Exploratory Data Analysis in R with Tidyverse

Matt Steele

2023-03-29

Contents

Continue Learning with these Resources

Tidyverse Documentation

```
browseURL("https://www.tidyverse.org/")
```

R download and documentation

```
browseURL("https://cran.r-project.org/")
```

RStudio download and documentation

```
browseURL("https://rstudio.com/")
```

O'Reilly Learning Platform

```
browseURL("https://databases.lib.wvu.edu/connect/1540334373")
```

R Programming for Statistics and Data Science

```
browseURL("https://libwvu.on.worldcat.org/oclc/1062397089")
```

Getting Started

Start a New File

You can create new files (R Scripts or RMarkdown files) that allow you to document your code and the outputs of your code.

- File > New File > **Type of Document you want to create**

Set your Preferences

You can set your preferences in RStudio.

- Tools > Global Options

Setting your working directory

The working directory in R is the folder where you are working. Hence, it's the place (the environment) where you have to store your files of your project in order to load them or where your R objects will be saved.

Function: getwd()

See the current directory you are in

```
#HELP FOR getwd()

?getwd
```

```
# USING THE getwd() function

getwd()
```

Function: setwd(path)

You can set your working directory in RStudio by going to

- Session > Set Working Directory > Choose Directory
- Choose the folder you want to work out of

After you set your working directory, save the path by copy and pasting the file path from the console area into the source area using the setwd() function

```
# Remember to save your working directory path to your script or markdown file

setwd("C:/Users/Matt/Documents/RWorkshop Development/workshop_eda_tidyverse-main")
```

Tidyverse Package

Tidyvese is a collection of packages focused on data analysis and data visualizations that share an underlying design philosophy, grammar, and data structures.

Install the Tidyvese Package

You must install a package before you can call it. But you only need to install it one time.

```
#INSTALL THE TIDYVERSE PACKAGE

install.packages("tidyverse")
```

Load Packages

For every new session that you want to use tidyverse functions, you must load it.

```
library(tidyverse) # you need to call the library during each session
```

Packages in Tidyverse

GGPlot2 data visualization functions

```
browseURL("https://ggplot2.tidyverse.org/")
```

tibble lighter and more user-friendly version of data.frame()

```
browseURL("https://tibble.tidyverse.org/")
```

tidyr create tidy and meaningfully arranged data

```
browseURL("https://tidyr.tidyverse.org/")
```

readr better importation of data into R

```
browseURL("https://readr.tidyverse.org/")
```

purrr better functional programming

```
browseURL("https://purrr.tidyverse.org/")
```

dplyr data manipulation tools

```
          browseURL("https://dplyr.tidyverse.org/")
```

Tibble

tibble package

The tibble allows you load a lighter and more user-friendly version of data.frame()

```
mtcars

mtcars <- as_tibble(mtcars)
  mtcars
```

Readr

readr package

The readr package allows for a better importation of data and also allows you to import Excel and Google Sheets files.

Function: read_csv()

Read csv file. Automatically creates a tibble data frame.

```
coalProd <- read_csv("coalpublicproduction.csv")
spec(coalProd)
```

Argument: col_types =()

change the values of variables

```
coalProd <- read_csv("coalpublicproduction.csv", col_types = list(
  Year = col_date("%Y"),
  MSHA_ID = col_character(),
  Mine_Name = col_character(),
  Mine_State = col_factor(),
  Mine_County = col_factor(),
  Mine_Status = col_factor(ordered = T),
  Mine_Type = col_factor(),
  Company_Type = col_factor(),
  Operation_Type = col_factor(),
  Operating_Company = col_factor(),
  Operating_Company_Address = col_character(),
  Union_Code = col_character(),
  Coal_Supply_Region = col_factor(),
  Production_short_tons = col_double(),
  Average_Employees = col_double(),
  Labor_Hours = col_double()
), show_col_types = F)

str(coalProd)
```

Hint: Use a function from a specific package

Because there can be crossover in the naming conventions of functions from packages, you can tell R specifically what package you want to call the function from using

- syntax: *package_name::function_name()*

```
#LINE 277

#you can also read files from the internet
```

```
bigfoot <- readr::read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2
```

Function: read_excel()

read a .xlsx file

```
dBoys <- readxl::read_excel("doughboys_episode_data.xlsx", sheet = 1)
dBoys
```

Function: write_csv()

exporting data and saving change made to a data frame to your working directory

```
starwars

write_csv(starwars, "starwars.csv")
```

Pipe Operator - %>%

The pipe operator allows you to separate and unite individual operations on an object

- CTRL + SHIFT + M (windows)
- CMD + SHIFT + M (mac)

data %>% operation A %>% operation B

Dplyr

dplyr package

manipulate and clean data in a data frame

Function: na.omit()

remove absent/missing data data

```
bigfoot_clean <- na.omit(bigfoot)
bigfoot_clean
```

Function: arrange()

sort data frame by variables

```
coalProd_by_name <- arrange(coalProd, desc(Mine_Name))
  view(coalProd_by_name)

coalProd_by_state <- arrange(coalProd, (Mine_State), desc(Production_short_tons))
  view(coalProd_by_state)
```

Function: filter()

subsets observations in a variable according to criteria

```r
# R is character and case-sensitive. Make sure your filter criteria matches the observations exactly

coalProd_wv <- filter(coalProd, Mine_State == "West Virginia")
coalProd_wv
```

Hint: Boolean operators in R R accepts Boolean operators which will allow you to add criteria and conditions to your filters

```
|     is    OR
&     is    AND
==    is    EQUAL
>     is    LESS THAN
<     is    GREATER THAN
>=    is    LESS THAN OR EQUAL TO
<=    is    GREATER THAN OR EQUAL TO
!=    is    NOT EQUAL TO
```

```r
coalProd_wv_labor <- filter(coalProd, Mine_State == "West Virginia" & Labor_Hours <= 5000)

mean(coalProd_wv_labor$Production_short_tons)
```

Function: select()

keeps ONLY selected variables/columns.

```r
coalProd_wv_status <- select(coalProd_wv, Mine_Name, Mine_County, Mine_Status)
coalProd_wv_status
```

Function: mutate()

create a new variable and append it to the data frame.

```r
bigfoot_clean <- mutate(bigfoot_clean, temperature_low_celcius = (temperature_low - 32) * 5/9)

head(bigfoot_clean$temperature_low_celcius, 10)
```

Hint: Overwriting versus creating new variables If your created variable name is the same as an existing variable in the data frame, the created varaible's will override the observations of the existing variable.

Recode observations using mutate()

you can use the mutate() function to change observations in a variable as well

```r
summary(coalProd$Mine_Type)

coalProd <- mutate(coalProd, Mine_Type_num = recode(Mine_Type, "Refuse" = 1, "Surface" = 2, "Undergroun

head(coalProd$Mine_Type_num, 10)
```

you can use the mutate() function to change numeric variables to categorical variable with the cut() function

```r
coalProd <- mutate(coalProd, Avg_Empl_cat = cut(Average_Employees, breaks=c(-Inf, 15, 500, Inf), labels

head(coalProd$Avg_Empl_cat, 10)
```

you can use the mutate() function to recode missing variables

```r
bigfoot_filter <- bigfoot %>%
  mutate(moon_phase = ifelse(is.na(moon_phase), 999, moon_phase)) %>%
  mutate(temperature_high = ifelse(is.na(temperature_high), 999, temperature_high)) %>%
  filter(moon_phase != "999", temperature_high != "999")

view(bigfoot_filter)
```

Function: sample_n() and sample_frac()

randomly sample observations from the data frame

```r
bigfoot_sample <- sample_n(bigfoot, 50, replace = F)
bigfoot_sample
```

Function: group_by() and summarise()

the function group_by() allows you group a categorical variable and create a new table. It can be used in combination with the summarise() function to get a summary of a group.

```r
coalProd_wv_by_county <- coalProd_wv %>% group_by(Mine_County) %>%
  summarise(N = n(),
            mean = mean(Production_short_tons),
            total = sum(Production_short_tons))

write_csv(coalProd_total_by_state, "coalproduction_bystate.csv")

# can also add the arrange

#coalProd_wv_by_county <- arrange(coalProd_wv_by_county, desc(total))
```

Descriptive Statistics

```r
summary(bigfoot_clean) # full data frame

summary(coalProd_wv$Production_short_tons) # subset variable

mean(coalProd_wv$Production_short_tons) # average of observed cases
median(coalProd_wv$Production_short_tons) # the middle number(s) of observed cases

min(coalProd_wv$Production_short_tons) # lowest number in observed cases
max(coalProd_wv$Production_short_tons)  # the middle number(s) of observed cases
range(coalProd_wv$Production_short_tons) # most commonly occurring number of observed cases

quantile(coalProd_wv$Production_short_tons, .25) #1st
quantile(coalProd_wv$Production_short_tons, .75) #2nd
fivenum(coalProd_wv$Production_short_tons) # vector of length 5 with the minimum, 25th percentile, medi

sd(coalProd_wv$Production_short_tons) # standard deviation

 coEffVar_prod <-   sd(coalProd_wv$Production_short_tons) /  mean(coalProd_wv$Production_short_tons)
  coEffVar_prod # coefficient of variation
```

GGPlot2

ggplot package

visualize your data with ggplot

Grammar of Graphics

ggplot allows you to visualize data by using three components: *a data set*, *a coordinate systems*, and *geoms*.

Steps of Plotting

1. Set the Data

```
coalProd_wv <- filter(coalProd, Mine_State == "West Virginia")

coalProd_wv
```

2. Set the Coordinates

aes(x=, y=) - sets the X axis and y axis variables

```
bar.coalProd <- ggplot(coalProd_wv, aes(x = Mine_County, y = Production_short_tons))

bar.coalProd
```

3. Set the Shape and Map the Data

```
bar.coalProd +
  geom_col()
```

Style and Customize your Plot

Add some color and give it a theme

ggplot themes ggplot color and fill

```
bar.coalProd +
  geom_col(fill = "skyblue") +
  theme_classic()
```

Adjust axis appearance and add some labels

ggplot labels ggplot scales

```
bar.coalProd +
  geom_col(fill = "skyblue") +
  theme_classic() +
  scale_y_continuous(labels = scales::label_number_si()) +
  theme(axis.text.x = element_text(angle = 90)) +
  labs (title = "Coal Production by County in West Virginia",
        subtitle = "2019 - 2021",
        caption = "EIA Annual Survey of Coal Production and Preparation",
        x = NULL,
        y = "In tons")
```

Break the plot out by facets or categorical variables

ggplot facets

```r
bar.coalProd +
  geom_col(aes(fill = Company_Type)) +
  theme_classic() +
  scale_y_continuous(labels = scales::label_number_si()) +
  theme(axis.text.x = element_text(angle = 90)) +
  labs (title = "Coal Production by County in West Virginia",
        subtitle = "by type of Mine: 2019 - 2021",
        caption = "EIA Annual Survey of Coal Production and Preparation",
        x = NULL,
        y = "In tons",
        fill = NULL) +
  facet_grid(rows = vars(Mine_Type))
```

Exporting Plots

You can export images of the plot either using the GUI from RStudio in the Plots tab. You can also code the export.

```r
# This will only export the most recently called plot

ggsave("coalproduction_wv_bycounty.png", height = 10, width = 10, dpi = 320) # saves the most recent pl
```

Additional Examples - Data Visualization

Scatter Plot

for comparing two numeric variables

```r
colnames(bigfoot_filter)

bigfoot.scatter <- ggplot(bigfoot_filter, aes(moon_phase, temperature_high))

bigfoot.scatter + geom_point()

bigfoot.scatter + geom_point() +
  geom_smooth(method = lm) +
  facet_grid(cols = vars(classification))
```

Histogram

for seeing the distribution of a variable

```r
bigfoot_filter

# Let's limit the data frame to that mooon phases over .50 during the summer

bigfoot_waning <- filter(bigfoot_filter, moon_phase > .50 & season == "Summer")
bigfoot_waning
```

```r
# Set the plot

bigfoot.hist <- ggplot(bigfoot_waning, aes(temperature_high))

# Set histogram

bigfoot.hist + geom_histogram()


# Add additional elements (labs, bins, color)

bigfoot.hist +
  geom_histogram(binwidth = 10 , color = "yellow", fill = "skyblue") +
  theme_classic() +
  labs(title = "Distubution of High Temperatures",
       caption = "high temperature of bigfoot sightings during a waning moon in summer",
       x = NULL,
       y = NULL)
```

Box Plot

show distributions of numeric data values compared between multiple groups.

```r
bigfoot.box <-  ggplot(bigfoot_filter, aes(season, temperature_high))

bigfoot.box + geom_boxplot()
```