Appendix

Appendix 1: Navigating R

1.1. Installing and Loading Packages

You will find that installing packages can be a pain. This section will try to make it as painless as possible. You can check what packages are loaded, as well as load packages, using the *Packages* tab. We suggest that you load packages using code so when you run the code in the future, the package loads automatically.

The first thing you want to do is install the desired package using the `install.packages()` function, and it can then be loaded with the `library()` function, which we suggest to use over the `require()` function. This can be done as follows:

```
install.packages("PackageName")  # Install package

library(PackageName)  # Load package
```

Was the package successfully loaded? If yes, GREAT! And you can stop reading this section. If it failed :-(keep reading and try a few trouble shooting steps.

Package not found:

Make sure that you typed the code correctly. Perhaps you forgot the period between install and packages or the S at the end of the word packages. Also, check to make sure that there are quotation marks around the package name when you install the package, but no quotation marks when you load the package. Remember as well that `R` is case sensitive.

Package XXX not available:

1. Sometimes `R` can be buggy and not allow you to set repositories. If step 1 fails, We have found the most success with the code below. Make sure to use the `library()` function afterward if the installation was successful.

```
install.packages("PackageName",
                 repos='https://cloud.r-project.org')
```


A. Wooditch et al., *A Beginner's Guide to Statistics for Criminology and Criminal Justice Using R*, https://doi.org/10.1007/978-3-030-50625-4

2. If the prior step fails, make sure that the device you are using is connected to the Internet and rerun the function.
3. `R` may not be checking in the right place to find the package. You can try to fix this by typing `setRepositories()`. You then want to enter the values 1 (for CRAN). Run `setRepositories()` again, and then enter 5 (for CRAN extras).
4. Still no work? Throw the laptop against the wall. No, just kidding. First, recheck that you have completed all prior steps correctly. Second, if that does not work, I would search the specific error that you are receiving online. There are many online forums that are very helpful in solving this issue.

1.2. Specifying Packages

In some circumstances, different functions will have the same name in different packages, or a package will contain a function with the same name as a function available in base `R`. Just because functions have the same name, it does not mean that they do the same thing or behave in the same way. For example, base `R` has a function called `range()` which simply calculates the range of values in a vector. However, the package called `mosaic` used in Chapter 6 also includes a function called `range()`, which behaves a little differently, containing additional options. You might have noticed that when you load packages, `R` will give you warnings about the function names which are conflicting. By default, `R` will use the function from the package which you loaded last. To avoid any confusion, you can actually specify which package you want to use by using `::` before the function name, either with the package name or simply `base` for base `R`.

```
base::range(numeric_vector) # Uses the range() function
                            ## from base R
mosaic::range(numeric_vector) # Uses the range() function
                              ## from mosaic
```

1.3. Projects and Working Directories

Working with `here()` As noted in Chapter 2, we recommend that you use `R` projects. Wherever your `R` project is saved will be the default working directory. This is very helpful when it comes to loading or saving data, because you don't have to specify the whole working directory each time. However, doing this yourself might bring up problems, largely because working directories can differ between operating systems (e.g., Windows and Mac). Your code might work for you, but not for other people (or somebody else's code won't work for you!). A useful way to address this is to use the `here()` function from the `here` package, in tandem with an `R` project. Instead of specifying the working directory manually, you can input each section of the directory within `here()` and stick them altogether for you in a way that works consistently.

```
# Not recommended: specify whole working directory outside
## of a project
df <- read_csv(file = "C:/Users/your_name/Documents/my_
                       project_folder/Datasets/my_file.csv")

# Improvement: work from your project
# But, this might not work for other people!
df <- read_csv(file = "Datasets/my_file.csv")

# Ideal situation: work from your R project and use here()
df <- read_csv(here("Datasets", "my_file.csv"))
```

1.4. Setting Working Directory

Although we recommend that you use `R` projects, and use working directories via the `here()` function, it is possible to set the working directory manually from within `R` using the function `setwd()`. How you do this varies between Windows and Mac, as demonstrated below.

Windows

Note that in Windows, you cannot simply copy and paste your desired working directory. You need to set the directory using two backward slashes or, alternatively, one forward slash.

```
# Either use two backward slashes
setwd("C:\\Users\\your_name\\Documents\\my_project_folder")

# Or change backward slashes to forward slashes
setwd("C:/Users/your_name/Documents/my_project_folder")
```

Mac

To set your working directory on a Mac:

```
setwd("/Users/your_name/Documents/my_project")
```

1.5. Get Working Directory

You can check to see if your working directory was successfully defined, or simply check what it is, by using `getwd()` on its own.

```
getwd()
```

1.6. Opening Data Files and Exporting Data

R is capable of reading and exporting data in numerous different formats, many of which are used in other software. Here, we make use of functions from the packages haven, readr, and readxl, contained within the tidy-verse, as well as functions from the openxlsx and foreign packages.

1.6.1. R Data Files

R data files have the file extension .R (or .RDA if the file was created in an older version of R). Note that if you don't assign the dataset to an object, R specifies a dataset name for you.

- Read: load(file) loads your .rda file into the R environment.
- Write: save(x, path) saves your data frame x into an R data file (.rda) specified in path.

```
# .rda Example
load("Dataset Name.rda") # Imports your .rda file

# Exports your data as an .rda file
write(ncvs, file = "New Dataset Name.rda")
```

1.6.2. General Delimited

- Read: read_delim(file, delim, ...) from the readr package reads in delimited files, where users can specify the delimiter in the delim argument. If you are importing a comma- or tab-delimited file, see below.
- Write: write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = TRUE,...) from the readr package writes R data objects x to a delimited file specified in path. The delimiter being used should be entered into the delim argument. The default arguments for this function include that the string that should be used for missing values is NA; the new file should replace the old and not be appended to the bottom (set append = TRUE for the opposite) and that the column names will be included at the top of the file.

```
# General Delimited Example
ncvs <- read_delim("Dataset Name.txt",
        delim = "\t", col_names = TRUE) # Importing a
                                        ## tab-delimited
                                        ## text file

write_delim(ncvs, "New Dataset Name.txt", delim = "\t",
na = "NA", append = FALSE, col_names = TRUE) # Exporting the
                                             ## tab-delimited
                                             ## text file
```

1.6.3. *Comma Separated*

- Read: `read_csv(file, ...)` from the `readr` package imports comma-separated files. You can specify whether or not you want the first row of your data to be considered the variable names. `read_csv2(file, ...)` can also be used for the `;` separator.
- Write: `write_csv(x, path, na = "NA", append = FALSE, col_names = TRUE,...)` writes your data frame `x` to a comma-separated file specified in `path`. The default arguments for this function include that the string that should be used for missing values is `NA`; the new file should replace the old and not be appended to the bottom (set `append = TRUE` for the opposite); and the column names will be included at the top of the file. `write_csv2()` may be used for the `;` separator.

```
# .csv Example
# Importing CSV file with the header
ncvs <- read_csv("Dataset Name.csv", col_names = TRUE)

# Importing CSV file without header
ncvs <- read_csv("Dataset Name.csv", col_names = FALSE)


# Exporting as a new CSV file with headers
write_csv(ncvs, file = "NewName.csv", na = "NA",
append = FALSE, col_names = TRUE)

# Exporting as a new CSV file without headers
write_csv(ncvs, file = " NewName.csv", na = "NA",
append = FALSE, col_names = FALSE)
```

1.6.4. *Tab Separated*

- Read: `read_tsv(file, ...)` from the `readr` package imports a tab-delimited file.
- Write: `write_tsv(x, path, na = "NA", append = FALSE, col_names = TRUE, ...)` from the `readr` package can be used to write a data frame or matrix `x` to a tab-delimited file. `col_names` must be either `True or False` and specifies whether the column names should be written to the top of the file.

```
# .tsv Example

# Import the tab-delimited text file
ncvs <- read_tsv("Dataset Name.tsv", col_names = TRUE)

# Export as a tab-delimited text file
write_tsv(ncvs, file = "NewName.tsv", na = "NA",
          append = FALSE, col_names = TRUE)
```

1.6.5. Excel

- Read: `read_excel(file, ...)` imports both .xls and .xlsx files into `R` through the `readxl` package. `read_excel(file, sheet = "name")` and `read_excel(file, sheet = 2)` both import a specific sheet from the Excel file that you want to use, either by name or index.
- Write: `write.xlsx(x, path, ...)` from the `openxlsx` package enables you to write your data frame `x` as an Excel file specified in `path`.

```
# Excel Example

# Import excel file as the object ncvs
ncvs <- read_excel("Dataset Name.xlsx")

# Export as dataset "New Dataset Name.xlsx"
write.xlsx(ncvs, file = "New Dataset Name.xlsx")
```

1.6.6. dBASE

- Read: `read.dbf(file, ...)` from the package `foreign` may be used to read DBF files.
- Write: `write.dbf(x, path, ...)` can write the `R` data frame `x` in DBF format.

```
# dBASE Example

# Import dbf dataset as the object ncvs
ncvs <- read.dbf("Dataset Name.dbf")

# Export as dataset named New Dataset Name.dbf
write.dbf(ncvs, file = "New Dataset Name.dbf")
```

1.6.7. Stata

- Read: `read_dta(file, ...)` reads *.dta* files using the package `haven`.
- Write: `write_dta(x, path, version = 14, ...)` writes your data to a Stata *.dta* file. This currently works with Stata versions 8–15.

```
# Stata Example

# Import Stata dataset as the object ncvs
ncvs <- read_dta("Dataset Name.dta")

# Export as Stata dataset named New Dataset Name.dta
write_dta(ncvs, file = "New Dataset Name.dta")
```

1.6.8. SPSS

- Read: `read_sav(file, ...)` from the package `haven` reads *.sav* files, and `read_por()` can be used for older SPSS files.
- Write: `write_sav(x, path, ...)`.

```
# SPSS Example

# Import SPSS dataset as the object named ncvs
ncvs <- read_sav("Dataset Name.sav")

# Export as SPSS dataset named New Dataset Name.sav
write_sav(ncvs, file = "New Dataset Name.sav")
```

1.6.9. SAS

- Read: `read_sas(file, ...)` from the `haven` package reads *.sas7bdat* files, and `read_xpt()` can be used to open SAS transport files (versions 5 and 8)
- Write: `write_sas(x, path, ...)` writes your data to a SAS format file specified in `path`, though this functionality is currently experimental. Make sure to keep apprised of package updates to get the most out of these functions.

```
# SAS Example

# Import SAS dataset
ncvs <- read_sas("Dataset Name.sas7bdat")

# Write to a SAS data file named New Dataset Name.sasb7dat
write_sas(ncvs, file = "New Dataset Name.sasb7dat")
```

1.6.10. From Web URL

You can load data directly from the web as long as you have the URL. To do so, you will want to create an object with the permanent `url` address. Then, we use a function to read the data into `R`. The data that can be saved using an `api` is in tab-separated format; therefore, we use the `read.table()` function from base `R`. We pass two arguments to the function. The `sep= '\t'` is telling `R` this file is tab separated. The `header = T` function is telling `R` that is `TRUE` (T) that this file has a first row that acts as a header (this row has the name of the variables). See example below where we load the data into an object named *sharkey*:

```
# URL Example
data_url <- "https://dataverse.harvard.edu/api/access/
             datafile/:persistentId?persistentId=doi:
             10.7910/DVN/46WIH0/ARS2VS"

sharkey <- read.table(url(data_url), sep = '\t',header = T)
```

1.6.11. Systat

- Read: `read.systat(file, ...)` reads *.sys* or *.syd* files using the `foreign` package.
- Write: The `foreign` package does not currently support writing data in `R` as a Systat file. Make sure to keep apprised of package updates to get the most out of these functions.

```
# Systat Example
ncvs <- read.systat("Dataset Name.syd") # Import Systat file
```

1.6.12. Minitab

- Read: `read.mtp(file, ...)` reads *.mtp* files using the `foreign` package.
- Write: The `foreign` package does not currently support writing data in `R` as an *.mtp* file. Make sure to keep apprised of package updates to get the most out of these functions.

```
# Minitab Example
ncvs <- read.mtp("Dataset Name.mtp") # Import .mtp dataset
```

1.6.13. Matlab

- Read: `read.mat(file, ...)` reads *.mat* files using the `rmatio` package.
- Write: `write.mat(x, path,...)` to save `R` objects as a MAT file.

```
# Matlab Example
ncvs <- read.mat("Dataset Name.mat") # Import MAT file

# Write object ncvs to a MAT file
write.mat(ncvs, file = "New Dataset Name.mat")
```

1.6.14. JSON

- Read: `read_json(file, ...)` reads JSON files using the `jsonlite` package.
- Write: `write_json(x, path,...)` to save `R` objects in JSON format.

```
# JSON Example
ncvs <- read_json("Dataset Name.json") # Import JSON file

# Write object ncvs to a JSON file
write_json(ncvs, file = "New Dataset Name.json")
```

1.7. Viewing Data Frame

It is good practice to do this to ensure `R` has read the data correctly and there's nothing terribly wrong with your dataset. It can also give you a first impression for what the data look like. If you are used to spreadsheet-like views of data, you can use the `View()` function, which should open this view in `R` Studio. This can also be used to view `R` objects.

```
# Example with a dataset named burglary_df
View(burglary_df)
```

1.8. Using `attach()` and `detach()`

The `attach` function attaches the dataset name to the `R` file path so you can access variables of a dataset without calling the dataset name. You can turn off the attach function using `detach()`.

```
# Without *Attach*:
df_name$my_variable

# With *Attach*:
attach(df_name)
my_variable

# Turn off *Attach*
detach("df_name")
```

1.9. Interrupting `R`

Sometimes `R` can take a long time to execute a task if you have asked it to perform a particularly complex or computationally demanding operation. In cases such as these, a small red stop sign will appear in the top right corner of your console in `RStudio`. Click on this stop sign to interrupt the process. You can also use the *Esc* key in Windows or Mac. If for some reason this still does not halt the process, you may need to wait out the operation or navigate to your operating system's task manager and quit `RStudio` altogether. Take caution, however, that exiting the program will cause any unsaved work to be lost. It is best practice to save your work, and save it often!

1.10. Keyboard Shortcuts

Though it is possible to use your cursor to navigate and execute tasks within the `RStudio` IDE, `RStudio` has a large number of keyboard shortcuts you can leverage without having to use your mouse. To check on the available shortcuts for `RStudio`, simply navigate to the *Tools* menu in the task bar, and select *Keyboard shortcuts help* from the dropdown. Users even have the option of customizing their keyboard shortcuts. Table A1 lists just a few common shortcuts that may come in handy for most users (adapted from this article: https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts).

Table A1 R Keyboard Shortcuts

TASK	SHORTCUT (WINDOWS/LINUX)	SHORTCUT (MAC)
Clear console	Ctrl + L	Ctrl + L
Navigate function history	Up/Down	Up/Down
Interrupt executing function	Esc	Esc
New document (except Chrome/Windows)	Ctrl+ Shift + N	Cmd + Shift + N
New document (Chrome only)	Ctrl + Alt + Shift + N	Cmd + Shift + Alt + N
Open document	Ctrl + O	Cmd + O
Save document	Ctrl + S	Cmd + S
Run current line/section	Ctrl + Enter	Cmd + Return
Run current document	Ctrl + Alt + R	Cmd + Option + R
Insert assignment operator	Alt + -	Option + -
Insert pipe operator	Ctrl + Shift + M	Cmd + Shift + M
Search R Help	Ctrl + Alt + F1	Ctrl + Option + F1
Quit session	Ctrl + Q	Cmd + Q

Again, be sure to check the *Keyboard shortcuts help* menu in your version of `RStudio` to see the available shortcuts.

Appendix 2: Data Transformation

Data transformation is often a necessary task in the data analysis process. Of course, `R` has multiple ways of accomplishing the various transformations you may need to do. For most of the tasks in this section, we focus on base `R` and `tidyverse` methods.

2.1. Recoding or Creating a New Variable

There are many different ways one can achieve generating or recoding a variable. For instance, we can use the `mutate()` and `case_when()` functions from `dplyr` to create transformed variables based on some criteria.

We can also create new variables, based on some computation or combination of other variables. In our data frame, `df`, we want to recode a variable called shot, which is a character variable of whether someone was injured via gunshot or another method.

```
# Recode the variable injury_type into variable called "shot"
  df <- df %>%
  mutate(shot = case_when
        (injury_type %in% "gun" ~ "Gun shot",
         injury_type %in% "stab" ~ "Not Gun Shot"))
```

You can also create new variables based on some calculation. For instance, if you wanted to create a variable that was a ratio of two variables, you could simply divide one variable by the other within the `mutate()` function.

```
# Create a new variable that is a calculation
# In this case, a ratio of var1 to var2
df <- df %>%
   group_by(sex) %>%
   mutate(ratio = var1/var2)
```

Or you could create a new variable that was equal to the sum of four variables divided by four.

```
# Create a new variable that is equal to the sum of four
## vars divided by four
df <- df %>%
   group_by(sex) %>%
   mutate(someindexscore = (var1 + var2 + var3 + var4)/4)
```

You also may want to add a totally new column to your data frame. You can use the `add_column()` function for this. The following example uses this function to create an ID variable for our data frame, `df`:

```
# Create a new column that is a row ID
add_column(df, newid = 1:nrow(df))
```

These are just several key ways to recode variables using `dplyr`. There are many functions that can perform various tasks that might be useful to you as you are cleaning your data. Which functions you choose simply depends on what you need to do with your data.

2.2. Binning Variables

Binning variables is useful, for example, when you need to create categories of a continuous variable, or when you simply want to collapse a certain number of categories into a fewer number of categories. Using functions from `dplyr` and `forcats`, like `mutate()` and `case_when()`, or `fct_collapse()`, you can define how you want your variable to be binned. For instance, below, we create the variable injury_location that collapses six different places people could have been injured into three categories: home, school, and other.

```
# Create a new collapsed character variable injury_location
## from numeric values
df <- df %>%
  mutate(injury_location = case_when(location %in% 0 ~ "Home",
                                     location %in% 1 ~ "School",
                                     location %in% 2:5 ~ "Other"))

# Create collapsed version of variable if it's a factor using
## fct_collapse()
# Manually decide your factor levels
df$injury_location <- fct_collapse(df$location,
  other = c("Work", "Park", "Mall", "Other"),
  school = "School",
  home = "Home")
```

If you have a continuous variable like age or number of arrests, you may want to be able to bin them into broader categories under certain circumstances. In the example below, we are converting a continuous variable of the number of full-time sworn officers in each law enforcement agency (Q_8) into an ordinal measure of agency size (agcysize).

```
# Create bins of agency size from the variable Q_8
bwcs <- bwcs %>%
  mutate(agcysize=case_when(
   Q_8 %in% 0:10 ~ "0-10 FTS",
   Q_8 %in% 11:50 ~ "11-50 FTS",
   Q_8 %in% 51:100 ~ "51-100 FTS",
   Q_8 %in% 101:500 ~ "101-500 FTS",
   Q_8 %in% 501:1000 ~ "501-1000 FTS",
   Q_8 >= 1001 ~ ">1000 FTS"))
```

2.3. Dealing with Missing Data

Missing data are a common occurrence in criminological research. It might arise due to a variety of reasons. In police-recorded crime data, it might be due to recording issues, or perhaps the information was simply not available or unknown (e.g., an offender's home address). In survey data, respondents might have refused to answer a question, or the respondent might have simply dropped out of participating. How to deal with missing values is a field of research in itself and should be considered carefully. One important reason for this is because missing data might be missing for an underlying systematic reason which impacts on your research. For instance, many people don't like answering questions about their income, but perhaps certain demographic groups (or people on certain incomes) are especially unlikely to answer this question. It would be unwise to simply remove all these people from your data, because you would end up with a biased sample only containing people who were willing to discuss their income. So, consider these issues carefully when dealing with your missing data! With that in mind, the following functions might be of use.

First, let's create an example data frame containing information about the number of prior offenses committed by a sample of ten offenders. The column crime_count contains missing values, because some of our offenders did not want to discuss their offending history. Note that `R` actually treats missings as missings using `NA`. This might seem obvious, but many software assign a specific value like `9999` to define a missing value. Note that `NA` is *not* the same as stating `NA`, which would be treated as a character, and therefore not missing!

```
df <- data.frame(
  id = 1:10,
  crime_count = c(1,4,0,NA,6,0,23,NA,54,NA))
```

To remove observations with missing values (which as stated above, is not always appropriate), we can use `drop_na()` from the `tidyr` package. Make sure you have this package installed and loaded before trying the following code. Because the data frame is so small, we will just print the output to the *Console* without assigning it to anything.

```
drop_na(data = df, crime_count)
```

Note that if you do not specify a variable, `drop_na()` will just remove any observations with missings in any column.

We can also replace missing values with another value using `replace_na()`, which is also from `tidyr`. Let's say we wanted to just replace missings with zeros. Again, in reality this might not be a good idea! We can do this for any particular column, but in this example, we only have one, so we only need to specify crime_count.

```
replace_na(data = df, list(crime_count = 0))
```

The replacement does not have to be a number. Here, we just assign *refused to answer* to these values.

```
replace_na(data = df, list(crime_count = "refused to answer"))
```

If we were to do things the other way round, we can also replace observed (non-missing) values with missings using `na_if()` in the `dplyr` package. This function is designed to be used within `mutate()` (see Chapter 2) to create a new variable. Here, we just replace crime count values of zero with missings.

```
df %>%
  mutate(na_example = na_if(x =  crime_count, y = 0))
```

2.4. Selecting Specific Rows, Columns, or Cells

Oftentimes in criminological research and data analysis in general, we only need to work with a subset of a dataset. The `dplyr` package offers ways of selecting various subsets of your data. For instance, one can make selections based on rows, certain columns, or even cells. Making a selection based on rows is equivalent to keeping certain *observations* in your dataset, while making a selection based on columns is equivalent to keeping certain *variables* in your dataset. The following code provides some examples of how to use the `slice()`, `filter()`, and `select()` functions from `dplyr` to keep what we need of our data and nothing more. We demonstrate how to use these functions on our data frame, `df`.

2.4.1. Selecting Rows (or Cases/Observations)

```
# Subset the first 100 rows of data using the
## slice() function
first100 <- df %>%
  slice(1:100)

# Alternatively...
first100 <- df[c(1:100),]
```

2.4.2. Selecting Columns (or Variables)

```
# Subset the first 30 variables (or columns) in your data
## frame using the select() function
first30 <- df %>%
  select(1:30)

# Alternatively...
first30  <- df[,c(1:30)]

# Same as above, but using the variable names
first30 <- df %>%
  select(Var1:Var30)
```

You can also use `select()` in conjunction with special functions like `starts_with()`, `ends_with()`, `contains()`, `matches()`, `num_range()`, `one_of()`, and `everything()` to more easily filter out the variables you want to select.

```
# Select only variables beginning with "crime"
crime <- df %>%
  select(starts_with("crime"))

# Select only variables ending with "year2"
crime2 <- df %>%
  select(ends_with("year2"))
```

2.4.3. Selecting Cells

Selecting certain cells in `R` is easy. Note that you can also use this way of specifying cells for recoding.

Here, we are selecting/recoding a cell that falls on the 109th row and in the 4th column.

```
df[109, 4]      # Row 109, column 4

df[109, 4]<-NA # Code cell as missing

df[109, 4]<-99 # Change cell value to 99
```

2.5. Selecting Cases Based on Criteria

We have covered how to go about selecting your subset by rows and columns, but you may also want to subset your data by some other criteria. For instance, you want to examine only males, or only youth, but your samples contain females and senior citizens. If you need to select cases from your data frame that meet certain conditions, you can use the filter() function from dplyr. Assume in the following example that we want to perform an analysis on a sample of recidivists. We can filter on the dummy variable recidivist such that only cases where recidivist is equal to 1 are kept. We also want only adults in our sample, so we can filter on age as well.

```
# Subset rows based on some condition(s) using filter()
adult_recidivist_sample <- df %>%
  filter(recidivist == 1 & age > 17)
```

2.6. Add Columns to a Data Frame

Unless you are lucky, you will sometimes need to merge multiple data sources together for your analyses. The dplyr package allows users to merge multiple data frames together through different *join* functions. Each type of join merges your data a slightly different way.

2.6.1. Inner Join

The inner_join() function keeps only cases that exist in *both* datasets you are merging. This means that if you have an ID for someone in your first dataset, but not in your second, that case will be dropped in the merged version.

```
# Inner join
df3 <- inner_join(df1, df2, by = "ID")
```

2.6.2. Left Join

The left_join() function does not drop ALL unmatched cases, but keeps unmatched cases from the first data frame, and simply assigning it an NA for columns from the second data frame. If the second data frame also had an unmatched case, this would be dropped.

```
# Left join
df3 <- left_join(df1, df2, by = "ID")
```

2.6.3. Right Join

The right_join() function is the same as the left_join() function, except that any unmatched cases from the second dataset are kept this time, and the unmatched cases from the first dataset are dropped.

```
# Right join
df3 <- right_join(df1, df2, by = "ID")
```

2.6.4. *Full Join*

The `full_join()` function returns all of the columns from both datasets and returns a `NA` when there are no matching values.

```
# Full join
df3 <- full_join(df1, df2, by = "ID")
```

2.7. Add Rows to a Data Frame

You may want to add more cases to your data frame rather than adding columns. This can be done by using the `rbind()` function from base `R` where you specify the names of the objects you want to add together (can be vector, matrix, or data frame). To use this function with a data frame, make sure that the variable names in data frames being combined match.

```
# Add rows with rbind()
New_df<-rbind(df1, df2)
```

2.8. Applying Functions to Every Column

If you want to apply a function to all columns in your data frame, you can use one of the `apply()` family of functions from base `R`. Some key functions from this family include `apply()`, `lapply()`, `sapply()`, and `tapply()`.

2.8.1. *Using `apply()`*

The `apply()` function does exactly what it sounds like—it applies a function to an array or matrix. You can choose whether to apply the function to rows, columns, or both. Note that to pass the function to rows, you will use the number 1, and to pass the function to columns, the number 2. This function then returns either a vector, or an array, or list of values.

```
# apply()

# Applies the function mean to all COLUMNS in df
apply(df, 2, mean)

# Applies the function mean to all ROWS in df
apply(df, 1, mean)
```

2.8.2. *Using `lapply()`*

Using the `lapply()` function applies a function to all elements of a list and returns a list of results.

```
# lapply()

# Applies the function mean to the list "mylist"
lapply(mylist, mean)
```

2.8.3. Using sapply()

The `sapply()` function is similar to `lapply()`, except that instead of returning a list, it returns a vector or matrix.

```
# sapply()

# Applies the function mean to the list "mylist"
sapply(mylist, mean)
```

2.8.4. Using tapply()

The `tapply()` function is useful in that it allows users to apply a function to parts of a vector rather than the whole thing. For instance, if you wanted to apply a function to groups within a vector, you can simply specify the vector that you want to apply the function to, the grouping vector, and finally, the function itself. For instance, if you want to calculate the mean number of officers by law enforcement agency type, you could do the following:

```
# tapply()

# Calculates the mean number of officers per agency type
tapply(df$num_officers, df$agencytype, mean)
```

2.9. Calculating Variable Transformations

Sometimes we need to transform our variables before we include them in a statistical model. You can use the mathematical operations discussed in Chapter 1 on most variables in `R` (if your variable is numeric!). The following are merely a few examples of key transformations you may want to make.

2.9.1. Logarithmic Transformation

Use the `log10()` function from base `R` to calculate the base 10 logarithm of a vector.

```
# Create a vector a
a <- c(50, 100, 40, 62, 922, 4000)
a

# transform a using log
b <- log(a)
b
```

2.9.2. *Natural Log*

To perform a natural log transformation on a vector, you can use the `log()` function, also available through base `R`.

```
# Create a vector named "a"
a <- c(50, 100, 40, 62, 922, 4000)
a

# Transform "a" using log()
b <- log(a)
b
```

2.9.3. *Exponentiation*

Exponentiating a value or set of values in `R` is very straightforward. You can perform calculations of values directly in `R` like a calculator.

```
# 10 squared
10^2

# 5 cubed
5^3
```

You can also perform calculations on vectors of values.

```
# create a vector named "c"
c <- c(10, 33, 52, 900, 2246)

# Square "c"
c^2

# Raise "c" to the 4th power
c^4
```

These are just several data transformations you may want to make. Luckily, with the use of `R` objects and vectorized operations, it is relatively easy to transform your data to fit your specific needs.

2.10. Summarize a Data Frame by Groups

With the `dplyr` package, you can summarize variable(s) within groups. For instance, in this example, imagine we want to calculate the mean and standard deviation of inmates' age (AGE) by their gender (GENDER), and the variables are stored in a data frame named `df`.

```
df %>%
group_by(GENDER) %>%
  summarize(mean_age = mean(AGE, na.rm = TRUE),
  sd_age = sd(AGE, na.rm = TRUE))
```

You can choose to store this as a data frame (named `new_df`).

```
new_df <- (df %>%
             group_by(GENDER) %>%
             summarize(mean_age = mean(AGE, na.rm = TRUE),
             sd_age = sd(AGE, na.rm = TRUE)))
```

2.11. Reshaping Data Frames

Reshaping data is a task many analysts must perform at one time or another. How else are you supposed to format your time series data to examine changes in delinquency over time? Though the task itself seems like it may be long and arduous, `R`, and more specifically `tidyr`, can make this process much smoother than what you might first expect.

2.11.1. Into Wide Format

Load the library for `tidyr`, and use the `spread()` function to convert your data into wide format. You just need to specify the data frame you want to reshape (`df`, in this case), as well as the variable that you will convert to multiple variables or column names (intervention_period).

```
# Convert data frame into wide format
wide_df <- df %>% spread(key = intervention_period,
                          value = num_crimes)
```

You can also use the newer approach to reshaping data in `tidyr`, `pivot_wider()`. This function works in a similar fashion, though it is still being updated, unlike the `spread()` function. Rather than specify the *key* and *value* column names, the `pivot_wider()` function allows you to specify the columns that uniquely identify each observation (though the default is that all columns in your data frame will be selected), as well as the columns from which to get the new column names (names_from) and values from (values_from). The following example will create a new data frame with multiple columns beginning with intervention_period and will include the number of crimes in each cell in the appropriate intervention_period column.

```
# Convert data frame into Wide format
wide_df <- pivot_wider(id_cols = id,
                        names_from  = intervention_period,
                        values_from = num_crimes)
```

2.11.2. Into Long Format

If on the flip side we want to take our 20 different variables indicating each intervention period and collapse it into a single column, use the `gather()` function to reshape wide format to long. Remember to specify both the *key* and the *value* columns, or what variables you want to gather *on*, and their values.

```
# Convert data frame into long format using "gather"
long_df <- df %>% gather(key = intervention_period,
                         value = num_crimes)
```

Again, you can also use the newer approach offered by `tidyr`: the `pivot_longer()` function. In the following example, we first need to specify the columns we want to pivot on or make longer (in this case, variables marking the intervention period), then the new column name for the pivoted column names, and then finally the new column name for the values associated with these columns.

```
# Convert data frame into long format using "pivot_longer()"
long_df <- df %>% pivot_longer(cols = starts_with("period"),
names_to = "intervention_period", values_to = "num_crimes")
```

Appendix 3: Formatting

3.1. Changing Classes

3.1.1. To Numeric Class

After importing data into `R`, you will often need to change the class of some of your variables. In the example below, the variable age was stored as a character class, i.e., the numbers are stored as strings rather than numbers. To change the class of the age variable, you can use the base `R` `as.numeric()` function.

```
# Change the "age" variable in the ncvs data frame
## to numeric class
ncvs$age <- as.numeric(ncvs$age)

# Dplyr method to change multiple variables to numeric
# Changes variables x, y, and z to numeric class
ncvs <- ncvs %>%
  mutate_at(vars(x, y, z), list(as.numeric))
```

3.1.2. To Character Class

Sometimes, you may want to change a variable to a character class or string. For example, if you import a dataset (df) that contains a column of zip codes (zip), R may treat this column as a numeric class initially. However, you may want zip code to be treated as a string variable. To change the class of zip, you can use the base R `as.character()` function.

```
# Change the "zip" variable in the data frame to a character
## class variable
df$zip <- as.character(df$zip)

# Dplyr method to change multiple variables to character
# Changes variables x, y, and z to character class
df <- df %>%
   mutate_at(vars(x, y, z), list(as.character))
```

3.1.3. To Factor Class

You may also need to convert variables to a factor class. This can be accomplished with the base R `as.factor()`, or the `as_factor()` function from the `forcats` package. There is a difference between the two functions in how levels are defined. Be sure to review the documentation for whichever function you choose. Let's convert the variable sex, a character variable, to a factor.

```
# Change the "sex" variable in the data frame to a factor
## class variable
df$sex2 <- as.factor(df$sex)

# Forcats method to change variable to a factor
df$sex2 <- df %>% as_factor(sex)
```

3.2. Formatting Dates

As noted in Chapter 3, research in criminology and criminal justice is increasingly making use of longitudinal and time-stamped data. For that reason, it is useful to know how to work with dates in R. There is a specific package in R used for working with dates called `lubridate`. Ensure that you have this package installed and load it using `library()` as you learned in the earlier chapters of this book.

First, let's create a simple example dataset to work with. You will find that a great deal of data, such as police-recorded crime records, come with dates in this kind of format. Here, we will use the popular format of DD-MM-YYYY to denote some specific days of the year in the variable day_fac, with a separate variable count denoting the number of events (e.g., crime counts) on that day.

Remember, we are using *DD-MM-YYYY* format, so the first date is 15 February 2012, and so on.

```
df <- data.frame(
  day_fac = c("15-02-2012","21-01-2012","01-03-2012",
              "01-04-2012","15-04-2012 ","01-12-2012"),
  count = c(54,102,32,57,301,1612)
  )
```

Notice that when we check the class of day_fac, it is a factor. Sometimes when you load in data like this (e.g., using `read_csv()`) it will be treated as a character. Either way, the fact is that R does not know that this variable is a date.

```
class(df$day_fac)
```

One implication of this is that when want to do things like arrange rows by date, R does it inappropriately. For example, it thinks that *01-12-2012* (1 December 2012) comes before *15-02-2012* (15 February 2012).

```
# This will just print the arranged df to your Console
arrange(df, day_fac)
```

Using the `lubridate` package, we can ensure that R treats dates correctly, either by reclassifying an existing variable or creating a new one. Here, the appropriate function from `lubridate` is `dmy()` because we know that the date format is *DD-MM-YYYY*. To retain the original for comparison, we will just create a new variable called day_dmy.

```
df <- df %>%
  mutate(day_dmy = dmy(day_fac))
```

Now when we check the class, it confirms that the new day_dmy variable is a date.

```
class(df$day_dmy)
```

This time, when we arrange by the new date, it gets it right.

```
# This will just print the arranged df to your Console
arrange(df, day_dmy)
```

3.3. Extract Parts of Dates from a String

Perhaps you have a date variable stored as a string, but you really need a column with just one part of the date, such as the year. See the example below for how you can extract a part of a date using a `substr()` function from base R, the `separate()` function from `tidyr`, or by using functions from the `lubridate` package.

```
# Extract parts of the date you need when the date
## is stored as a string
df <- data.frame(
  date = c("01-01-2015", "01-02-2015" , "01-01-2016",
           "01-02-2016"), count = c(100, 200, 300, 400))

# The first value is position in the string you want
## to start your subset
df$year <- substr(df$date, 7, 10)


# The second value is what position in the string you want
## to end your subset
df$month <- substr(df$date, 4, 5)
df$day <- substr(df$date, 1, 2)

# You can also use the tidyr function separate()
df <- df %>% separate(date, c("Month", "Day", "Year"),
                      sep = "-")

# Alternatively, you can transform the string variable to
## date format
# Use the year() function from lubridate

# Transforms string to the month, day, year date format
df <- df %>%
  mutate(date2 = mdy(date))

# Extracts the year from the MDY formatted variable
df <- df %>%
  mutate(year2 = year(date2))
```

The `lubridate` package has many other options depending on the format of your dates. It also has advanced functionality with timings such as hours, seconds (even milliseconds, nanoseconds, and so on), as well as time zones. However, hopefully the above demonstration showcases how important it is to treat dates appropriately in `R` and how useful the `lubridate` package is!

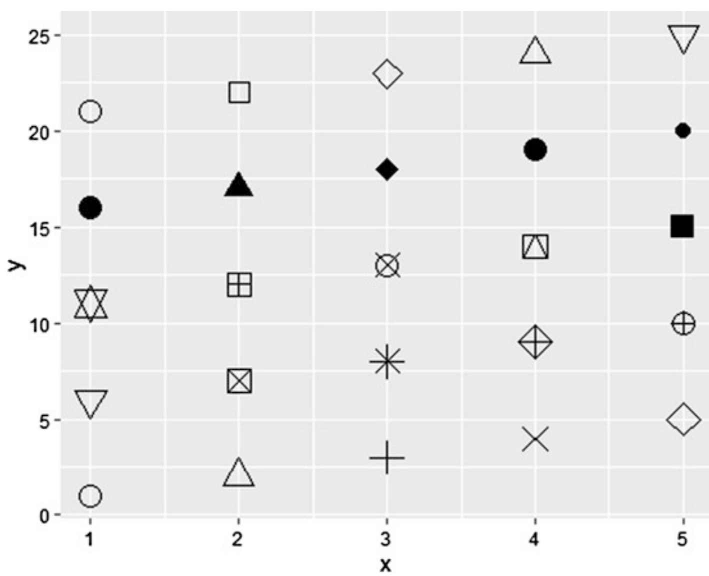
Appendix 4: Pimp My ggplot

4.1. Shape Options

In Chapter 3, we covered data visualization using `ggplot2`. This included the use of geometries such as `geom_line()` and `geom_point()`. We also mapped variables to different aesthetics including `shape` and `linetype`. In doing so, we saw some of the common shapes (e.g., circles and squares) and line types (e.g., dotted and dashed) used to display data. By adapting some code from the `ggplot2` documentation, we can visualize the 25 different shapes available. Note that the position of each shape on the y-axis corresponds to its unique number. So, if you wanted all your data points to be shaped with the + symbol, you would specify `shape = 3`.

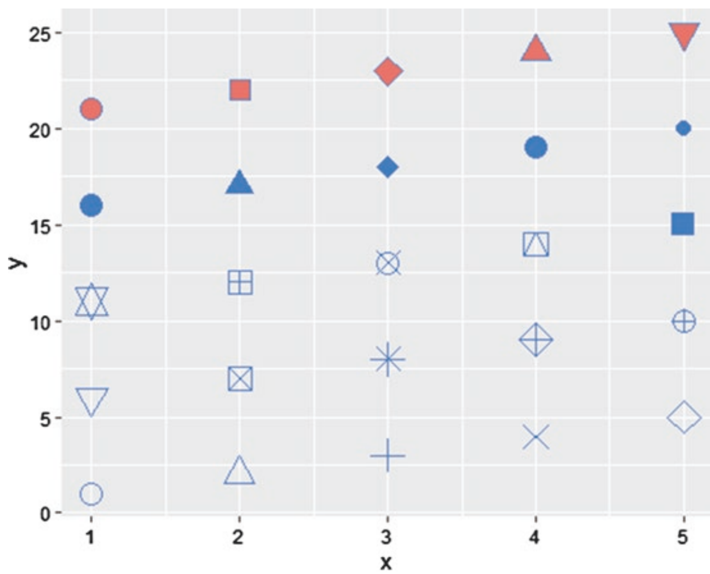
```
points_df <- data.frame(x = 1:5 , y = 1:25, option = 1:25)

ggplot(data = points_df) +
  geom_point(mapping = aes(x = x, y = y, shape = option),
             size = 5) +
  scale_shape_identity()
```



It is worth being aware that shapes might respond differently to additional aesthetics such as `fill` and `color`. This demonstrates important distinctions between shapes that might otherwise appear identical (e.g., 1 and 21).

```
ggplot(data = points_df) +
  geom_point(mapping = aes(x = x, y = y, shape = option),
             size = 5, fill = "salmon",
             color = "dodgerblue") +
  scale_shape_identity()
```

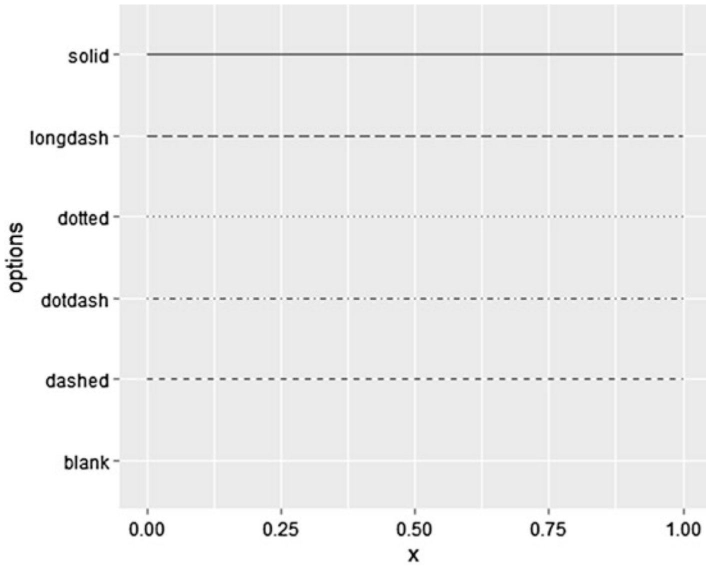


4.2. Line Types

For line types, we can view the names of the options available in the help documentation `?linetype`. There are six options by default (excluding a `blank` one). Like the shapes, these options can be referred to by number (0–6). To take a look at some of these options, we can create a basic data frame containing the line type names and visualize it.

```
lines_df <- data.frame(options = c("blank", "solid",
                                   "dashed", "dotted",
                                   "dotdash", "longdash"))

ggplot(data = lines_df) +
  geom_segment(mapping = aes(x = 0, xend = 1, y = options,
                             yend = options,
                             linetype = options)) +
  scale_linetype_identity()
```



4.3. Font Types

The function `element_blank()` assigns nothing, and as such is often used to remove something (e.g., axis ticks). For instance, to remove the x-axis title, we would add the following to the `theme()` function:

```
theme(axis.title.x = element_blank())
```

The function `element_text()` is used to specify options to text (e.g., font style or size). For instance, to change the y-axis text to size `10`, at a `90` degree angle, in font style `mono`, and in bold type, we would add the following to `theme()`:

```
theme(axis.text.y = element_text(size = 10,
                                 angle = 90,
                                 family = "mono",
                                 face = "bold"))
```

The function `element_rect()` is used to specify options relating to panel borders or backgrounds. To make the plot background pink, for example, we would use the following within `theme()`:

```
theme(plot.background = element_rect(fill = "pink"))
```

The function `element_line()` is for lines, such as the grid lines (e.g., panel grid) of your graphic. So, to change the panel grid line color, we would add:

```
theme(panel.grid = element_line(color = "black"))
```

You may also want to check out the `extrafont` package if you would like more options to change the appearance of the text in `R`.

4.4. Color Options

Colors in `R` can be referenced just as they are in HTML/CSS, where red, green, and blue are represented using hexadecimal (*hex*) values (`00` to `FF`) in a string that starts with a pound symbol, e.g., `#000099`. `R` also has several pre-defined color options that you can use instead by just specifying the name of the color, e.g., `"red"`, `"darkred"`, `"tomato"`, and `"salmon"`. You can obtain a list of these colors simply by running the function `colors()`, which will print the list of color names to your console. The available hex color codes are provided in Fig. A4.1.

Figure A4.1 *Hex Code Color Options*

000000 16	005f00 22	008700 28	00af00 34	00d700 40	00ff00 46	5fff00 82	5fd700 76	5faf00 70	5f8700 64	5f5f00 58	5f0000 52
00005f 17	005f5f 23	00875f 29	00af5f 35	00d75f 41	00ff5f 47	5fff5f 83	5fd75f 77	5faf5f 71	5f875f 65	5f5f5f 59	5f005f 53
000087 18	005f87 24	008787 30	00af87 36	00d787 42	00ff87 48	5fff87 84	5fd787 78	5faf87 72	5f8787 66	5f5f87 60	5f0087 54
0000af 19	005faf 25	0087af 31	00afaf 37	00d7af 43	00ffaf 49	5fffaf 85	5fd7af 79	5fafaf 73	5f87af 67	5f5faf 61	5f00af 55
0000d7 20	005fd7 26	0087d7 32	00afd7 38	00d7d7 44	00ffd7 50	5fffd7 86	5fd7d7 80	5fafd7 74	5f87d7 68	5f5fd7 62	5f00d7 56
0000ff 21	005fff 27	0087ff 33	00afff 39	00d7ff 45	00ffff 51	5fffff 87	5fd7ff 81	5fafff 75	5f87ff 69	5f5fff 63	5f00ff 57

8700ff 93	875fff 99	8787ff 105	87afff 111	87d7ff 117	87ffff 123	afffff 159	afd7ff 153	afafff 147	af87ff 141	af5fff 135	af00ff 129
8700d7 92	875fd7 98	8787d7 104	87afd7 110	87d7d7 116	87ffd7 122	afffd7 158	afd7d7 152	afafd7 146	af87d7 140	af5fd7 134	af00d7 128
8700af 91	875faf 97	8787af 103	87afaf 109	87d7af 115	87ffaf 121	afffaf 157	afd7af 151	afafaf 145	af87af 139	af5faf 133	af00af 127
870087 90	875f87 96	878787 102	87af87 108	87d787 114	87ff87 120	afff87 156	afd787 150	afaf87 144	af8787 138	af5f87 132	af0087 126
87005f 89	875f5f 95	87875f 101	87af5f 107	87d75f 113	87ff5f 119	afff5f 155	afd75f 149	afaf5f 143	af875f 137	af5f5f 131	af005f 125
870000 88	875f00 94	878700 100	87af00 106	87d700 112	87ff00 118	afff00 154	afd700 148	afaf00 142	af8700 136	af5f00 130	af0000 124

d70000 160	d75f00 166	d78700 172	d7af00 178	d7d700 184	d7ff00 190	ffff00 226	ffd700 220	ffaf00 214	ff8700 208	ff5f00 202	ff0000 196
d7005f 161	d75f5f 167	d7875f 173	d7af5f 179	d7d75f 185	d7ff5f 191	ffff5f 227	ffd75f 221	ffaf5f 215	ff875f 209	ff5f5f 203	ff005f 197
d70087 162	d75f87 168	d78787 174	d7af87 180	d7d787 186	d7ff87 192	ffff87 228	ffd787 222	ffaf87 216	ff8787 210	ff5f87 204	ff0087 198
d700af 163	d75faf 169	d787af 175	d7afaf 181	d7d7af 187	d7ffaf 193	ffffaf 229	ffd7af 223	ffafaf 217	ff87af 211	ff5faf 205	ff00af 199
d700d7 164	d75fd7 170	d787d7 176	d7afd7 182	d7d7d7 188	d7ffd7 194	ffffd7 230	ffd7d7 224	ffafd7 218	ff87d7 212	ff5fd7 206	ff00d7 200
d700ff 165	d75fff 171	d787ff 177	d7afff 183	d7d7ff 189	d7ffff 195	ffffff 231	ffd7ff 225	ffafff 219	ff87ff 213	ff5fff 207	ff00ff 201

080808 232	121212 233	1c1c1c 234	262626 235	303030 236	3a3a3a 237	444444 238	4e4e4e 239	585858 240	626262 241	6c6c6c 242	767676 243
eeeeee 255	e4e4e4 254	dadada 253	d0d0d0 252	c6c6c6 251	bcbcbc 250	b2b2b2 249	a8a8a8 248	9e9e9e 247	949494 246	8a8a8a 245	808080 244

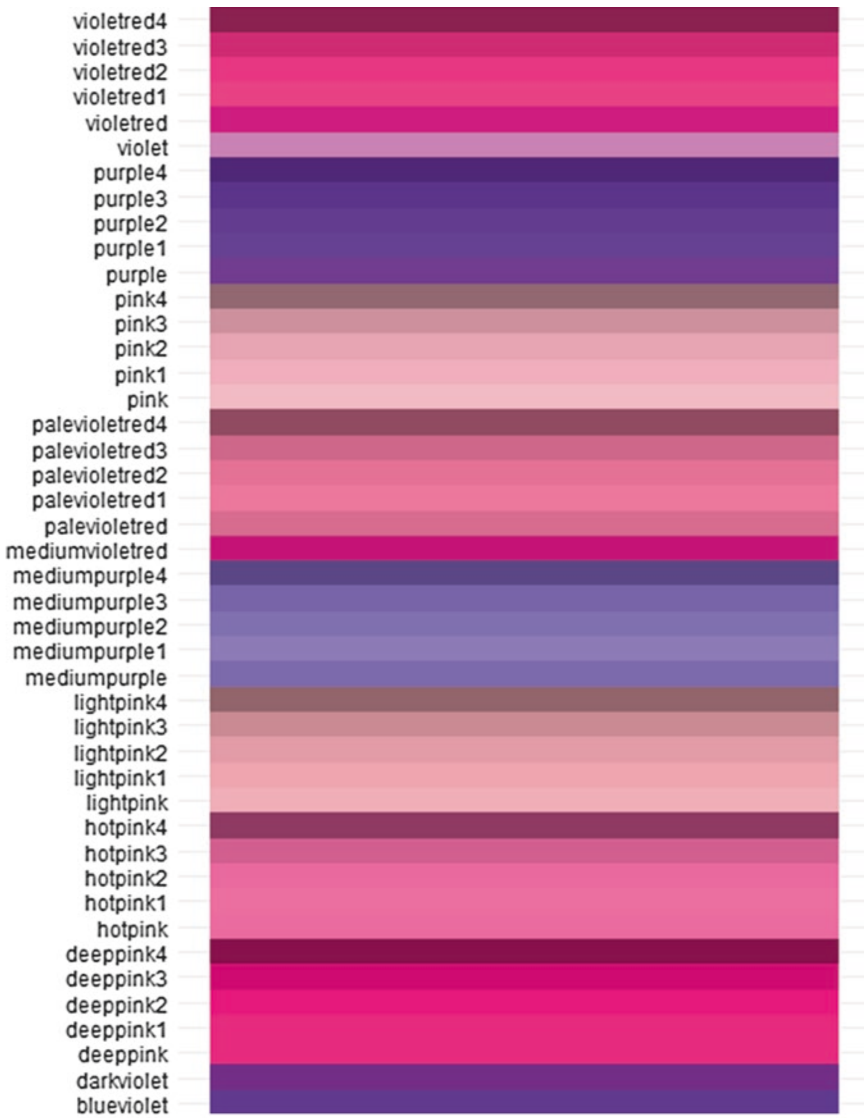
000000 0	cd0000 1	00cd00 2	cdcd00 3	0000ee 4	cd00cd 5	00cdcd 6	e5e5e5 7
7f7f7f 8	ff0000 9	00ff00 10	ffff00 11	5c5cff 12	ff00ff 13	00ffff 14	ffffff 15

4.4.1. `ggplot2` Color Options

You can also visualize the colors themselves using some of the skills picked up in Chapter 3, with some additional tweaks. Here, we just show a sample of colors, because there are far too many (over six hundred!) in total. Remember to ensure that the relevant libraries are loaded before you try this code, such as by using `library(ggplot2)`. For this example, we use `ggplot2`, `stringr`, and `dplyr`.

```
# Pull all colors containing the words pink, violet or purple
colors_df <- data.frame(col_names = colors()) %>%
                filter(str_detect(col_names,
                       "pink|violet|purple"))

# Visualize these colors in a tile plot
ggplot(data = colors_df) +
  theme_minimal() +
  geom_tile(aes(x = col_names, fill =
            as.factor(1:nrow(colors_df)), y = 1)) +
  scale_fill_manual(values = colors_df$col_names) +
  coord_flip() +
  theme(legend.position = "none",
        axis.title = element_blank(),
        axis.text.x = element_blank())
```



4.4.2. Color Palettes

Rather than refer to colors manually by name, we can use functions available within the `scales` package to extract the hex value names for the default `ggplot2` palette, or any other palette available.

```
# Print hex value names (in this example, for six colors)
hue_pal()(6) # default for ggplot2

## [1] "#F8766D" "#B79F00" "#00BA38" "#00BFC4" "#619CFF"
       "#F564E3"

# Specific palette name, e.g., spectral
brewer_pal(palette = "Spectral")(6)
## [1] "#D53E4F" "#FC8D59" "#FEE08B" "#E6F598" "#99D594"
       "#3288BD"
```

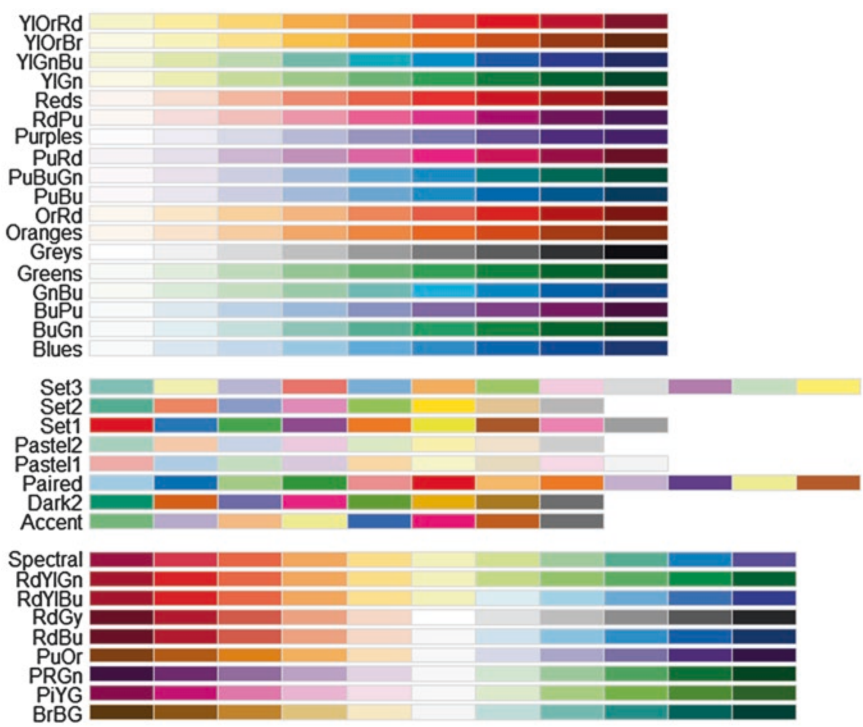
If we are not sure what these colors look like, we can also visualize them, along with the respective hex values.

```
# Visualize hex values with names
show_col(hue_pal()(6)) # default for ggplot2
```


```
# Specific palette name e.g. spectral
show_col(brewer_pal(palette = "Spectral")(6))
```

Or simply visualize all the palettes available, along with their respective palette names, using the `RColorBrewer` package.

```
display.brewer.all()
```



Appendix 5: Saving Output

5.1. Exporting Plots

In Chapter 3, we covered how to explore your data using the visualization tools available in ggplot2. Once you've created a visual with the R environment, you will likely want to save it for use in a paper or presentation. The simplest way to do this is to use the *Export* tab from the *Plots* window in the RStudio environment. One of the downsides of this method is that it is not reproducible. Someone running your code (including your future self) might get the same graphic within R but then export it using different settings (e.g., format, dimensions). For that reason, we recommend using the function ggsave() within ggplot2. It allows you to export your graphics in a way that is reproducible. It is also flexible, with numerous different options around dimensions, formats, and resolution, which saves you a bunch of time when producing lots of different visuals.

To start with, you will want to generate your graphic and assign it to an object. If you want to run through this code, take a look at the first example using police-recorded crime data in Chapter 3.

```
my_plot <- ggplot(data = burglary_df, mapping = aes
                  (x = incscore,
                   y = burglary_count))+ geom_point()
```

You can then input this object into the ggsave() function, using the arguments available within the function to specify our preferences. A brief explanation of each is given using comments in the below code chunk, but you can view the help documentation using ?ggsave to get the full details. Note that we don't specify where to save the file, so by default it will be saved to whatever the current working directory is. Remember that you can check this using getwd(). If you want to specify a working directory, you can either include it before the file name or use the path option within the function.

```
ggsave(plot = my_plot,                 # name of ggplot object
       filename = "my_plot_file.png", # file name
       device = "png",                # device i.e. file format
       units = "cm",                  # units of dimensions
       width = 10,                    # width dimension
       height = 8,                    # height dimension
       dpi = 300)                     # pixel density,
                                      ## i.e., resolution
```

It's worth noting that `ggsave()` will guess the device (i.e., file format) based on the extension used within the `filename` argument. However, in the above example, we have been explicit and stated this using `device = "png"`. There are a number of other formats available (e.g., pdf, tiff, jpeg).

Appendix 6: List of Data Sources and Dataset Names

CHAPTER	DATA SOURCE	FILE NAME(S)
1	NA	NA
2	National Crime Victimization Survey (NCVS)	NCVS lone offender assaults 1992 to 2013.sav
3	2017 crime data from Greater Manchester, England	gmp_2017.csv; gmp_monthly_2017.csv
4	2016 LEMAS-Body Worn Camera Supplement	37302-0001-Data.rda
5	2004 Survey of Inmates in State and Federal Correctional Facilities (SISFCF)	04572-0001-Data.Rda
6	Simulated data	NA
7	Simulated data	NA
8	Simulated data	NA
9	British Crime Survey	*bcs_2007_8_teaching_data_unrestricted.dta*
10	Synthetic data containing information about IQ scores of prisoners	NA
11	National Youth Survey	*nys_1_ID.dta*, *nys_2_ID.dta*
12	Stop and searches carried out in London by police	*stop_search_london.csv*
13	Seattle Neighborhoods and Crime Survey	Seattle_Neighborhoods_Crime_RandomSample.dta
14	Prof. Sharkey et al.'s dataset to study the effect of nonprofit organizations in the levels of crime	sharkey.csv
15	Crime Survey for England and Wales	csew1314_teaching.csv

Appendix 7: Citations to Packages/Software

PACKAGE/ SOFTWARE	CITATION
arm	Gelman, A., & Hill, J. (2007). Data analysis using regression and multi-level hierarchical models (Vol. 1). New York, NY, USA: Cambridge University Press
car	Fox, J., & Weisberg, S. (2019). An R Companion to Applied Regression, Third edition. Sage, Thousand Oaks CA. https://socialsciences.mcmaster.ca/jfox/Books/Companion/
DescTools	Signorell, A., et al. (2020). DescTools: Tools for Descriptive Statistics. R package version 0.99.34, https://cran.r-project.org/package=DescTools
dplyr	Wickham, H., François, R., Henry, L. & Müller, K. (2019). dplyr: A Grammar of Data Manipulation. R package version 0.8.3. https://CRAN.R-project.org/package=dplyr
forcats	Wickham, H. (2019). forcats: Tools for Working with Categorical Variables (Factors). R package version 0.4.0. https://CRAN.R-project.org/package=forcats
Ggally	Schloerke, B., et al. (2020). GGally: Extension to 'ggplot2'. R package version 1.5.0. https://CRAN.R-project.org/package=GGally
ggplot2	Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York
gmodels	Warnes, G.C., Bolker, B., Lumley, T. & Johnson, R.C. (2018). gmodels: Various R Programming Tools for Model Fitting. R package version 2.18.1. https://CRAN.R-project.org/package=gmodels
GoodmanKruskal	Pearson, R. (2016) Goodman Kruskal: association analysis for categorical variables. R package version 0.0.2. https://CRAN.R-project.org/package=GoodmanKruskal
haven	Wickham, H. & Miller, E. (2019). haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files. R package version 2.2.0. https://CRAN.R-project.org/package=haven
here	Müller, K. (2017). here: A Simpler Way to Find Your Files. R package version 0.1. https://CRAN.R-project.org/package=here
labelled	Larmarange, J. (2019). labelled: Manipulating Labelled Data. R package version 2.2.1. https://CRAN.R-project.org/package=labelled
modeest	Poncet, P. (2019). modeest: Mode Estimation. R package version 2.4.0. https://CRAN.R-project.org/package=modeest
moments	Komsta, L. & Novomestky, F. (2015). moments: Moments, cumulants, skewness, kurtosis and related tests. R package version 0.14. https://CRAN.R-project.org/package=moments
mosaic	Pruim, R., Kaplan, D.T., & Horton, N.J. (2017). The mosaic Package: Helping Students to 'Think with Data' Using R. The R Journal, 9(1):77-102
qualvar	Gombin, J. (2018). qualvar: Implements Indices of Qualitative Variation Proposed by Wilcox (1973). R package version 0.2.0. https://CRAN.R-project.org/package=qualvar
R	R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/
readr	Wickham, H., Hester, J. & Francois, R. (2018). readr: Read Rectangular Text Data. R package version 1.3.1. https://CRAN.R-project.org/package=readr

PACKAGE/ SOFTWARE	CITATION
Rstudio	RStudio Team (2016). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA URL http://www.rstudio.com/
sjlabelled	Lüdecke, D. (2020). sjlabelled: Labelled Data Utility Functions. R package version 1.1.3. doi: 10.5281/zenodo.1249215 (URL: https://doi.org/10.5281/zenodo.1249215), URL: https://CRAN.R-project.org/package=sjlabelled
skimr	Waring, E., Quinn, M., McNamara, A., Arino de la Rubia, E., Zhu, H. & Ellis, S. (2020). skimr: Compact and Flexible Summaries of Data. R package version 2.1. https://CRAN.R-project.org/package=skimr
tibble	Müller, K. & Wickham, H. (2019). tibble: Simple Data Frames. R package version 2.1.3. https://CRAN.R-project.org/package=tibble
tidyverse	Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686
tigerstats	Robinson, R. & White, H. (2016). tigerstats: R Functions for Elementary Statistics. R package version 0.3. https://CRAN.R-project.org/package=tigerstats

Appendix 8: Index of R Functions

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
`as.vector()`	Coerce an object into a vector (base R)	82, 88
`attach()`	Commonly used to attach a data frame object for easier access (base R)	79, 81, 88
`attributes()`	Access object attributes, such as value labels (base R)	88, 137, 138, 182, 218, 223, 258, 265
`bind_rows()`	Combine data frame(s) together row-wise (`dplyr`)	97, 106
`BinomCI()`	Compute confidence intervals for binomial proportions (`DescTools`)	128–130, 132, 133
`boxplot()`	Produce a box and whisker plot (base R)	186–188
`c()`	Concatenates elements to create vectors (base R)	14, 16–18, 29, 71, 124–126, 170, 177, 213, 215, 216, 219, 221, 238, 263, 281, 283, 286, 287, 291, 292
`case_when()`	Allows users to vectorize multiple if or if else statements (`dplyr`)	33, 34, 37, 38, 72, 177, 278–280
`cat()`	Combines/concatinates character values and prints them (base R)	162, 165, 168
`ceiling()`	Always round up (base R)	9, 20
`chisq.test()`	Produces the chi-square test (base R)	148–150, 153, 212, 213, 215
`class()`	Check the class of an object (base R)	148–150, 153, 212, 213, 215
`complete.cases()`	Returns only complete cases that do not have NAs (base R)	250, 268
`confint()`	Computes confidence intervals for parameters in a fitted model (base R)	263, 265, 268
`contains()`	Used in conjunction with `select()`, selects only variables that contains a certain string (`dplyr`)	283
`cor.test(…method = "kendall")`	Conducts a Kendall's correlation test (`stats`)	225, 238
`cor.test()`	Obtains correlation coefficient (base R)	220, 221, 232, 233, 235, 236, 238, 244
`cor()`	Produces the correlation of two variables (base R)	232, 233, 235–237, 243, 244
`count()`	Counts the number of occurrences (`dplyr`)	31, 33, 38
`CrossTable()`	Produces contingency tables (`gmodels`)	142–145, 147, 148, 151–153
`cut()`	Divides by the specified interval (base R)	207, 208
`data.frame()`	Create a new data frame object (base R)	17–20, 93, 142, 157, 158, 263, 281, 291–294, 298
`dbinom()`	Find probability of events occurring X number of times (`stats`)	116–118
`detach()`	Turns off the `attach()` function (base R)	79, 88, 277
`diff()`	Computes differences between values in a numeric vector (base R)	84, 85, 87, 88
`dim()`	Check the dimensions of an R object (base R)	16, 20, 63, 137

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
display()	Gives a clean printout of lm, glm, and other such objects (arm)	259, 268
DM()	Computes deviation from the mode (qualvar)	82, 87, 88
dmy()	Creates a date variable in the format of DD-MM-YYYY (lubridate)	291
do()	Loop for resampling (mosaic)	95, 97
drop_na()	Removes observations with missing values (tidyr)	281
element_blank()	Assigns nothing to the component of the graphic it is called in (ggplot2)	295, 298
element_line()	Used to specify options relating to lines (ggplot2)	295
element_rect()	Used to specify options relating to panel borders or backgrounds (ggplot2)	295
element_text()	Refer to a text element in thematic options—see Index (ggplot2)	70–73, 76, 187, 295
ends_with()	Used in conjunction with select(), selects only variables that end with some suffix (dplyr)	283
everything()	Used in conjunction with select(), selects all variables (dplyr)	283
facet_wrap()	Facet graphics by one or more variables (ggplot2)	55, 58, 60, 71
factor()	Creates a factor (base R)	30, 31, 38, 53–55, 57, 58, 60, 139–141, 143, 145, 147, 151, 187, 213, 215, 216, 219, 221, 290, 298
factorial()	Compute the factorial of a numeric vector (base R)	114, 115, 118
fct_explicit_na()	Provides missing values an explicit factor level (forcats)	140, 141, 153
filter()	Subsets a data frame to rows when a condition is true (dplyr)	36–38, 69, 82, 153, 229, 237, 282, 283, 284, 298
fisher.test()	Produces Fisher's exact test (base R)	149, 153
fitted()	Extract fitted values from objects when modeling functions (base R)	193, 194, 208
floor()	Always round down (base R)	9, 20
for()	Initiates a for loop (base R)	111, 112
full_join()	Joins two data frames together, keeping all columns from both data frames and returning an NA when there are no matching values (dplyr)	285
function()	Creates a user-specified function (base R)	162, 164, 165, 168
gather()	Reshapes a data frame to long format (tidyr)	289
geom_bar()	Geometry layer for bar plot (ggplot2)	54, 60
geom_boxplot()	Geometry layer for box plot (ggplot2)	57, 60, 70–72, 187, 188
geom_density()	Geometry layer for density plots (ggplot2)	97, 101, 106, 195, 198, 199

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
`geom_errorbar()`	Draw error bars by specifying maximum and minimum value (`ggplot2`)	106, 130, 132
`geom_histogram()`	Geometry layer for histograms (`ggplot2`)	52, 53, 58, 60, 73, 92, 96
`geom_line()`	Geometry layer for line charts (`ggplot2`)	56, 60, 234, 253, 293
`geom_point()`	Geometry layer for scatterplots (`ggplot2`)	43–47, 50, 51, 55, 57, 59, 60, 103, 130, 234, 235, 252–254, 261, 293, 294, 302
`geom_smooth()`	Geometry layer for smoothed lines (`ggplot2`)	59, 60, 235, 254, 261
`geom_vline()`	Geometry layer for adding vertical lines (`ggplot2`)	92, 96, 101, 103, 106
`get_labels()`	Returns value labels of labelled data (`sjlabelled`)	27, 30, 38
`getwd()`	Returns the current working directory (base `R`)	271, 302, 23
`ggcorr()`	Visualize a correlation matrix (`GGally`)	239–241, 244, 248
`ggpairs()`	Makes a matrix of plots, e.g., correlations, scatterplots (`GGally`)	241–244
`ggplot()`	Initialize a ggplot graphic, i.e., specify data, aesthetics (`ggplot2`)	43–48, 50-55, 57, 59, 60, 70, 71, 73–75, 92, 96, 101, 103, 104 129, 130, 186, 187, 188, 196, 198, 199, 208, 230, 234, 235, 251-253, 261, 267, 293, 294, 298, 299, 302
`ggsave()`	Saves plot as a file (`ggplot2`)	302, 303
`ggtitle()`	Adds a title or subtitle to a graph made using `gglot()` (`ggplot2`)	71–73, 268
`GKtau()`	Conducts the Goodman-Kruskal measure of association (`GoodmanKruskal`)	216, 223, 225
`GoodmanKruskalGamma()`	Conducts Goodman-Kruskal measure of association (`DescTools`)	218, 219, 223, 225
`group_by()`	Group observations by variable(s) for performing operations (`dplyr`)	64, 66, 67, 75, 76, 88, 95, 97, 140, 141, 172, 173, 177, 190, 243, 279, 288
`guides()`	Used to customize plot legend when using `ggplot()` (`ggplot2`)	187, 188, 208
`head()`	Returns the first parts of a vector, matrix, table, or data frame (base `R`)	34, 38, 94, 186
`here()`	Find a project's files based on the current working directory (`here`)	41, 79, 137, 171, 185, 211, 229, 235, 247
`if_else()`	Tests conditions for true or false, taking on values for each (`dplyr`)	102, 106
`inner_join()`	Joins two data frames together, keeping only data that exists in both datasets (`dplyr`)	284

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
`install.packages()`	Installs non-base R packages (base R)	23, 24, 38, 41, 63, 79, 90, 120, 136, 156, 171, 185, 211, 229, 247, 269
`IQR()`	Compute interquartile range (base R)	69, 76
`is.na()`	Returns TRUE when values are missing, FALSE if not (base R)	80–82, 88, 139
`KendallTauB()`	Conducts the Kendall measure of association (DescTools)	220, 221, 223, 225
`kruskal.test()`	Performs a Kruskal-Wallis rank sum test (base R)	202, 208
`labs()`	Specify labels for ggplot object, e.g., title, caption (ggplot2)	49–51, 53, 54, 57, 60, 70–73
`Lambda()`	Conducts the measure of association (DescTools)	216, 223, 225
`lapply()`	Applies a function to all elements of a list, returning a list (base R)	285, 286
`left_join()`	Joins two data frames together, keeping unmatched cases from the first data frame (dplyr)	284
`leveneTest()`	Computes Levene's test for homogeneity of variance across groups (car)	192, 193, 206, 208
`library()`	loads the installed non-base R package (base R)	23, 24, 38, 41, 59, 63, 79, 90, 120, 136, 156, 171, 185, 211, 229, 247, 269, 290
`list()`	Create a list (base R)	16, 20, 282, 289, 290
`lm()`	Fit linear models (base R)	258, 259, 267, 268
`load()`	Loads an R datafile (.R or .Rda) (base R)	63, 79, 272
`log()`	Computes the natural logarithm (base R)	235, 286, 287
`log10()`	Computes common (i.e., base 10) logarithms (base R)	187, 188, 199, 200, 208, 286
`matches()`	Used in conjunction with select(), selects only variables that match a regular expression (dplyr)	283
`matrix()`	Creates a vector with two dimensions (base R)	15, 19, 20
`max()`	Returns the maximum value (base R)	64, 75, 76
`mdy()`	Creates a date variable in the format of MM-DD-YYYY (lubridate)	292
`mean()`	Compute arithmetic mean (base R)	64–67, 75, 76, 87, 88, 92–98, 100, 101, 141, 159, 177, 190, 288
`median()`	Compute the median (base R)	66, 67, 75, 76
`merge()`	Merge datasets by common row or columns names (base R)	180, 182
`min()`	Returns the minimum value (base R)	64, 65, 75, 76
`mlv()`	Compute the mode (modeest)	64, 67, 76, 80, 81

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
`mutate()`	Creates new vectors or transforms existing ones (`dplyr`)	32–34, 37, 38, 59, 72, 102, 153, 177, 182, 278, 279, 280, 282, 291, 292
`n()`	Count observations, within `summarize()`, `mutate()`, or `filter()` (`dplyr`)	32
`na_if()`	Replace non-missing values with missing values (`dplyr`)	282
`names()`	Provides the element names (base R)	16, 20
`nrow()`	Counts the number of rows (base R)	31, 38, 154, 158, 250, 279, 298
`num_range()`	Used in conjunction with `select()`, selects only variables that match a numerical range (`dplyr`)	283
`oneway.test()`	Tests if 2+ samples from normal distributions have same means (base R)	195, 208
`pairwise.t.test()`	Pairwise comparisons between group levels (base R)	203, 208
`par()`	Set graphics parameters such as margins (base R)	71, 76
`paste()`	Combines a series of string text (base R)	112, 114, 118
`pbinom()`	Find cumulative probability of a binomial probability distribution (`stats`)	117, 118
`Phi()`	Conducts the measure of association (`DescTools`)	212–214, 223, 225
`pivot_longer()`	Reshapes a data frame to long format (`tidyr`)	289
`pivot_wider()`	Reshapes a data frame to wide format (`tidyr`)	288
`pnorm()`	Probability of random variable following normaldistribution (base R)	162, 168
`pnormGC()`	Compute probabilities for normal random variables (`tigerstats`)	159, 160, 168
`predict()`	Makes predictions from the results of model fitting functions (base R)	262, 267, 268
`print()`	Prints arguments and returns it invisibly (base R)	15, 20, 111–113
`prop_z_test()`	Function created in Chapter 10 for a single-sample *z*-test for proportions	164, 166, 168
`prop.test()`	Test null hypothesis that proportions in groups are the same (base R)	124–126, 131, 133, 176, 177, 181
`qplot()`	Creates a variety of plots/graphs (base R)	250, 251, 268
`qqline()`	Adds a reference line to Q-Q plot produced by `qqnorm()` (base R)	196, 208
`qqnorm()`	Produces a normal Q-Q plot of the variable (base R)	196, 208
`qqPlot()`	Draws theoretical quantile-comparison plots for variables (`car`)	197, 200, 208
`quantile()`	Compute quantiles as per specified probabilities (base R)	69, 76
`range()`	Compute the minimum and maximum values (base R)	84, 85, 270, 283

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
rbind()	Appends rows to a data frame (base R)	285
read_csv()	Read in comma separated values file (readr)	41, 60, 185, 229, 247, 271, 273, 291
read_delim()	Reads in a delimited file (readr)	272
read_dta()	Imports a .dta Stata file (haven)	137, 153, 171, 211, 274
read_excel()	Reads in an .xls or .xlsx file (readxl)	274
read_json()	Reads in a JSON file (jsonlite)	277
read_sas()	Reads in a SAS format file (haven)	275
read_sav()	Reads in an SPSS .sav file (haven)	275
read_spss()	Imports SPSS .sav files (haven)	24, 38
read_tsv()	Reads in a tab-separated file (readr)	273
read.dbf()	Reads in a .dbf file (foreign)	274
read.mat()	Reads in a Matlab file (rmatio)	276
read.mtp()	Reads in a Minitab file (foreign)	276
read.systat()	Reads in a Systat file (foreign)	276
read.table()	Reads in data in tabular format (base R)	275
recode()	Replaces values of a integer/factor variable	172, 182
remove_labels()	Removes value labels from a variable (sjlabelled)	28, 29, 38
remove_var_label()	Removes a variable's label (labelled)	28, 38
replace_na()	Replaces missing values with another value (tidyr)	281, 282
require()	Attempts to load a package in R, returning a logical value of whether the attempt was successful or not (base R)	269
resid()	Extract residuals from objects returned by modeling functions (base R)	193, 194, 196, 208
return()	Used in functions to tell R what to return/print for the user (base R)	162, 168
right_join()	Joins two data frames together, keeping unmatched cases from the second data frame (dplyr)	284
rm()	Remove object from R environment (base R)	229, 244
rnorm()	Create synthetic normally distributed data (base R)	92, 93, 106, 157, 158
round()	Rounds to nearest whole number or specified number of decimals (base R)	23, 94, 157, 158, 234, 253
sample()	Randomly sample from a vector or data frame (mosaic)	32, 94, 95, 97, 99, 100, 106
sapply()	Applies a function over a vector or list (base R)	285, 286
save()	Saves an R data file (base R)	272
scale_color_brewer()	Default color scheme options (ggplot2)	46, 48, 50, 51, 60
scale_color_viridis_d()	Colorblind-friendly palettes from viridis package (ggplot2)	48
scale_fill_discrete()	Specify fill of discrete aesthetics, e.g., color palette (ggplot2)	97, 106
scale_y_log10()	Log scales the y-axis on your chart (ggplot2)	187, 188, 208

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
`scale()`	Mean centers or re-scales a numeric variable (base R)	158, 168
`ScheffeTest()`	Scheffé's test for pairwise and otherwise comparisons (`DescTools`)	204, 208
`sd()`	Computes standard deviation of a numeric vector (base R)	86, 88, 99–102, 159, 288
`select()`	Select columns to retain or drop (`dplyr`)	32, 35, 36, 38, 229, 234, 282, 283
`separate()`	Separates a string by the given separator (`tidyr`)	292
`set.seed()`	Random number generator start point (base R)	93, 106
`setRepositories()`	Sets the repository from which R should search for a package (base R)	270
`setwd()`	Sets the working directory (base R)	24, 271
`single_t_test()`	Function created in Chapter 10 for single-sample *t*-tests for means	165, 166, 168
`skewness()`	Calculate degree of skewness in a numeric vector (`modeest`)	74–76
`skim()`	Provide summary statistics specific to object class (`skimr`)	67, 76, 80, 84, 173
`slice()`	Select rows based on their position in the data frame (`dplyr`)	36, 38, 282
`SomersDelta()`	Conducts Somers' measure of association (`DescTools`)	222, 223, 225
`spread()`	Reshapes a data frame to wide format (`tidyr`)	288
`sqrt()`	Finds the square root (base R)	9, 20, 100, 162, 164–167
`starts_with()`	Used in conjunction with `select()`, selects only variables that start with some prefix (`dplyr`)	283, 289
`str()`	Returns internal structure of an R object (base R)	88, 142
`StuartTauC()`	Conducts the Kendall measure of association (`DescTools`)	221, 225
`substr()`	Selects part of a string (base R)	291, 292
`sum()`	Sum values in a vector (base R)	67, 139, 177
`summarize()`	Create new summary variable(s), e.g., counts, mean (`dplyr`)	82, 88, 95, 97, 99, 100, 104, 140, 141, 153, 172, 177, 190, 243, 288
`summary.lm()`	Summary method for class `lm` (base R)	205, 208, 265
`summary()`	Produce summary of model results (base R)	69, 76, 84, 86, 191, 249, 258, 259, 261, 263, 265–267
`summary()$coefficients`	Extract coefficients only from summary (base R)	268
`summary()$r.squared`	Extract R squared only from summary (base R)	267, 268

FUNCTION	DESCRIPTION (PACKAGE)	PAGE #S
`symbox()`	Transforms x to a series of selected powers and displays box plots (`car`)	199, 208
`t.test()`	Performs one and two sample *t*-tests on vectors of data (base R)	175, 181, 182, 203
`table()`	Generates a frequency table (base R)	27, 30, 53, 82, 83, 102, 102, 139, 140, 142, 150, 189, 213, 215, 216, 219, 221, 276
`tapply()`	Applies a function to parts of a vector (base R)	285, 286
`theme_bw()`	The traditional dark-on-white `ggplot` theme (`ggplot2`)	187, 188, 195, 199, 208
`theme_minimal()`	Default minimalist theme for ggplot graphics (`ggplot2`)	50, 51, 60, 130, 298
`theme()`	Customize ggplot graphics (`ggplot2`)	50, 60, 70–73, 187, 188, 295, 298
`TukeyHSD()`	Implements Tukey's honest significant difference method (base R)	204, 208
`var_label()`	Returns or sets a variable label (`labelled`)	27–29, 37, 38
`var.test()`	Performs an *F*-test to compare the variances of two samples from normal populations (base R)	174, 181, 182
`var()`	Computes variance (base R)	85, 86, 88
`View()`	View data in new window (base R)	17, 20, 25, 34, 35, 37, 42, 63, 94, 137, 171, 180, 211, 277
`vif()`	Calculate the variance inflation for OLS or other linear models (`car`)	266, 268
`which()`	Provides the position of the elements such as in a row (base R)	158
`while()`	Initiates a while loop (base R)	114, 118
`with()`	Evaluates an expression, often used to specify the data you want to use (base R)	142, 143, 145, 147, 151, 153
`write_csv()`	Writes a comma-separated file (`readr`)	273
`write_delim()`	Writes a delimited file (`readr`)	272
`write_dta()`	Writes a Stata .dta file (`haven`)	274
`write_json()`	Writes a JSON file (`jsonlite`)	277
`write_sas()`	Writes a SAS format file (`haven`)	275
`write_sav()`	Writes an SPSS .sav file (`haven`)	275
`write_tsv()`	Writes a tab-separated file (`readr`)	273
`write.dbf()`	Writes a .dbf file (`foreign`)	274
`write.mat()`	Writes a Matlab (MAT) file (`rmatio`)	276
`write.xlsx()`	Writes an Excel file (.xlsx) (`openxlsx`)	274
`year()`	Extracts the year from a date (`lubridate`)	292
`z_test()`	Function created in Chapter 10 for a single-sample *z*-test	161, 162, 164–166, 168

Glossary

68-95-99.7 rule Empirical rule that states that 68% of the cases in a normal distribution should fall within 1 standard deviation of the mean (so within a z-score of -1 and +1); 95% of the cases in the distribution should fall within 2 standard deviations of the mean (so within a z-score of -2 and +2); and 99.7% of the cases in the distribution should fall within 3 standard deviations of the mean (so within a z-score of -3 and +3). In the real world, you will likely not find a distribution where this rule is exact.

Aesthetics Describe visual characteristics that represent the data.

Arrangements The different ways events can be ordered and result in a single outcome. For example, there is only one arrangement for gaining the outcome of ten heads in ten tosses of a coin. There are, however, ten different arrangements for gaining the outcome of nine heads in ten tosses of a coin.

Array A three-dimensional data structure that can contain homogenous elements (of the same class).

Assignment operators Symbols used to make assignations to objects.

Atomic vector A one-dimensional data structure that can contain homogeneous elements (of the same class).

Bell Curve See Gaussian distribution.

Binomial distribution The probability or sampling distribution for an event that has only two possible outcomes.

Binomial formula The means of determining the probability that a given set of binomial events will occur in all its possible arrangements.

Bivariate regression A technique for predicting change in a dependent variable using one independent variable.

Bonferroni correction A post-hoc pairwise comparison of means that controls the type I error rate by dividing the selected α-level by the number of pairwise comparisons made.

Central limit theorem A theorem that states: "If repeated independent random samples of size N are drawn from a population, as N grows large, the sampling distribution of sample means will be approximately normal." The central limit theorem enables the researcher to make inferences about an unknown population using a normal sampling distribution.

A. Wooditch et al., *A Beginner's Guide to Statistics for Criminology and Criminal Justice Using R*, https://doi.org/10.1007/978-3-030-50625-4

Chi-square statistic The test statistic resulting from applying the chi-square formula to the observed and expected frequencies for each cell. This statistic tells us how much the observed distribution differs from that expected under the null hypothesis.

Coefficient of variation (CV) A measure of dispersion calculated by dividing the standard deviation by the mean.

Comments Code annotations that are not interpreted by `R`.

Concordant pairs of observations Pairs of observations that have consistent rankings on two ordinal variables.

Confidence interval An interval of values around a statistic (usually a point estimate). If we were to draw repeated samples and calculate a 95% confidence interval for each, then in only 5 in 100 of these samples would the interval fail to include the true population parameter. In the case of a 99% confidence interval, only 1 in 100 samples would fail to include the true population parameter.

Contingency table A tabular way of viewing the relationship between categorical variables (also referred to as cross tabs).

Covariation A measure of the extent to which two variables vary together relative to their respective means. The covariation between the two variables serves as the numerator for the equation to calculate Pearson's r.

Cramer's V A measure of association for two nominal variables that adjusts the chi-square statistic by the sample size. V is appropriate when at least one of the nominal variables has more than two categories.

Data Information used to answer a research question; typically will be stored in a data frame. Data (plural) are made up of numerous datum (singular).

Data frame A data structure that is defined by the number of rows and columns.

Data transformation An adjustment of data to a different unit or scale (normally to deal with normality issues).

Dependent sample *t*-test A test of statistical significance that is used when two samples are not independent.

Dependent variable (Y) The variable assumed by the researcher to be influenced by one or more independent variables.

Directional hypothesis A research hypothesis that indicates a specific type of outcome by specifying the nature of the relationship that is expected.

Discordant pairs of observations Pairs of observations that have inconsistent rankings on two ordinal variables.

Environment Where objects are stored.

Eta squared The proportion of the total sum of squares that is accounted for by the between sum of squares. Eta squared is sometimes referred to as the percent of variance explained.

Expected frequency The number of observations one would predict for a cell if the null hypothesis were true.

External validity The extent to which a study sample is reflective of the population from which it is drawn. A study is said to have high external validity when the sample used is representative of the population to which inferences are made.

***F*-distribution** A continuous probability distribution used as the null distribution in ANOVA.

Gamma (γ) PRE measure of association for two ordinal variables that uses information about concordant and discordant pairs of observations within a table. Gamma has a standardized scale ranging from -1.0 to 1.0.

Gaussian distribution Normal distribution or bell curve.

Geom Abbreviation for geometries from the `ggplot2` package.

Geometries Describe the objects that represent the data.

Goodman and Kruskal's lambda (λ) PRE measure of association for two nominal variables that uses information about the modal category of the dependent variable for each category of the independent variable. Lambda has a standardized scale ranging from 0 to 1.0.

Goodman and Kruskal's tau (τ) PRE measure of association for two nominal variables that uses information about the proportional distribution of cases within a table. Tau has a standardized scale ranging from 0 to 1.0. For this measure, the researcher must define the independent and dependent variables.

Heteroscedasticity A situation in which the variances of scores on two or more variables are not equal. Heteroscedasticity violates one of the assumptions of the parametric test of statistical significance for the correlation coefficient.

Independent Describing two events when the occurrence of one does not affect the occurrence of the other.

Independent sample *t*-test A test of statistical significance that examines the difference observed between the means of two unrelated samples.

Independent variable (X) A variable assumed by the researcher to have an impact on the value of the dependent variable, Y.

Index of qualitative variation (IQV) A measure of dispersion calculated by dividing the sum of the possible pairs of observed scores by the sum of the possible pairs of expected scores (when cases are equally distributed across categories).

Inferential statistics A broad area of statistics that provides the researcher with tools for making statements about populations on the basis of knowledge about samples. Inferential statistics allow the researcher to make inferences regarding populations from information gained in samples.

Interval/ratio variables Numeric variables with equal intervals between values; functionally the same, yet ratio-level variables have a true zero.

Kendall's tau Measures the strength and direction of two rank-ordered variables on a standardized scale between 0 and 1.0, whereby higher values indicate a stronger relationship.

Kendall's τ_b PRE measure of association for two ordinal variables that uses information about concordant pairs, discordant pairs, and pairs of observations tied on both variables examined. τ_b has a standardized scale ranging from −1.0 to 1.0 and is appropriate only when the number of rows equals the number of columns in a table.

Kendall's τ_c A measure of association for two ordinal variables that uses information about concordant pairs, discordant pairs, and pairs of observations tied on both variables examined. τ_c has a standardized scale ranging from −1.0 to 1.0 and is appropriate when the number of rows is not equal to the number of columns in a table.

Kruskal-Wallis test A nonparametric test of statistical significance for multiple groups, requiring at least an ordinal scale of measurement.

Levene's test A test of the equality of variances.

Linear relationship An association between two variables whose joint distribution may be represented in linear form when plotted on a scatter diagram.

List A one-dimensional data structure that can contain heterogenous elements (of different classes).

Logical operators Boolean operators that return `TRUE` or `FALSE`.

Marginal The value in the margin of a table that totals the scores in the appropriate column or row.

Matrix A specific type of array that has at least two columns and two rows and can contain homogeneous elements (of the same class).

Mean A measure of central tendency calculated by dividing the sum of the scores by the number of cases.

Measures of central tendency Descriptive statistics that allow us to identify the typical case in a sample or population. Measures of central tendency are measures of typicality.

Median A measure of central tendency calculated by identifying the value or category of the score that occupies the middle position in the distribution of scores.

Mode A measure of central tendency calculated by identifying the score or category that occurs most frequently.

Multicollinearity Condition in a multivariate regression model in which independent variables examined are very strongly intercorrelated. Multicollinearity leads to unstable regression coefficients

Multiple comparisons problem The problem associated with the chance of obtaining a false-positive (type I error) increase as the number of comparisons increase.

Multiplication rule The means for determining the probability that a series of events will jointly occur.

Nominal variables Categorical, unordered variables.

Non-directional hypothesis A research hypothesis that does not indicate a specific type of outcome, stating only that there is a relationship or a difference.

Nonparametric tests Tests that do not make an assumption about the distribution of the population; also called distribution-free tests.

Normal distribution A bell-shaped frequency distribution, symmetrical in form. Its mean, mode, and median are always the same. The percentage of cases between the mean and points at a measured distance from the mean is fixed.

Null hypothesis A statement that reduces the research question to a simple assertion to be tested by the researcher. The null hypothesis normally suggests that there is no relationship or no difference.

Object A specialized data structure; everything in R is an object.

Observed frequency The observed result of the study, recorded in a cell.

OLS regression See ordinary least squares regression analysis.

One-way analysis of variance (ANOVA) A parametric test of statistical significance that assesses whether differences in the means of several samples (groups) can lead the researcher to reject the null hypothesis that the means of the populations from which the samples are drawn are the same.

Ordinal variables Categorical, ordered variables.

Ordinary least squares regression analysis A type of regression analysis in which the sum of squared errors from the regression line is minimized.

Outliers A single or small number of exceptional cases that substantially deviate from the general pattern of scores.

Packages Modules that expand what R can do.

Parametric tests Tests that make an assumption about the shape of the population distribution.

Pearson's correlation coefficient See Pearson 's *r*.

Pearson's *r* A commonly used measure of association between two variables. Pearson's r measures the strength and direction of linear relationships on a standardized scale from –1.0 to 1.0.

Percent of variance explained (R^2) A measure for evaluating how well the regression model predicts values of Y. It represents the improvement in predicting Y that the regression line provides over the mean of Y.

Phi (φ) A measure of association for two nominal variables that adjusts the chi-square statistic by the sample size. Phi is appropriate only for nominal variables that each has two categories.

Population The universe of cases that the researcher seeks to study. The population of cases is fixed at a particular time (e.g., the population of the United States). However, populations usually change across time.

Population distribution The frequency distribution of a particular variable within a population.

Project A self-contained working directory.

Proportional reduction in error (PRE) The proportional reduction in errors made when the value of one measure is predicted using information about the second measure.

QQ-plot Used to check for normality of data, plots the correlation between the sample and a normal distribution.

R A language and free software environment used for statistical computing.

R Script Where R programming code is written and stored.

Range A measure of dispersion calculated by subtracting the smallest score from the largest score. The range may also be calculated from specific points in a distribution, such as the 5th and 95th percentile scores.

Regression coefficient (*b*) A statistic used to assess the influence of an independent variable, X, on a dependent variable, Y. The regression coefficient b is interpreted as the estimated change in Y that is associated with a one-unit change in X.

Regression error (*e*) The difference between the predicted value of Y and the actual value of Y.

Regression line The line predicting values of Y. The line is plotted from knowledge of the Y-intercept and the regression coefficient.

Regression model The hypothesized statement by the researcher of the factor or factors that define the value of the dependent variable, Y. The model is normally expressed in equation form.

Reproducibility When there is a record of one's research such that these steps can be repeated by others and the findings reproduced.

Residual An index of the relative deviation of the observed frequency from the expected frequency for a cell of a contingency table. It is useful for guiding the interpretation of an association between two nominal variables.

RStudio An integrated development environment (IDE) designed specifically for R.

Sample A set of actual observations or cases drawn from a population.

Sampling distribution A distribution of all the results of a very large number of samples, each one of the same size and drawn from the same population under the same conditions. Ordinarily, sampling distributions are derived using probability theory and are based on probability distributions.

Sample statistic A characteristic of a sample—for example, the mean number of previous convictions in a random sample of 1,000 prisoners.

Scatterplot A graph whose two axes are defined by two variables and upon which a point is plotted for each subject in a sample according to its score on the two variables.

Scheffé's test A multiple comparison test that accounts for family-wise error rate by weighting the test statistic by the mean squared error, between-samples degrees of freedom, and group sizes.

Single-sample *t*-test A test of statistical significance that is used to examine whether a sample is drawn from a specific population with a known or hypothesized mean. In a *t*-test, the standard deviation of the population to which the sample is being compared is unknown.

Single-sample *z*-test A test of statistical significance that is used to examine whether a sample is drawn from a specific population with a known or hypothesized mean. In a *z*-test, the standard deviation of the population to which the sample is being compared either is known or—as in the case of a proportion—is defined by the null hypothesis.

Somers' D PRE measure of association for two ordinal variables that uses information about concordant pairs, discordant pairs, and pairs of observations tied on the independent variable. Somers' D has a standardized scale ranging from −1.0 to 1.0.

Spearman's correlation coefficient See Spearman 's rho.

Spearman's rho (r_s) A measure of association between two rank-ordered variables. Spearman's r measures the strength and direction of linear relationships on a standardized scale between −1.0 and 1.0.

Standard deviation A measure of dispersion calculated by taking the square root of the variance.

Standard deviation unit A unit of measurement used to describe the deviation of a specific score or value from the mean in a *z* distribution.

Standard error The standard deviation of a sampling distribution.

Synthetic data Computer-generated data.

Test for equality of variance An *F*-test used to assess the null hypothesis that the two population variances are equal.

Themes Customizations that can alter the general appearance of a plot.

Tibble Modern version of base R's data frame (simpler and more user-friendly) that is from the *tidyverse* package.

Tied pairs of observations (ties) Pairs of observation that have the same ranking on two ordinal variables.

Tukey's honestly significant difference (HSD) A parametric test of statistical significance, adjusted for making pairwise comparisons. The HSD test defines the difference between the pairwise comparisons required to reject the null hypothesis.

Type I error Also known as alpha error and false positive. The mistake made when a researcher rejects the null hypothesis on the basis of a sample statistic (i.e., claiming that there is a relationship) when in fact the null hypothesis is true (i.e., there is actually no such relationship in the population).

Variance (s^2) A measure of dispersion calculated by adding together the squared deviation of each score from the mean and then dividing the sum by the number of cases.

Variation ratio A measure of dispersion calculated by subtracting the proportion of cases in the modal category from 1.

Welch's ANOVA ANOVA test for when the equality of variances assumption (homoscedasticity) is not met.

Y-intercept (b_0) The expected value of Y when X = 0. The Y-intercept is used in predicting values of Y.

***z*-score** Score that represents an observation in standard deviation units from the mean.

Index

A. Wooditch et al., *A Beginner's Guide to Statistics for Criminology and Criminal Justice Using R*, https://doi.org/10.1007/978-3-030-50625-4