
long.grades <- melt(wide.grades, id = c("Name")); long.grades
```

```
##    Name variable value
## 1  Zhao  Chinese    80
## 2  Qian  Chinese    76
## 3   Sun  Chinese    80
## 4    Li  Chinese    79
## 5  Zhao     Math    81
## 6  Qian     Math    79
## 7   Sun     Math    74
## 8    Li     Math    82
## 9  Zhao  English    83
## 10 Qian  English    77
## 11  Sun  English    73
## 12   Li  English    67
```

6.8 Rename variables

- names(dataframe)[names(dataframe) == "oldname"] <- "newname"

```
renamedata <- lab1data
names(renamedata)
```

```
##  [1] "id"         "income"     "education"  "gender"     "age"
##  [6] "union"      "height"     "weight"     "surveydate" "female"
## [11] "year"
```

```
names(renamedata)[names(renamedata) == "id"] <- "identity"
names(renamedata)
```

```
##  [1] "identity"   "income"     "education"  "gender"     "age"
##  [6] "union"      "height"     "weight"     "surveydate" "female"
## [11] "year"
```

7. Linear Regression Analysis

7.1 Running a linear regression

- model <- lm(y~x1+x2+x3, data = dataset): the basic command for running a multivariate linear regression
- model$coef: check out the regression coefficients
- summary(model): display the regression results
- stargazer(model): output the model results in a professional way
- plot(model): visualize model results and post-estimation diagnosis

```r
lab1data$logincome <- log((lab1data$income+1), base = 10)
linear.model <- lm(logincome ~ education+gender+age+I(age^2)+as.factor(union), data=lab1data)
linear.model$coef
```

```
##                (Intercept)                  education
##               0.4933134079               0.0915235417
##                 gendermale                        age
##               0.6237411500               0.0924578166
##                   I(age^2) as.factor(union)used to be
##              -0.0008735615               0.7281852491
##       as.factor(union)yes
##               0.6530310744
```

```r
# you can also check out residuals, fitted values, degrees of freedom, etc.
```

7.2 Outputing the regression results

```r
summary(linear.model)
```

```
##
## Call:
## lm(formula = logincome ~ education + gender + age + I(age^2) +
##     as.factor(union), data = lab1data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.2977 -0.1647  0.4574  0.9354  4.1323
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  4.933e-01  1.401e-01   3.522  0.00043 ***
## education                    9.152e-02  5.942e-03  15.404  < 2e-16 ***
## gendermale                   6.237e-01  3.010e-02  20.723  < 2e-16 ***
## age                          9.246e-02  5.137e-03  17.999  < 2e-16 ***
## I(age^2)                    -8.736e-04  4.861e-05 -17.971  < 2e-16 ***
## as.factor(union)used to be   7.282e-01  5.215e-02  13.964  < 2e-16 ***
## as.factor(union)yes          6.530e-01  5.486e-02  11.903  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.508 on 10235 degrees of freedom
## Multiple R-squared:  0.1419, Adjusted R-squared:  0.1414
## F-statistic:   282 on 6 and 10235 DF,  p-value: < 2.2e-16
```

```r
library(stargazer)
```

```
##
## Please cite as:
```

```
##  Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Tables.
```

```
##  R package version 5.2.3. https://CRAN.R-project.org/package=stargazer
```

```r
stargazer(linear.model, type = "text", title = "Table 1. The linear regression results")
```

```
##
## Table 1. The linear regression results
```

```
## ================================================
##                              Dependent variable:
##                            ----------------------
##                                    logincome
## ------------------------------------------------
## education                           0.092***
##                                      (0.006)
##
## gendermale                          0.624***
##                                      (0.030)
##
## age                                 0.092***
##                                      (0.005)
##
## I(age2)                             -0.001***
##                                     (0.00005)
##
## as.factor(union)used to be          0.728***
##                                      (0.052)
##
## as.factor(union)yes                 0.653***
##                                      (0.055)
##
## Constant                            0.493***
##                                      (0.140)
##
## ------------------------------------------------
## Observations                         10,242
## R2                                    0.142
## Adjusted R2                           0.141
## Residual Std. Error             1.508 (df = 10235)
## F Statistic                282.008*** (df = 6; 10235)
## ================================================
## Note:                      *p<0.1; **p<0.05; ***p<0.01
```

8. The Tidyverse Package

R is a tool developed by statisticians in the 1990s to facilitate statistical analysis. Now it has become increasingly popular not only in academia but also in the business sector. With a particular focus on the tidyverse, this session aims to introduce h some basic data wrangling and visualization skills in R.

As a programming language, R has a number of built-in packages and functions. In addition, there are many packages and functions developed by third parties to make R more useful and powerful in various contexts. Tidyverse is just one of them. It is a collection of R packages developed principally by Hadley Wickham and his collaborators. These packages have made R an especially powerful tool for data wrangling, analysis, modeling, and visualization.

What does tidyverse include?

```
library(tidyverse)
tidyverse_packages()
```

```
##  [1] "broom"         "conflicted"    "cli"           "dbplyr"
##  [5] "dplyr"         "dtplyr"        "forcats"       "ggplot2"
##  [9] "googledrive"   "googlesheets4" "haven"         "hms"
## [13] "httr"          "jsonlite"      "lubridate"     "magrittr"
```

```
## [17] "modelr"        "pillar"        "purrr"         "ragg"
## [21] "readr"         "readxl"        "reprex"        "rlang"
## [25] "rstudioapi"    "rvest"         "stringr"       "tibble"
## [29] "tidyr"         "xml2"          "tidyverse"
```

Although there are many packages in the tidyverse family, some are more popular and useful than others. The core packages we are going to learn in this course include

- tibble
- readr
- tidyr
- dplyr
- ggplot2

8.1 Tidying Data with tidyr: Separating and Uniting variables

```
# Loading a dataset from the AER package
# You can use data(package=.packages(all.available=TRUE)) to check out all available datasets
data(Fatalities, package="AER")
Fatalities <- as_tibble(Fatalities)
Fatalities %>% glimpse()
```

```
## Rows: 336
## Columns: 34
## $ state        <fct> al, al, al, al, al, al, al, az, az, az, az, az, az, az, a~
## $ year         <fct> 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1982, 1983, 198~
## $ spirits      <dbl> 1.37, 1.36, 1.32, 1.28, 1.23, 1.18, 1.17, 1.97, 1.90, 2.1~
## $ unemp        <dbl> 14.4, 13.7, 11.1, 8.9, 9.8, 7.8, 7.2, 9.9, 9.1, 5.0, 6.5,~
## $ income       <dbl> 10544.15, 10732.80, 11108.79, 11332.63, 11661.51, 11944.0~
## $ emppop       <dbl> 50.69204, 52.14703, 54.16809, 55.27114, 56.51450, 57.5098~
## $ beertax      <dbl> 1.53937948, 1.78899074, 1.71428561, 1.65254235, 1.6099070~
## $ baptist      <dbl> 30.3557, 30.3336, 30.3115, 30.2895, 30.2674, 30.2453, 30.~
## $ mormon       <dbl> 0.32829, 0.34341, 0.35924, 0.37579, 0.39311, 0.41123, 0.4~
## $ drinkage     <dbl> 19.00, 19.00, 19.00, 19.67, 21.00, 21.00, 21.00, 19.00, 1~
## $ dry          <dbl> 25.0063, 22.9942, 24.0426, 23.6339, 23.4647, 23.7924, 23.~
## $ youngdrivers <dbl> 0.211572, 0.210768, 0.211484, 0.211140, 0.213400, 0.21552~
## $ miles        <dbl> 7233.887, 7836.348, 8262.990, 8726.917, 8952.854, 9166.30~
## $ breath       <fct> no, no, no, no, no, no, no, no, no, no, no, no, no, no, n~
## $ jail         <fct> no, no, no, no, no, no, no, yes, yes, yes, yes, yes, yes,~
## $ service      <fct> no, no, no, no, no, no, no, yes, yes, yes, yes, yes, yes,~
## $ fatal        <int> 839, 930, 932, 882, 1081, 1110, 1023, 724, 675, 869, 893,~
## $ nfatal       <int> 146, 154, 165, 146, 172, 181, 139, 131, 112, 149, 150, 17~
## $ sfatal       <int> 99, 98, 94, 98, 119, 114, 89, 76, 60, 81, 75, 85, 87, 67,~
## $ fatal1517    <int> 53, 71, 49, 66, 82, 94, 66, 40, 40, 51, 48, 72, 50, 54, 3~
## $ nfatal1517   <int> 9, 8, 7, 9, 10, 11, 8, 7, 7, 8, 11, 19, 16, 14, 5, 2, 2, ~
## $ fatal1820    <int> 99, 108, 103, 100, 120, 127, 105, 81, 83, 118, 100, 104, ~
## $ nfatal1820   <int> 34, 26, 25, 23, 23, 31, 24, 16, 19, 34, 26, 30, 25, 14, 2~
## $ fatal2124    <int> 120, 124, 118, 114, 119, 138, 123, 96, 80, 123, 121, 130,~
## $ nfatal2124   <int> 32, 35, 34, 45, 29, 30, 25, 36, 17, 33, 30, 25, 34, 31, 1~
## $ afatal       <dbl> 309.438, 341.834, 304.872, 276.742, 360.716, 368.421, 298~
## $ pop          <dbl> 3942002, 3960008, 3988992, 4021008, 4049994, 4082999, 410~
## $ pop1517      <dbl> 208999.6, 202000.1, 197000.0, 194999.7, 203999.9, 204999.~
## $ pop1820      <dbl> 221553.4, 219125.5, 216724.1, 214349.0, 212000.0, 208998.~
## $ pop2124      <dbl> 290000.1, 290000.2, 288000.2, 284000.3, 263000.3, 258999.~
## $ milestot     <dbl> 28516, 31032, 32961, 35091, 36259, 37426, 39684, 19729, 1~
```

```
## $ unempus      <dbl> 9.7, 9.6, 7.5, 7.2, 7.0, 6.2, 5.5, 9.7, 9.6, 7.5, 7.2, 7.~
## $ emppopus     <dbl> 57.8, 57.9, 59.5, 60.1, 60.7, 61.5, 62.3, 57.8, 57.9, 59.~
## $ gsp          <dbl> -0.022124760, 0.046558253, 0.062797837, 0.027489973, 0.03~
```

If we want to split the year variable into two variables: century and year, we can use the **separate()** function.

```
Fatalities %>%
  separate(year, into=c("century", "year.new"), sep=2) %>%
  head()
```

```
## # A tibble: 6 x 35
##   state century year.new spirits unemp income emppop beertax baptist mormon
##   <fct> <chr>   <chr>      <dbl> <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl>
## 1 al    19      82          1.37 14.4  10544.   50.7    1.54    30.4  0.328
## 2 al    19      83          1.36 13.7  10733.   52.1    1.79    30.3  0.343
## 3 al    19      84          1.32 11.1  11109.   54.2    1.71    30.3  0.359
## 4 al    19      85          1.28  8.90 11333.   55.3    1.65    30.3  0.376
## 5 al    19      86          1.23  9.80 11662.   56.5    1.61    30.3  0.393
## 6 al    19      87          1.18  7.80 11944    57.5    1.56    30.2  0.411
## # i 25 more variables: drinkage <dbl>, dry <dbl>, youngdrivers <dbl>,
## #   miles <dbl>, breath <fct>, jail <fct>, service <fct>, fatal <int>,
## #   nfatal <int>, sfatal <int>, fatal1517 <int>, nfatal1517 <int>,
## #   fatal1820 <int>, nfatal1820 <int>, fatal2124 <int>, nfatal2124 <int>,
## #   afatal <dbl>, pop <dbl>, pop1517 <dbl>, pop1820 <dbl>, pop2124 <dbl>,
## #   milestot <dbl>, unempus <dbl>, emppopus <dbl>, gsp <dbl>
```

In other cases, we may want to combine multiple columns into one. For instance, we may have one column called city and another column called state. It might be a good idea to organize the two pieces of information in this way: Saint Louis, MO. The **unite()** function can be helpful for this purpose.

Suppose we want to combine *state* and *year* in the **Fatalities** data set.

```
Fatalities %>% unite(StateYear, c(state, year), sep="-") %>% head()
```

```
## # A tibble: 6 x 33
##   StateYear spirits unemp income emppop beertax baptist mormon drinkage   dry
##   <chr>       <dbl> <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl>    <dbl> <dbl>
## 1 al-1982      1.37 14.4  10544.   50.7    1.54    30.4  0.328     19    25.0
## 2 al-1983      1.36 13.7  10733.   52.1    1.79    30.3  0.343     19    23.0
## 3 al-1984      1.32 11.1  11109.   54.2    1.71    30.3  0.359     19    24.0
## 4 al-1985      1.28  8.90 11333.   55.3    1.65    30.3  0.376     19.7  23.6
## 5 al-1986      1.23  9.80 11662.   56.5    1.61    30.3  0.393     21    23.5
## 6 al-1987      1.18  7.80 11944    57.5    1.56    30.2  0.411     21    23.8
## # i 23 more variables: youngdrivers <dbl>, miles <dbl>, breath <fct>,
## #   jail <fct>, service <fct>, fatal <int>, nfatal <int>, sfatal <int>,
## #   fatal1517 <int>, nfatal1517 <int>, fatal1820 <int>, nfatal1820 <int>,
## #   fatal2124 <int>, nfatal2124 <int>, afatal <dbl>, pop <dbl>, pop1517 <dbl>,
## #   pop1820 <dbl>, pop2124 <dbl>, milestot <dbl>, unempus <dbl>,
## #   emppopus <dbl>, gsp <dbl>
```

8.2 Transforming the data structure

Data can be stored in various formats, with wide and long formats being the most common in quantitative analysis. We can convert the two structures using **stack()** and **unstack()** functions in base R. But in tidyverse, we use the pivot functions (**pivot_longer()** and **pivot_wider()**) for that task.

8.2.1 Pivot_longer() This function is equivalent to the **stack()** function in base R. Suppose we have a small data set as follows.

```
gdp <- tibble(country=c("US", "Canada", "Mexico"),
              year2021=c(23, 2, 1),
              year2020=c(20, 1.9, 0.8),
              year2019=c(19, 1.8, 0.75))
head(gdp)
```

```
## # A tibble: 3 x 4
##   country year2021 year2020 year2019
##   <chr>      <dbl>    <dbl>    <dbl>
## 1 US            23     20      19
## 2 Canada         2      1.9     1.8
## 3 Mexico         1      0.8     0.75
```

If we want to stack the three columns relating to year, we can write code as follows.

```
gdp_longer <- gdp %>%
  pivot_longer(c(year2021, year2020, year2019), names_to="year", values_to="gdp")
gdp_longer
```

```
## # A tibble: 9 x 3
##   country year       gdp
##   <chr>   <chr>    <dbl>
## 1 US      year2021 23
## 2 US      year2020 20
## 3 US      year2019 19
## 4 Canada  year2021  2
## 5 Canada  year2020  1.9
## 6 Canada  year2019  1.8
## 7 Mexico  year2021  1
## 8 Mexico  year2020  0.8
## 9 Mexico  year2019  0.75
```

8.2.2 Pivot_wider() **pivot_wider()** is the opposite of **pivot_longer()**. This function can be helpful if you want to make sure each unique observation is represented by a single row.

```
gdp_longer %>% pivot_wider(names_from="year", values_from="gdp")
```

```
## # A tibble: 3 x 4
##   country year2021 year2020 year2019
##   <chr>      <dbl>    <dbl>    <dbl>
## 1 US            23     20      19
## 2 Canada         2      1.9     1.8
## 3 Mexico         1      0.8     0.75
```

If you use the built-in **unstack()** function, the **country** variable will be omitted.

```
unstack(gdp_longer, gdp~year)
```

```
##   year2019 year2020 year2021
## 1    19.00     20.0       23
## 2     1.80      1.9        2
## 3     0.75      0.8        1
```

8.3. Manipulating Your Data with dplyr

The *dplyr* is perhaps one of the most popular packages in tidyverse. It is efficient in transforming your data through filtering, selecting, sorting, ordering, renaming and summarizing. Additionally, it also allows you to add new variables and group observations. In short, the *dplyr* is all you need for data wrangling in R.

- filter(): Pick observations by their values
- arrange(): Sort the dataset by reordering the rows
- relocate(): Change the order of variables
- select(): Pick or drop variables by their names
- rename(): Change variable names
- mutate(): Create new variables
- case_when(): Create new variables based on other variables
- group_by(): Transform the dataset into group-based structure and changes the scope of each function from operating on the entire dataset to operating on it group-by-group
- summarise(): Collapse many values down to a single summary

```
# A quick look at some rows in the data file
Fatalities %>% filter(year==1982) %>% glimpse()
```

```
## Rows: 48
## Columns: 34
## $ state        <fct> al, az, ar, ca, co, ct, de, fl, ga, id, il, in, ia, ks, k~
## $ year         <fct> 1982, 1982, 1982, 1982, 1982, 1982, 1982, 1982, 1982, 198~
## $ spirits      <dbl> 1.37, 1.97, 1.19, 2.21, 2.25, 2.42, 2.59, 2.51, 1.94, 1.3~
## $ unemp        <dbl> 14.4, 9.9, 9.8, 9.9, 7.7, 6.9, 8.5, 8.2, 7.8, 9.8, 11.3, ~
## $ income       <dbl> 10544.152, 12309.069, 10267.303, 15797.136, 15082.339, 17~
## $ emppop       <dbl> 50.69204, 56.89330, 54.47586, 59.51593, 64.96674, 61.7294~
## $ beertax      <dbl> 1.5393795, 0.2147971, 0.6503580, 0.1073986, 0.2147971, 0.~
## $ baptist      <dbl> 30.35570, 3.95890, 22.96720, 1.72310, 2.30000, 0.10000, 0~
## $ mormon       <dbl> 0.328290, 4.919100, 0.328290, 1.678540, 1.823010, 0.23331~
## $ drinkage     <dbl> 19.0, 19.0, 21.0, 21.0, 21.0, 18.5, 20.0, 19.0, 19.0, 19.~
## $ dry          <dbl> 25.006300, 0.000000, 36.712799, 0.000000, 0.113151, 0.227~
## $ youngdrivers <dbl> 0.211572, 0.209012, 0.204903, 0.190196, 0.229148, 0.19284~
## $ miles        <dbl> 7233.887, 6810.157, 7208.500, 6858.677, 7742.842, 6440.05~
## $ breath       <fct> no, no, no, no, no, no, yes, yes, no, no, no, yes, no, no~
## $ jail         <fct> no, yes, no, no, no, no, no, no, no, no, no, no, no, yes,~
## $ service      <fct> no, yes, no, no, yes, no, no, yes, no, no, no, no, no, ye~
## $ fatal        <int> 839, 724, 550, 4615, 668, 515, 122, 2653, 1229, 256, 1651~
## $ nfatal       <int> 146, 131, 102, 944, 140, 158, 34, 587, 225, 47, 418, 235,~
## $ sfatal       <int> 99, 76, 64, 553, 96, 90, 23, 282, 132, 24, 229, 131, 61, ~
## $ fatal1517    <int> 53, 40, 36, 241, 27, 39, 5, 148, 90, 29, 106, 65, 52, 30,~
## $ nfatal1517   <int> 9, 7, 5, 61, 5, 8, 2, 35, 16, 4, 23, 21, 6, 7, 14, 15, 2,~
## $ fatal1820    <int> 99, 81, 62, 567, 91, 86, 24, 259, 131, 38, 203, 127, 59, ~
## $ nfatal1820   <int> 34, 16, 24, 161, 36, 35, 9, 86, 41, 10, 80, 45, 19, 18, 3~
## $ fatal2124    <int> 120, 96, 61, 758, 100, 96, 23, 350, 161, 32, 224, 141, 66~
## $ nfatal2124   <int> 32, 36, 18, 249, 35, 42, 14, 123, 59, 13, 98, 53, 21, 24,~
## $ afatal       <dbl> 309.438, 173.668, 271.459, 1379.130, 219.750, 195.110, 45~
## $ pop          <dbl> 3942002.2, 2896996.5, 2306998.5, 24785976.0, 3071998.8, 3~
## $ pop1517      <dbl> 208999.59, 140999.98, 121999.99, 1157001.75, 143000.14, 1~
## $ pop1820      <dbl> 221553.44, 156378.70, 121269.50, 1321004.38, 169956.84, 1~
## $ pop2124      <dbl> 290000.06, 217999.98, 157000.02, 1892998.12, 246000.05, 2~
## $ milestot     <dbl> 28516, 19729, 16630, 169999, 23786, 20138, 4591, 79498, 4~
## $ unempus      <dbl> 9.7, 9.7, 9.7, 9.7, 9.7, 9.7, 9.7, 9.7, 9.7, 9.7, 9.7, 9.~
## $ emppopus     <dbl> 57.8, 57.8, 57.8, 57.8, 57.8, 57.8, 57.8, 57.8, 57.8, 57.~
## $ gsp          <dbl> -0.022124760, -0.043181900, -0.034733824, -0.011686022, 0~
```

```
# With logical operators
Fatalities %>% filter(year==1982 | year==1983 | year==1984) %>% nrow()
```

```
## [1] 144
```

```
# The "in" operator
Fatalities %>% filter(year %in% c(1982,1984)) %>% nrow()
```

```
## [1] 96
```

What if you want to sort in order or year and state?

```
Fatalities %>% arrange(desc(year), desc(state)) %>% head()
```

```
## # A tibble: 6 x 34
##   state year  spirits unemp income emppop beertax baptist mormon drinkage   dry
##   <fct> <fct>   <dbl> <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl>    <dbl> <dbl>
## 1 wy    1988    1.55   6.30 13098.   64.6  0.0433   3.01   8.51      19.5 0
## 2 wi    1988    1.65   4.30 14941.   67.4  0.140    0.370  0.370     21   1.86
## 3 wv    1988    0.790  9.90 11295.   46.2  0.384    1.70   0.328     21   0
## 4 wa    1988    1.49   6.20 15855.   62.5  0.194    1.31   3.19      21   0
## 5 va    1988    1.33   3.90 17012.   66.5  0.612   12.1    0.610     21   0
## 6 vt    1988    1.82   2.80 14728.   69.0  0.574    0.127  0.232     21   0.336
## # i 23 more variables: youngdrivers <dbl>, miles <dbl>, breath <fct>,
## #   jail <fct>, service <fct>, fatal <int>, nfatal <int>, sfatal <int>,
## #   fatal1517 <int>, nfatal1517 <int>, fatal1820 <int>, nfatal1820 <int>,
## #   fatal2124 <int>, nfatal2124 <int>, afatal <dbl>, pop <dbl>, pop1517 <dbl>,
## #   pop1820 <dbl>, pop2124 <dbl>, milestot <dbl>, unempus <dbl>,
## #   emppopus <dbl>, gsp <dbl>
```

Keep in mind that when you sort data, missing values are always sorted at the end.

We can also use select(), group_by() and summarize() to explore the data structure.

```
Fatalities%>%
select(state,year, income) %>%
group_by(year) %>%
summarize(count = n(),mean_income = mean(income, na.rm=TRUE), sd=sd(income)) %>%
filter(year!="1980")
```

```
## # A tibble: 7 x 4
##   year  count mean_income    sd
##   <fct> <int>       <dbl> <dbl>
## 1 1982     48      12998. 1767.
## 2 1983     48      13108. 1853.
## 3 1984     48      13583. 1980.
## 4 1985     48      13843. 2117.
## 5 1986     48      14186. 2274.
## 6 1987     48      14550. 2460.
## 7 1988     48      14894. 2628.
```

9. Data Visualization Using ggplot2

9.1 Basics of package ggplot2

In this section, we learn how to use ggplot2 to create various plots including histograms, box plots, scatter plots, bar plots, dot plots, pie charts, line plots, and maps. To learn ggplot2, you must get familiar with several key concepts first.

- **Plotting aesthetics**: according to R4DS, "an aesthetic is a visual property of the objects in your plot. Aesthetics include things like the size, the shape, or the color of your points." Each aesthetic property can represent a variable. For example, you can use two different colors red and blue to distinguish the Republican Party from the Democratic Party in a single plot. In that case, we say the party variable is **mapped** to the plot via color.
  - size
  - color
  - shape (point, line)
  - width (line)
- **Geometric objects**: given the same set of data, one can create different plots. For instance, you can create a histogram to visualize student heights. Additionally, you may also choose a box plot. The difference arises from the geometric object you choose.
  - geom_histogram()
  - geom_point()
  - geom_bar()
  - geom_line()
  - geom_boxplot()
  - geom_dotplot()
  - geom_segment()
  - geom_text()
  - geom_polygon()
  - ...
- **Layered plotting**: the plotting in ggplot2 is implemented based on a layered grammar of graphics. That means after creating a basic plot, you can continue to add more layers to that plot by using a plus sign.
  - For instance, after creating a point plot of Y against year, you may continue to add one more layer by using a **geom_line()** to connect those points.
  - After creating a dotplot with the **geom_dotplot()** function, you may continue to use the **geom_segment()** function to add a second layer to make the plot look better.
- If you want to fine-tune your plot, such as adjusting the background, changing label size or editing plot legends, you can do this through the **theme()** function. To facilitate plotting, the **ggplot2** package has already provided quite a few built-in themes.
  - theme_classic()
  - theme_bw()
  - theme_grey()
  - theme_gray()
  - theme_dark()
  - theme_light()
  - theme_minimal()
  - theme_void()
  - ...
- If the built-in themes and the **theme()** function fall short of your expectation, you can change lots of plotting properties such as x- and y-scales as well as colors and coordinates through additional functions such as **labs()**, **scale_x_continuous()**, **scale_fill_manual()** and **coord_flip()**. Again, these functions are added to your plotting code via a plus sign.

Below is a code template.

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

```
#      stat = <STAT>,
#      position = <POSITION>) +
#   <COORDINATE_FUNCTION> +
#   <FACET_FUNCTION>
```

9.2 Histograms

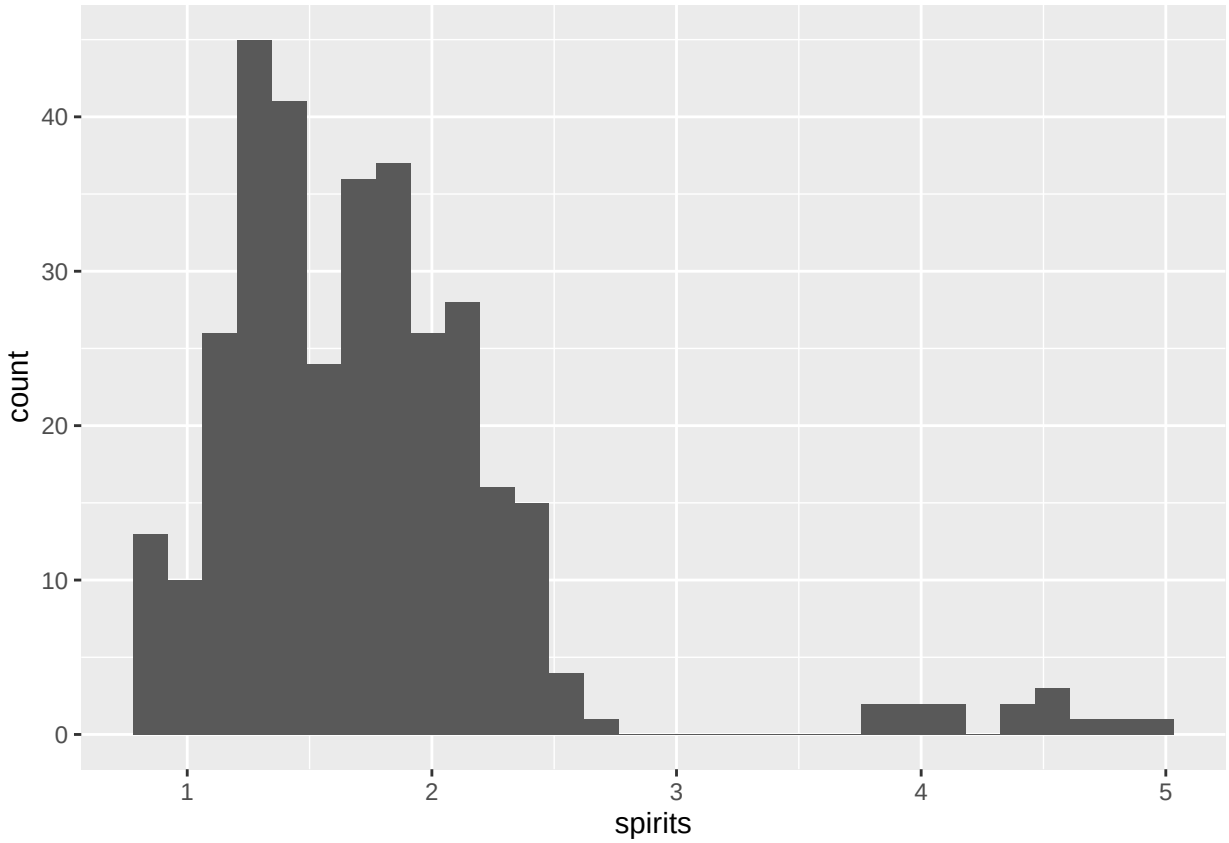
This unit utilizes the **Fatalities** data set from the AER package. Load the data set first.

```
# Loading the Fatalities dataset
data(Fatalities, package="AER")
```

Let's create a histogram for the **spirits** variable first.

```
Fatalities %>% ggplot(mapping=aes(x=spirits)) + geom_histogram()
```

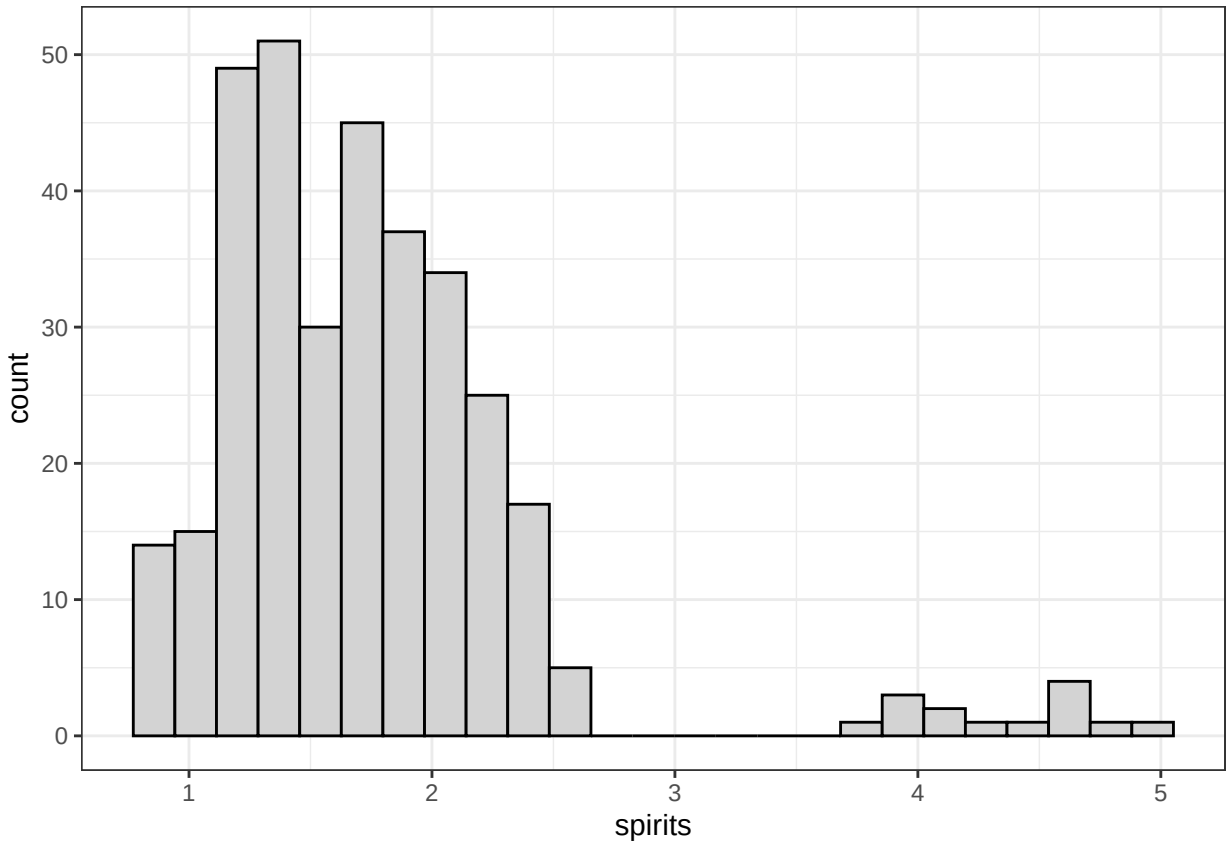
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



When you create a histogram with ggplot2, there are many arguments that have been optimized unless you specify them directly. For instance, the default number of bins is 30, and the default fill color of each bin is black. We can change those plotting styles by specifying arguments in the **geom_histogram()** function.

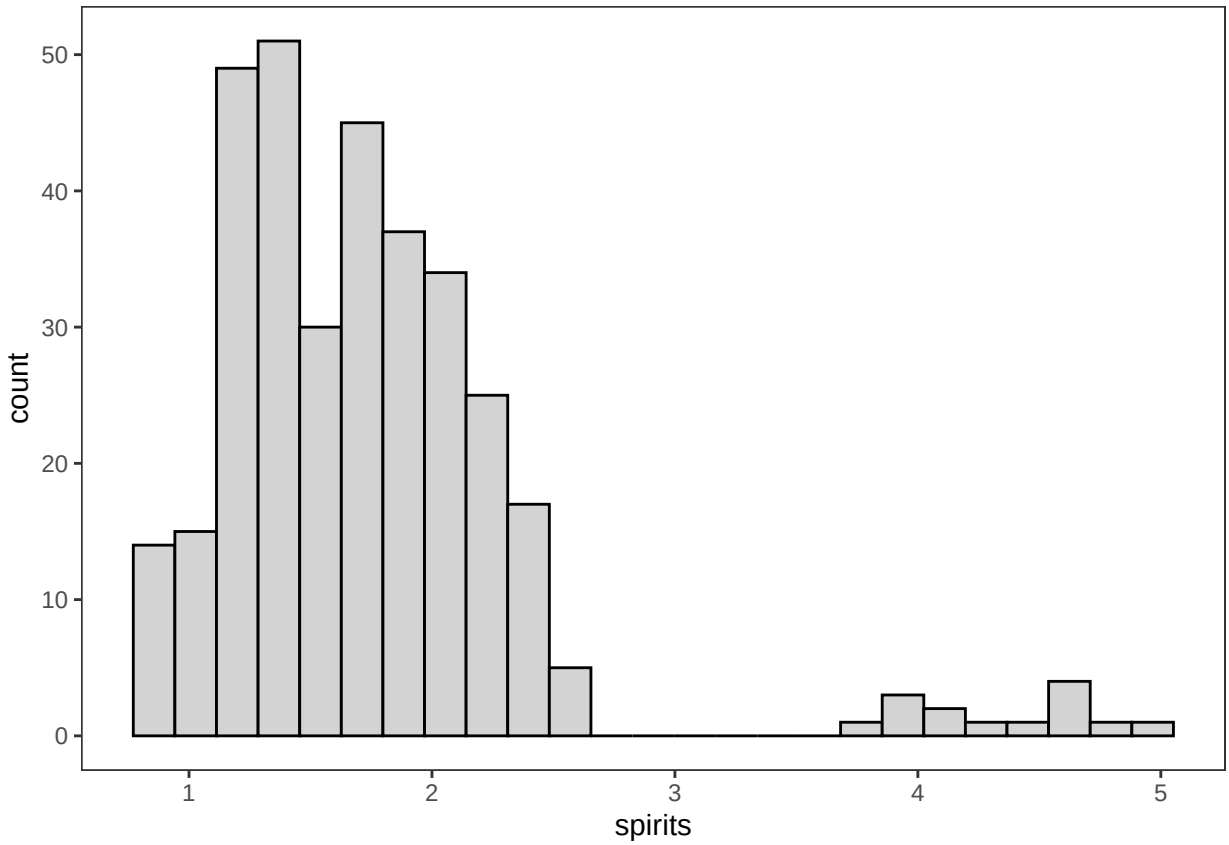
Additionally, we can also pick a theme to make the plot look a bit better.

```
Fatalities %>% ggplot() +
  geom_histogram(aes(x=spirits), bins=25, col="black", fill="lightgrey") +
  theme_bw()
```



What if you want to change the grid lines in the main plotting area? That involves the setting of the **theme()** function.

```
Fatalities %>% ggplot() +
  geom_histogram(aes(x=spirits), bins=25, col="black", fill="lightgrey")+
  theme_bw()+
  theme(panel.grid.minor=element_blank(), panel.grid.major=element_blank())
```

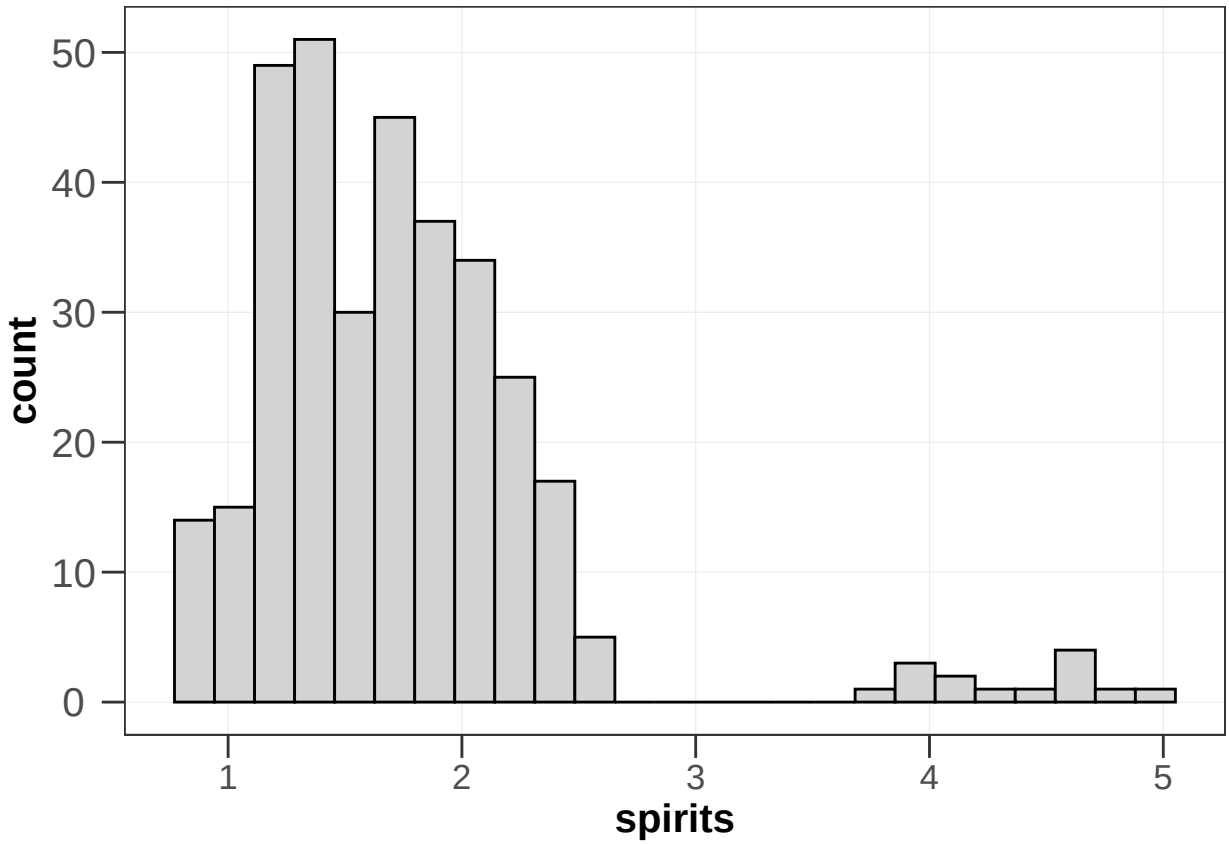


Additionally, you can also set axis properties (axis labels, ticks, titles) within the **theme()** function.

```
hist.plot <- Fatalities %>% ggplot() +
  geom_histogram(aes(x=spirits), bins=25, col="black", fill="lightgrey")+
  theme_bw()+
  theme(panel.grid.minor=element_blank(), panel.grid.major=element_line(size=0.2),
        axis.text.x=element_text(size=rel(1.5), angle=0, hjust=0.5),
        axis.text.y=element_text(size=rel(1.75), angle=0, hjust=0.5),
        axis.title=element_text(size=15, face="bold"),
        axis.ticks.length=unit(.3, "cm"))
```

```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

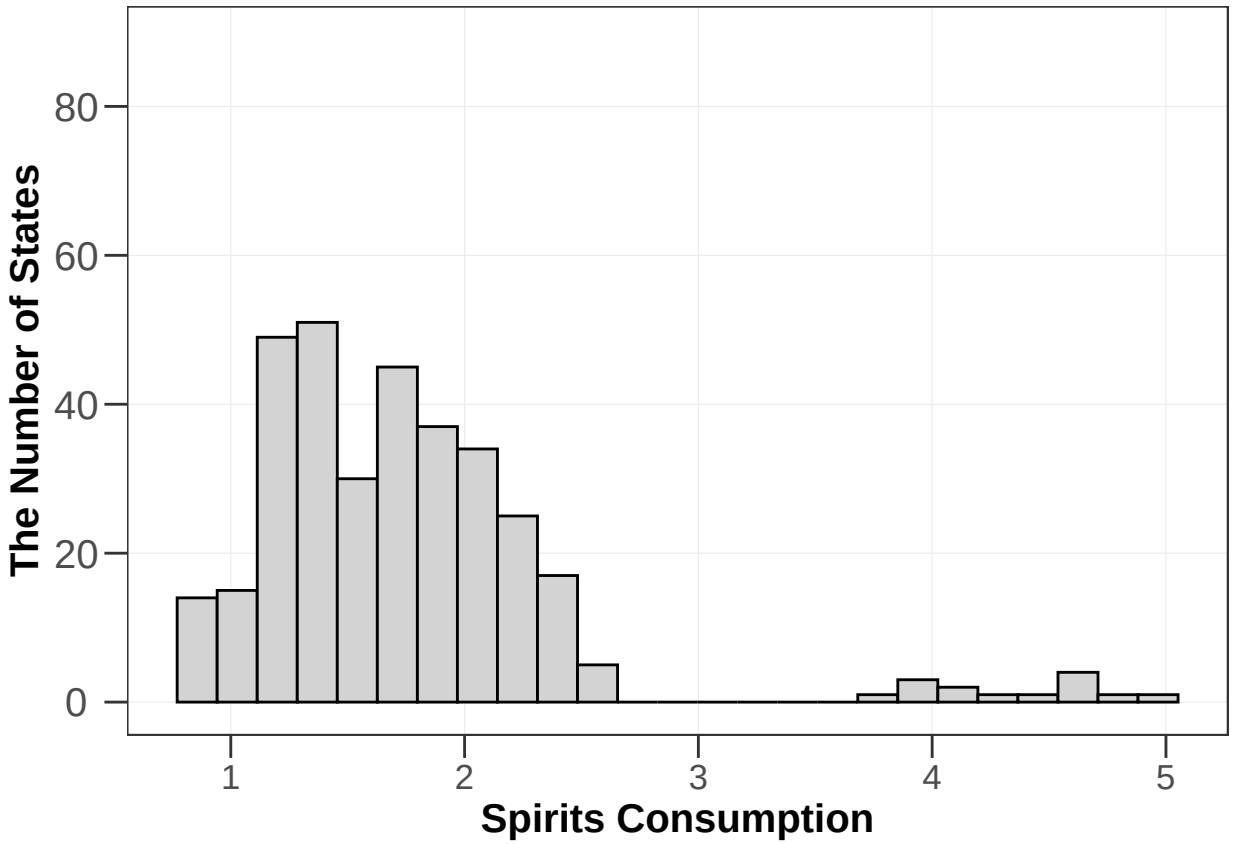
```
print(hist.plot)
```



There are hundreds of arguments in the **theme()** function. It's not necessary to memorize all of them.

Finally, if we want to change the labels on the x- and y-axes, we can use the **labs()** function. If you want to change the scale of the two axes, we can use **scale_x_continuous()** and **scale_y_continuous()** when x and y are both continuous variables.

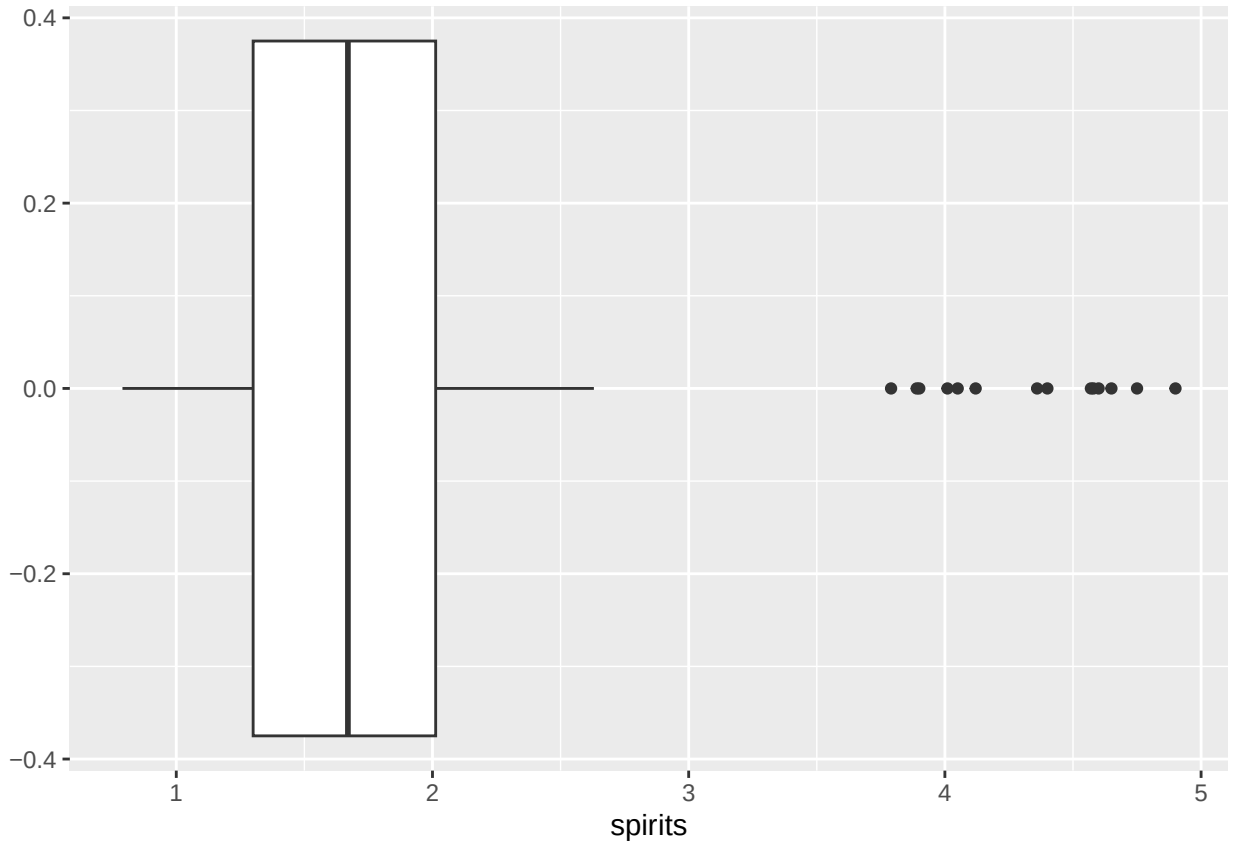
```
hist.plot + labs(x="Spirits Consumption", y="The Number of States") +
  scale_y_continuous(breaks=seq(0, 80, 20), limits=c(0, 89))
```



Instead of typing a lot of duplicated code, we can save the previous plotting code. When we want to add more layers or change plotting properties, we just need to call out that plotting object and then use a plus sign to add what we want. It's super handy!

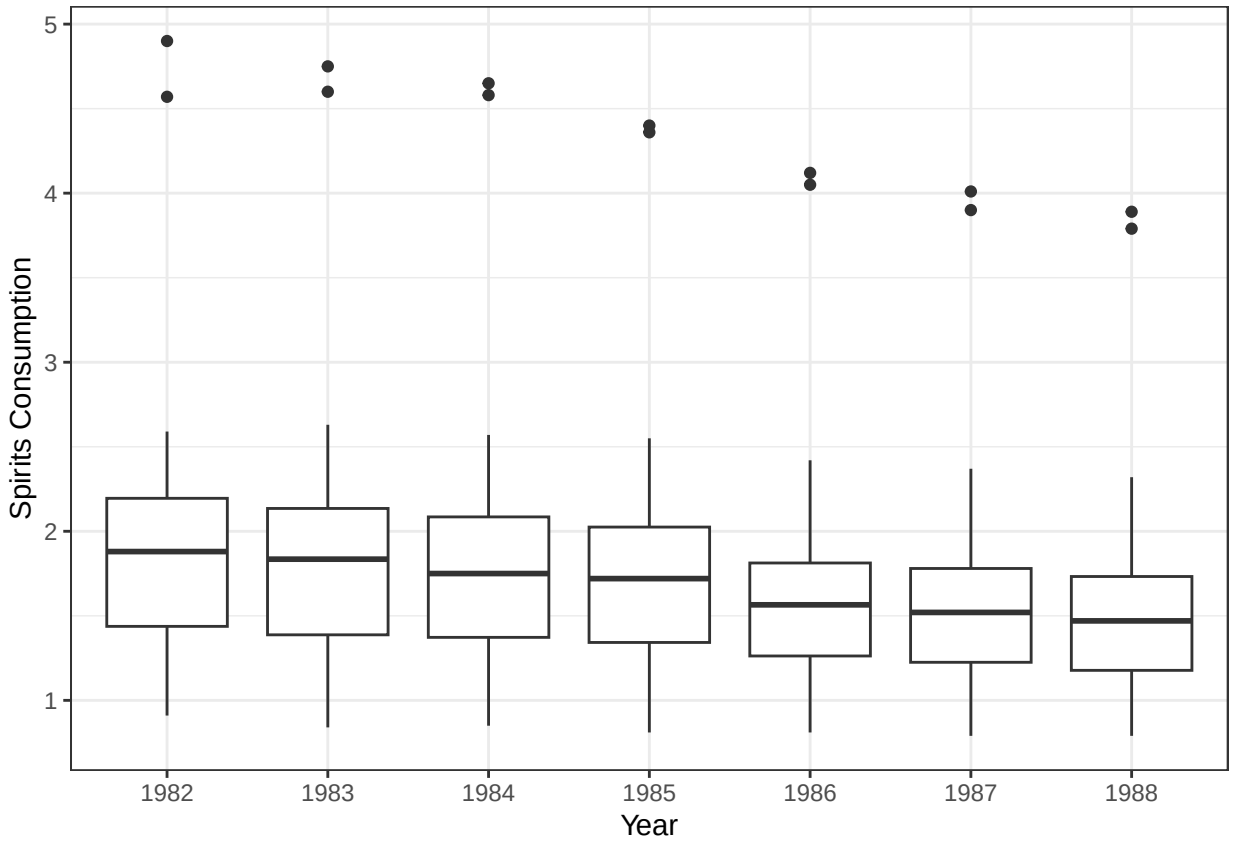
9.3 Box Plots

```
Fatalities %>% ggplot(aes(x=spirits)) + geom_boxplot()
```



The above plot is just a one-dimention boxplot. What if we want to create a sub-boxplot for each subgroup of the second variable?

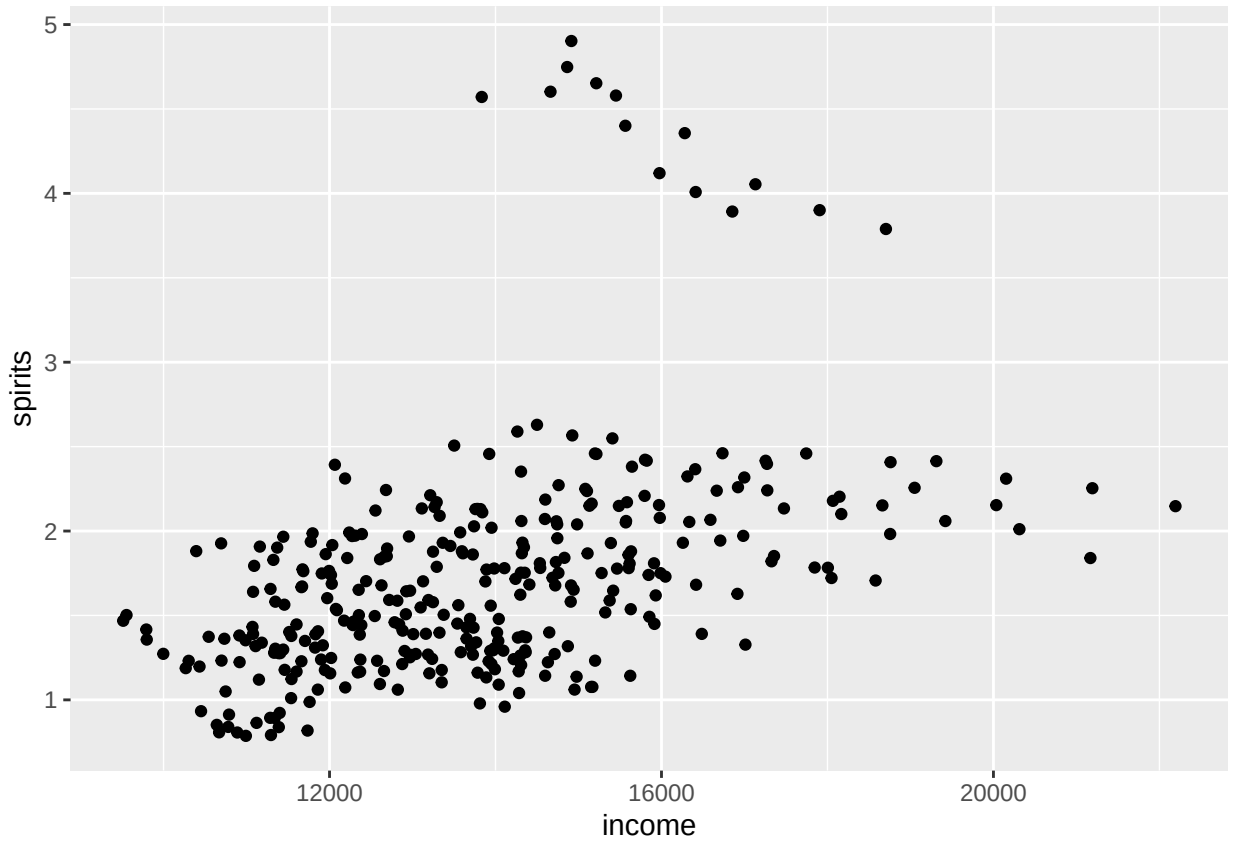
```
Fatalities %>% ggplot() + geom_boxplot(mapping=aes(y=spirits, x=year))+
  theme_bw()+ labs(x="Year", y="Spirits Consumption")
```



9.4 Scatter Plots

Scatter plots involve the mapping of observations to a point. Each observation/row in the original data set becomes a point in the plot. Thus, we need to use geom_point().

```
Fatalities %>% ggplot() + geom_point(mapping=aes(y=spirits, x=income), position="jitter")
```



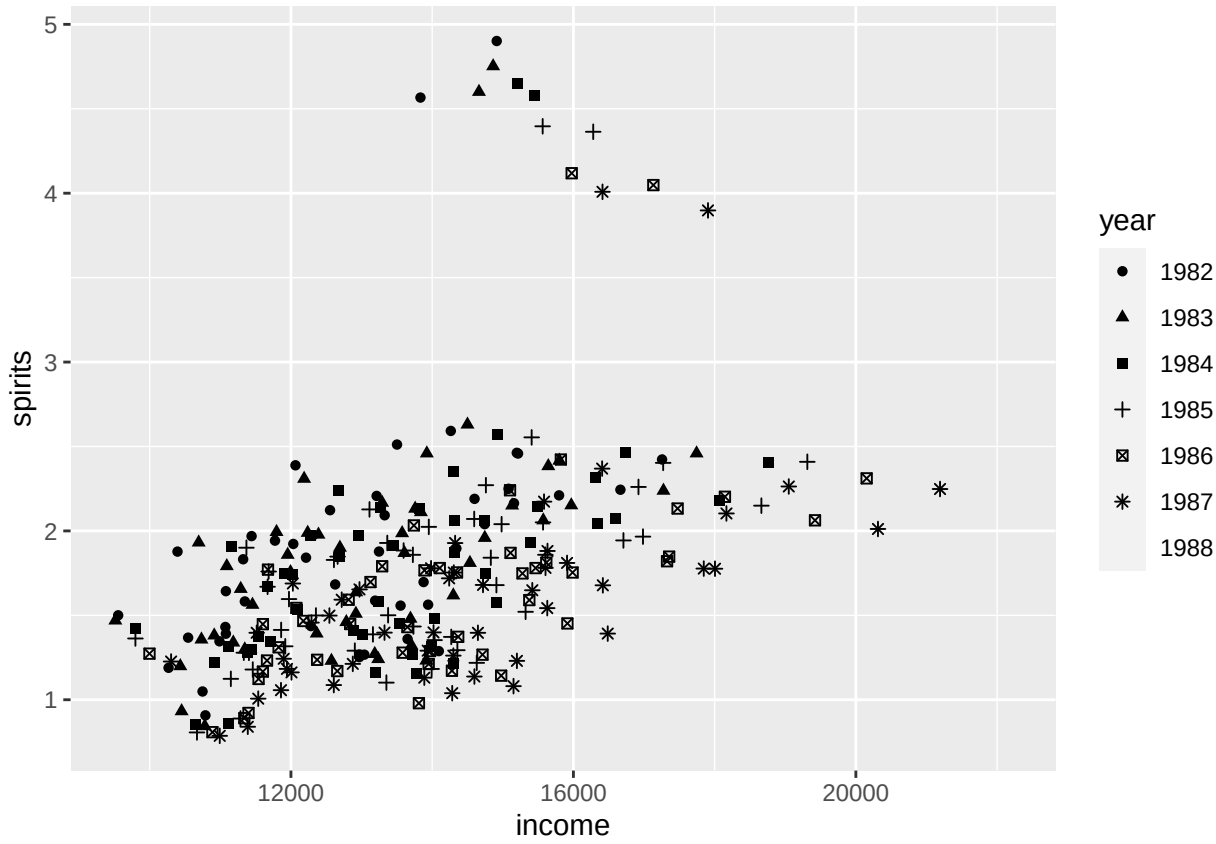

```
# when many points overlap, use the position="jitter" to avoid overlapping
```

If we are interested in checking out different years, we can map different years to different shapes.

```
Fatalities %>% ggplot() + geom_point(mapping=aes(y=spirits, x=income, pch=year), position="jitter")
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
```

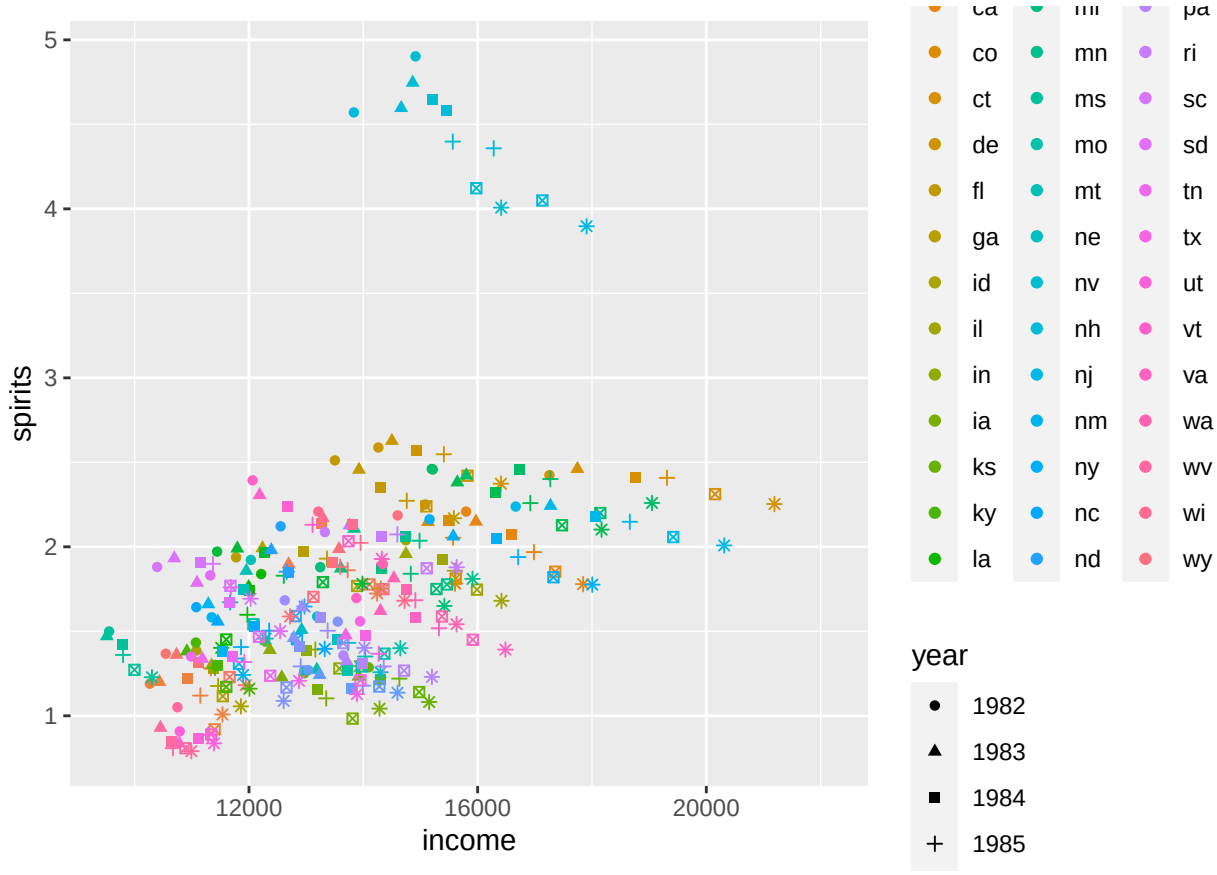
```
## Warning: Removed 48 rows containing missing values (`geom_point()`).
```




```
# Note that the shape palette can deal with 6 discrete values at most
```

If you want to see state information as well, you can add one more aesthetic. This time consider using the color aesthetic.

```
Fatalities %>% ggplot() + geom_point(mapping=aes(y=spirits, x=income, pch=year, col=state),
               position="jitter")
```

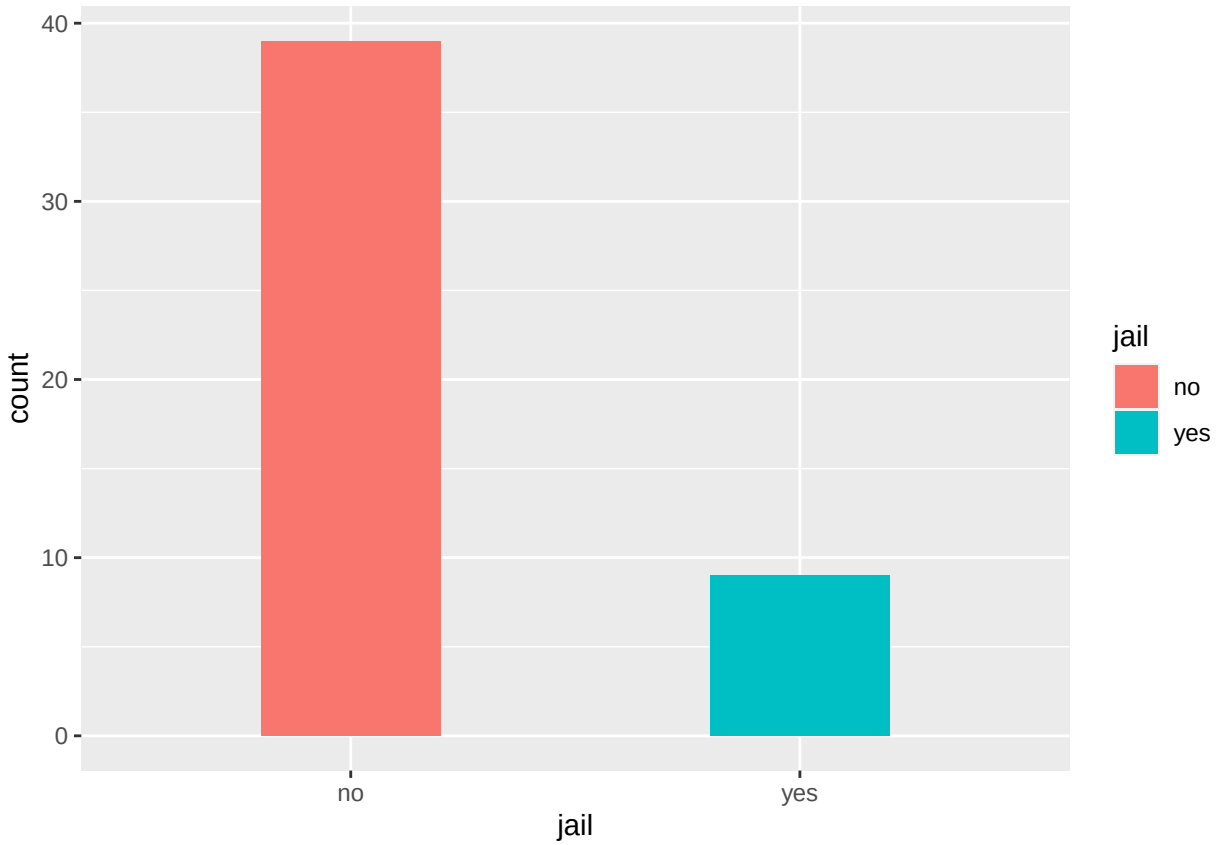


Based on this plot, we find that the outliers seem to come from a single state.

Although the plot conveys much information, the legend is too messy because there is insufficient space for the legend. Additionally, the color mapping does not work very well because there are too many states to map. To overcome this limitation, we can use facet, a skill that will be covered shortly.

9.5 Bar Plots

```
Fatalities %>% filter(year==1982) %>%
  ggplot() + geom_bar(mapping=aes(x=jail, fill=jail), width=0.4)
```




```
# Showing which states are coded as "yes" in 1982
# Filter for 1982 and jail == "yes"
Fatalities %>%
  filter(year == 1982, jail == "yes") %>%
  distinct(state)
```

```
##     state
## 8      az
## 92     ks
## 106    la
## 113    me
## 162    mt
## 274    tn
## 309    wa
## 316    wv
## 330    wy
```

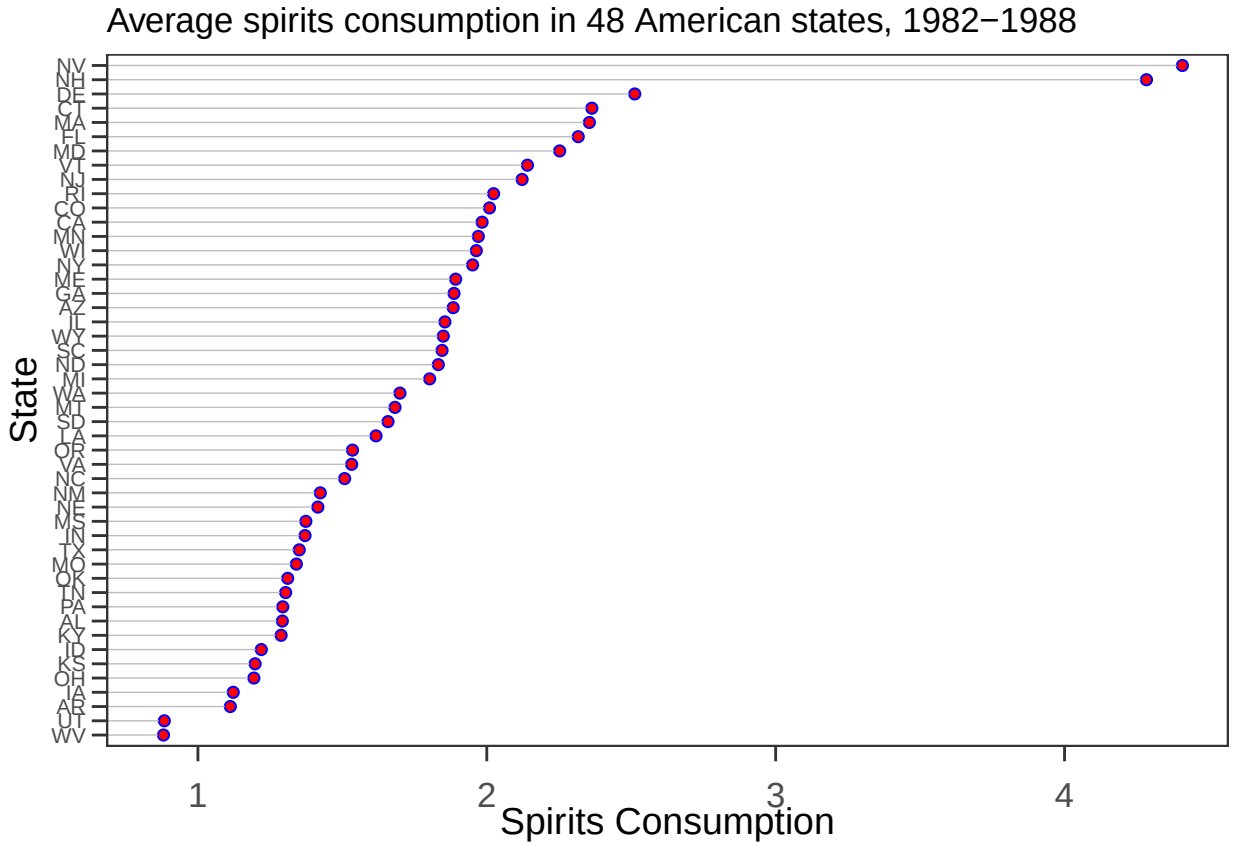
9.6 Dot Plots

We can use the **geom_dotplot()** function to create a dot plot. The default output does not look very good, and we can add a segment to demonstrate the distance between each dot and the y-axis.

```
dot.plot <- Fatalities %>%
  group_by(state) %>%
  dplyr::summarise(spirits.avg=mean(spirits, na.rm=T)) %>%
  mutate(state=str_to_upper(state)) %>%
  ggplot(aes(x=spirits.avg, y=reorder(state, spirits.avg))) +
  geom_dotplot(binaxis="y", dotsize=0.5, fill="red", col="blue") +
```

```
  geom_segment(aes(yend=state),
               xend=0,
               color="grey", linetype="solid", lwd=0.25)+
  theme_bw()+
  theme(panel.grid.major=element_blank(), panel.grid.minor=element_blank(),
        axis.text.x=element_text(size=rel(1.5), vjust=-1),
        axis.text.y=element_text(size=rel(0.9)), axis.ticks.length=unit(.2, "cm"),
        axis.title=element_text(size=14))+
  labs(x="Spirits Consumption", y="State")+
  ggtitle("Average spirits consumption in 48 American states, 1982-1988")

dot.plot
```

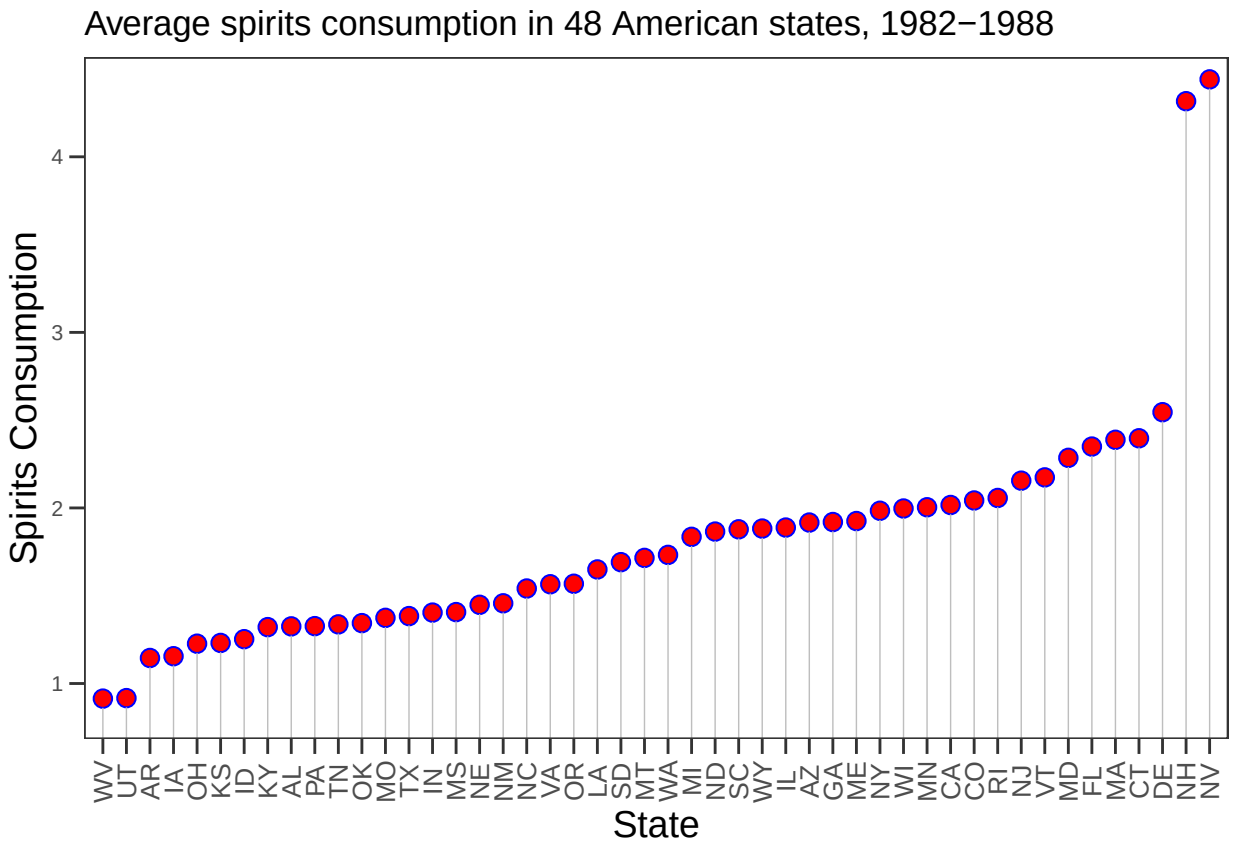
```
## Bin width defaults to 1/30 of the range of the data. Pick better value with
## `binwidth`.
```



Sometimes you may want to flip the coordinate system, putting your x variable on the Y axis and y variable on the X axis. This can be done through the **coord_flip()** function.

```
dot.plot + coord_flip() +
  theme(axis.text.x=element_text(size=rel(1.1), angle=90, vjust = 0.5))
```

```
## Bin width defaults to 1/30 of the range of the data. Pick better value with
## `binwidth`.
```



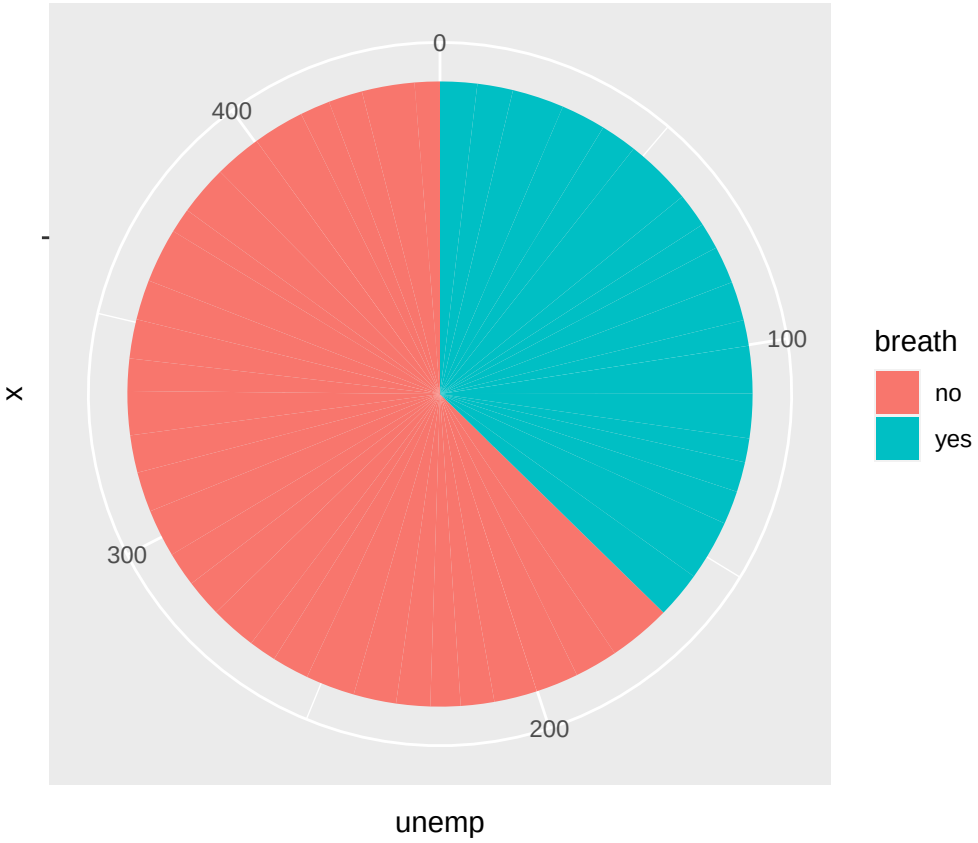
Obviously, you will need to apply the **theme()** function to change the font size for the values of the state variable because they overlap a lot.

9.7 Pie Charts

There is no built-in function for creating a pie chart. However, you may use a **geom_bar()** and **coord_polar()** to create one. The logic is as follows:

- You have a group variable and a value variable
- You build a stacked barplot with one bar only by virtue of the **geom_bar()** function
- You make the bar circular with the **coor_polar()** function

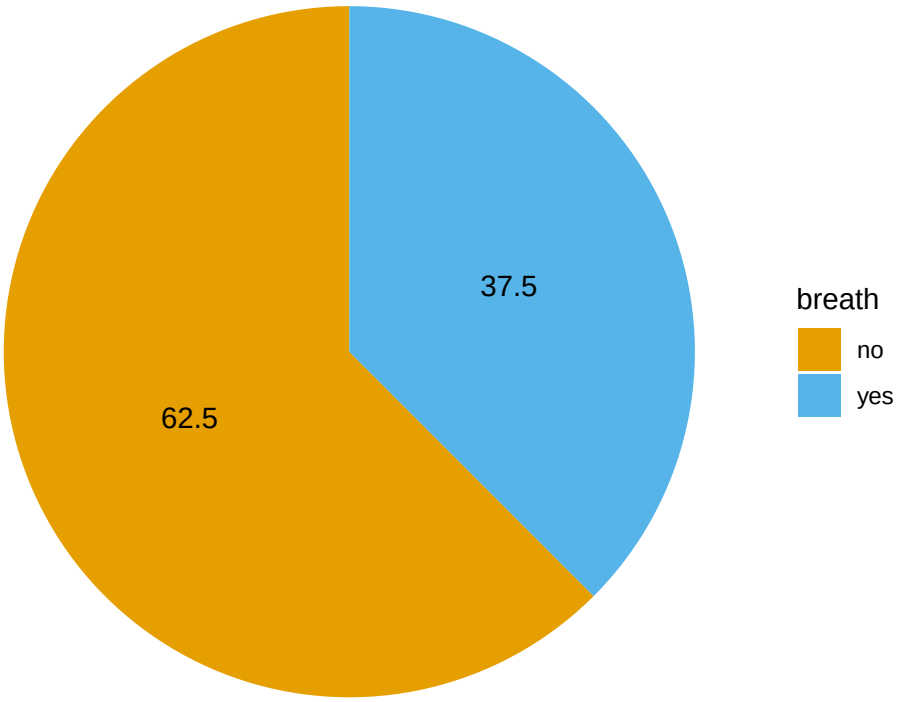
```
Fatalities %>%
  filter(year==1982) %>%
  ggplot() +
  geom_bar(aes(x="", y=unemp, fill=breath), stat="identity") +
  coord_polar(theta="y")
```



If you do not like this default polar coordinate system, you may use the **theme_void()** function. Additionally, you can add numeric labels and specify different colors by using **geom_text()** and **scale_fill_manual()**, respectively.

```
Fatalities %>%

  # Preparing data
  filter(year==1982) %>%
  select(breath) %>%
  group_by(breath) %>%
  dplyr::summarise(n=n()) %>%
  mutate(sum=sum(n), percentage=n/sum*100) %>%

  # Creating a bar plot first
  ggplot(aes(x="", y=percentage, fill=breath)) +
  geom_bar(stat="identity") +

  # Adjusting the coordinate system
  coord_polar(theta="y") +

  # Removing the default theme
  theme_void() +

  # Adding numeric labels
  geom_text(aes(label=percentage), position=position_stack(vjust=0.5)) +

  # Choosing your preferred color
```

```
  scale_fill_manual(values=c("#E69F00", "#56B4E9"))
```



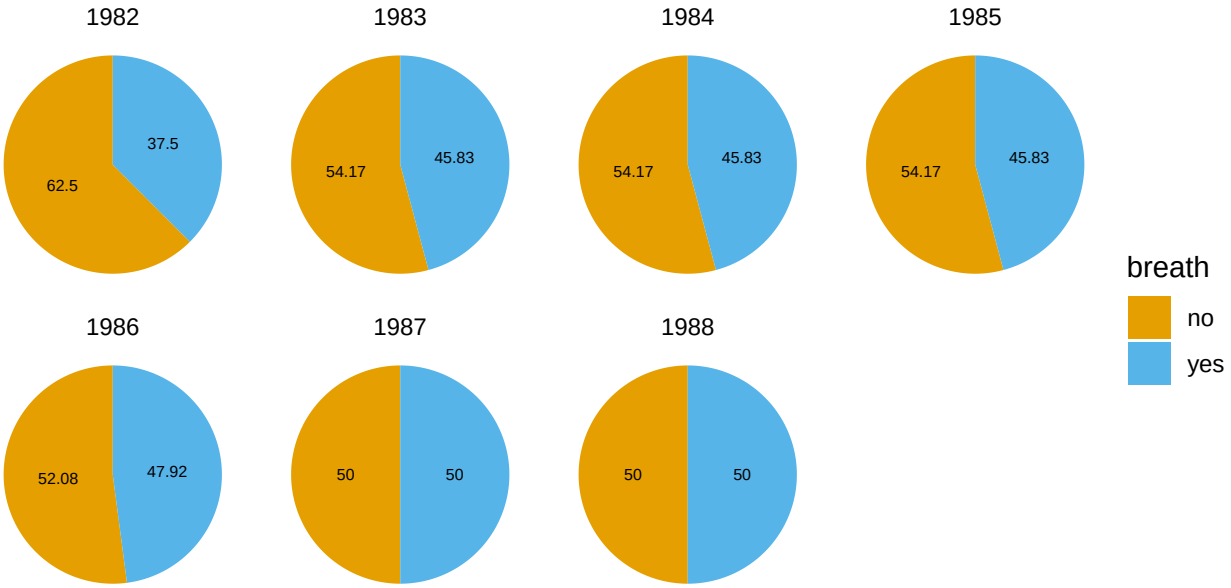
What if you want to create a plot for each year? In this case, you can use the facet feature. A facet is just a subplot created based on a subset of the entire data. You will need to decompose the data set into different years, then you create a pie chart for each year.

```
Fatalities %>%
  select(year, breath) %>%
  group_by(year, breath) %>%
  dplyr::summarise(n=n()) %>%
  mutate(sum=sum(n), prop=round(n/sum*100, 2)) %>%
  print(n=20) %>%
  ggplot(aes(x="", y=prop, fill=breath)) +
  geom_bar(stat="identity") +
  coord_polar(theta="y") +
  theme_void() +
  geom_text(aes(label=prop), position=position_stack(vjust=0.5), size=2.2) +
  scale_fill_manual(values=c("#E69F00", "#56B4E9")) +

  # Creating sub-figures with the facet function
  facet_wrap(~year, nrow=2)
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 5
## # Groups:   year [7]
##    year  breath     n   sum  prop
##    <fct> <fct>  <int> <int> <dbl>
##  1 1982  no        30    48  62.5
##  2 1982  yes       18    48  37.5
##  3 1983  no        26    48  54.2
##  4 1983  yes       22    48  45.8
```

```
##  5 1984  no        26    48  54.2
##  6 1984  yes       22    48  45.8
##  7 1985  no        26    48  54.2
##  8 1985  yes       22    48  45.8
##  9 1986  no        25    48  52.1
## 10 1986  yes       23    48  47.9
## 11 1987  no        24    48  50
## 12 1987  yes       24    48  50
## 13 1988  no        24    48  50
## 14 1988  yes       24    48  50
```



9.8 Line Plots

When we have time series data, it is helpful to use line plots to show how variables evolve over time. You can plot multiple time series in one plot (e.g., interest rate, inflation rate). This can be done through a mapping of different groups to an aesthetic such as color or shape.

Keep in mind that when you create a line plot, you must specify the group variable in the **aes()** function in addition to determining x and y. Otherwise, the lines will not appear in the plot.

- if you only have one time series variable to plot, group=1;
- if you have more than one time series variable to plot, the group argument should be equal to the grouping variable.

```
Fatalities %>%

  # Selecting variables for plotting
  select(year, state, fatal1517, fatal1820, fatal2124) %>%

  # Filtering data by states
  filter(str_detect(state, "^w")) %>%

  # Transforming the data structure
  pivot_longer(fatal1517:fatal2124,
               names_to="Age Group",
               values_to="fatalities") %>%
  # print(n=10) %>%
```

```
# Creating the plot
ggplot(aes(x=year, y=fatalities, group=`Age Group`, col=`Age Group`)) +
geom_point() +
geom_line() + # must specify the group argument
facet_wrap(~state, nrow=2) +
theme_bw() +
theme(axis.text.x=element_text(size=rel(1), angle=90, vjust=0.35), legend.position="bottom")
```