CENTER FOR
TEACHING & LEARNING

Using R as a Grad Student

R is a robust programming language specifically designed for statistical computing and data visualization. It is extensively used for data analysis within academia because it and its packages are free, open-source, and widely available.

How to install R

The Comprehensive R Archive Network (CRAN) serves as the primary resource for all matters related to R. It is the central platform for downloading and installing R, discovering contributed packages to address specific issues, accessing answers to frequently asked questions, staying informed about the latest advancements, obtaining programming tips, and much more.

You first need a compatible computer to install R. **Begin by visiting CRAN at http://cran.r-project.org/. Select the appropriate operating system—Windows, Mac OS, or Linux—and follow the on-screen instructions**. You will need to download the 'base' package and execute the setup program, which will be named something like R*.exe for PCs or R*.dmg for Macs.

The second step is to install R Studio, a graphical user interface that provides a more user-friendly experience with the R language. **Steps to install the R Studio free version are here: https://posit.co/download/rstudio-desktop/ and the bottom of the page contains the executables/installers for Windows, Mac OS, and Linux.** After that, you will likely need to install several Packages to get the functionality you need.

Common packages in R

These packages form the backbone of R, covering a wide range of data manipulation, analysis, and visualization needs. You have to install them manually, however. **You can find them at CRAN to download, or in R Studio you can go to Tools, then click Install Packages, select the CRAN repository, and type the name of the package you need** (many of which are noted in the following list).

Once your **click Install, or type the install.packages() command**, R will have the package available for use. **At the start of each session, ensure that you load the packages using the library() command;** there are examples provided below in several of the list items.

Data Manipulation

dplyr: is a powerful package in R for data manipulation. It provides a set of functions that are particularly useful for data cleaning and transformation. Here is a list of some of the most commonly used functions in dplyr:

- **filter()**: Subset rows using column values.
- **select()**: Select columns by names.
- **mutate()**: Add new variables or transform existing ones.
- **summarize() (or summarise())**: Reduce multiple values to a single summary.
- **arrange()**: Reorder rows.
- **inner_join()**: Return all rows from x where there are matching values in y, and all columns from x and y.
- **left_join()**: Return all rows from x, and all columns from x and y. Rows in x with no match in y will have NA values in the new columns.

- **right_join()**: Return all rows from y, and all columns from x and y. Rows in y with no match in x will have NA values in the new columns.
- **full_join()**: Return all rows and columns from both x and y. Where there are not matching values, returns NA for the one missing.
- **semi_join()**: Return all rows from x where there are matching values in y, keeping just columns from x.
- **anti_join()**: Return all rows from x where there are not matching values in y, keeping just columns from x**.**
- **group_by()**: Group data by one or more variables.
- **ungroup()**: Remove grouping.
- **rename()**: Rename columns.
- **union()**: Return rows that appear in either or both x and y.
- **intersect()**: Return rows that appear in both x and y.
- **setdiff()**: Return rows that appear in x but not in y.

tidyr: Helps to tidy your data by reshaping it into a consistent format, making it easier to analyze. Some key functions in tidyr:

- **drop_na()**: This function removes rows containing missing values.
- **replace_na()**: This function replaces missing values with specified values.
- **fill()**: This function fills missing values in a column with the previous or next value.
- **complete()**: This function completes a dataset by filling in missing combinations of data.
- **unite()**: This function combines multiple columns into a single column.

Data Import

- **readr**: Fast and friendly functions for reading rectangular data, such as CSV files.
- **readxl**: Reads Excel files (both .xls and .xlsx formats).
- **haven**: Enables you to read and write data from SAS, SPSS, and Stata.

Data Visualization

- **ggplot2**: A system for declaratively creating graphics, based on The Grammar of Graphics.
- **plotly**: Creates interactive web-based graphics via the plotly.js JavaScript graphing library.
- **lattice**: Provides a high-level data visualization system with an emphasis on multivariate data.

Statistical Modeling

- **stats**: The default package for statistical functions that comes with R.
- **lme4**: Provides functions to fit linear and generalized linear mixed-effects models.
- **survival**: Contains functions for survival analysis, including Kaplan-Meier plots, Cox models, and more.

Reading data from a CSV file
Using read.csv("") to read data from a CSV file. Example:

- data <- read.csv("path/to/your/file.csv")

CENTER FOR
TEACHING & LEARNING

Reading data from Excel Files

Using readxl package to read Excel files.

- install.packages("readxl")
- library(readxl)
- data <- read_excel("path/to/your/file.xlsx")

Descriptive Statistics

Descriptive statistics summarize and describe the features of a dataset. In R, you can use various functions and packages to calculate descriptive statistics. Here's a basic guide on how to perform these analyses:

- Input your dataset

Install and load the psych package

- input your dataset
- install.packages("psych")
- library(psych)
- describe(faculty$time)

Describe is the function, faculty is the dataset, time is the variable. $ sign is the sign that connects the variable to the dataset. Therefore, faculty$time means the time variable in the faculty dataset.

Note: R is case-sensitive. Therefore, variable names data, Data, and DATA would be considered and treated as three different variables. Ensuring consistency in the naming of variables and functions is crucial to prevent errors arising from case sensitivity.

To Make a Histogram

- hist(faculty$time)
- hist is the function for histogram.
- When you run the code, you will see the histogram for the variable "time" in the faculty dataset on the right window for Plots.
- Download the histogram by clicking on "Export".

Calculating Mean, Median, Range, etc.

You can perform this calculation without the psych package:

- Input your dataset into R
- mean(faculty$time)
- median(faculty$time)
- min(faculty$time)
- max(faculty$time)
- range(faculty$time)
- sd(faculty$time)
- var(faculty$time)

Getting Summary of Descriptive Statistics for One Variable

- summary(faculty$time)
- summary(reading)

mean, median, min, max, range, sd, var are the functions, faculty is the dataset, time is the variable. The "$" is the sign that connects the variable to the dataset.

Correlation

To calculate the correlation in R, you can use the cor() function, which computes the correlation matrix for your dataset. Here's a detailed guide:

- Read the dataset

Compute the correlation matrix using Spearman method and pairwise complete observations.

- correlation_matrix <- cor(data, use = "pairwise.complete.obs", method = "spearman").
- data is the dataset.

For Pearson just change the method to Pearson.

Visualizing Correlation Matrix

You can visualize the correlation matrix using the corrplot package:

- Install and load corrplot package
- install.packages("corrplot")
- library(corrplot)
- Plot the correlation matrix
- corrplot(correlation_matrix, method = "circle").

This will produce a visual representation of the correlation matrix, making it easier to interpret the relationships between variables.

Performing T-test in R

Performing a t-test in R involves a few straightforward steps. Here's a step-by-step guide:

- Read your dataset
- install.packages("ggplot2") If you want to visualize the data.
- Load library(stats) for statistical functions
- library(ggplot2) for visualization (optional)

Independent Samples T-test

If group1 and group2 are independent (e.g., measurements from two different groups):

- t.test(group1, group2)

Paired Samples T-test

If group1 and group2 are paired (e.g., before and after measurements on the same subjects):

- t.test(group1, group2, paired = TRUE)

Linear Regression

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable and one or more independent variables. In R, you can perform linear regression using the lm() function. Here's a step-by-step guide:

- Install and load necessary packages (if not already installed).
- install.packages("ggplot2").
- library(ggplot2).
- Prepare your data: Ensure your data is in a data frame. For this example, let's use the built-in mtcars dataset. data(mtcars) head(mtcars).
- Fit a linear model: Use the lm() function to fit a linear model. For example, to predict mpg (miles per gallon) using wt (weight of the car):
- model <- lm(mpg ~ wt, data = mtcars)
- View the model summary: The summary() function provides detailed information about the model.
- summary(model)

Plotting Results

Use ggplot2 to visualize a regression line along with the data points.

- ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
   geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Linear Regression of MPG on Weight",
  x = "Weight",
  y = "Miles per Gallon").

ggplot: This initializes the ggplot object.

mtcars: This is the dataset being used. mtcars is a built-in dataset in R that contains various attributes of cars.

aes(x = wt, y = mpg): This sets up the aesthetic mappings. Here, wt (weight) is mapped to the x-axis and mpg (miles per gallon) is mapped to the y-axis.

geom_point(): This adds a layer of points (scatter plot) to the plot. Each point represents a car from the mtcars dataset.

geom_smooth(method = "lm", se = FALSE): This adds a smoothed line to the plot. method = "lm" specifies that a linear model (lm) is to be used for the smoothing, which means a linear regression line will be added. se = FALSE specifies that the standard error should not be displayed, so the confidence interval around the regression line will not be shown.

labs(title = "Linear Regression of MPG on Weight", x = "Weight", y = "Miles per Gallon"): labs: This adds labels to the plot. title = "Linear Regression of MPG on Weight" sets the title of the plot. x = "Weight" sets the label for the x-axis. y = "Miles per Gallon" sets the label for the y-axis.

Multiple regression

Multiple regression is a statistical technique that allows you to examine the relationship between a dependent variable and multiple independent variables. In R, you can perform multiple regression using the lm() function, which stands for linear model. Guide on how to perform a multiple regression in R:

- Prepare your data**:** Ensure your data is in a data frame format and that it is clean (no missing values, etc.).
- Load the necessary libraries (if not already loaded).
- For data manipulation and analysis: library(dplyr)
- For statistical modeling: library(stats)
- Fit the multiple regression model**:** Use the lm() function to fit the model. The formula interface is used to specify the dependent and independent variables. If you assume that your dependent variable is 'Y' and independent variables are 'X1', 'X2', and 'X3' then the model <- lm(Y ~ X1 + X2 + X3, data = data).
- Check the summary of the model**:** Use the summary() function to get detailed information about the model, including coefficients, R-squared, p-values, etc. summary(model).
- Diagnostics and Model Validation: After fitting the model, it's important to validate its assumptions: Check residuals**:** Ensure that residuals are normally distributed and homoscedastic (constant variance). Residual diagnostics plots:
- par(mfrow = c(2, 2))
- plot(model)
- Multicollinearity**:** Check for multicollinearity using the Variance Inflation Factor (VIF)
- library(car)
- vif(model)