Week 4 - Descriptive Statistics in Mathematical Modelling

Introduction

In this lab, what we will be covering is the filtering of datasets and how to generate some basic statistics. We will also be covering how to load datasets from different places. Finally, we will be looking at how to generate a markdown document which combines all of these steps.

Loading

Data can come in a variety of forms, but for the purposes of this module we will be looking at three different ways. We will be looking at in built datasets, datasets from different libraries and datasets from csv files.

Loading datasets from inbuilt libraries:

R comes with several built-in datasets that you can use for practice or testing. To load datasets from inbuilt libraries, you can use functions like data() or datasets::data(). Here's how to do it:

```r
# Load a dataset using the data() function
data(mtcars)

# Load a dataset using the datasets package
library(datasets)
data(iris)
```

Loading datasets from other libraries:

Many R packages come with datasets that are specific to the package's domain. To load datasets from other libraries, you first need to install and load the package, then you can access the datasets using functions provided by the package. Here's an example:

```
# Install and load the package
install.packages("ggplot2")
library(ggplot2)

# Load a dataset from the ggplot2 package
data(diamonds)
```

Loading datasets from CSV files:

You can also load datasets from CSV (comma-separated values) files stored on your local machine or from a URL. To do this, you can use the read.csv() function or its variants like read.csv2() for reading CSV files with different delimiters. Here's how to do it:

```
# Load a CSV file from local directory
my_data <- read.csv("path/to/your/file.csv")

# Load a CSV file from a URL
my_data <- read.csv("https://example.com/data.csv")
```

If your CSV file has a different delimiter than a comma, you can specify it using the sep argument in the read.csv() function. For example, if your CSV file uses semicolons as delimiters:

```
my_data <- read.csv("path/to/your/file.csv", sep = ";")
```

These are the basic steps to load datasets from various sources in R. Depending on the format and location of your data, you may need to adjust the loading process accordingly.

Loading datasets from Excel

To load datasets from Excel into R, you can use the readxl package, which is designed specifically for this purpose. Here's a step-by-step guide:

Install and Load the readxl package: If you haven't already installed the readxl package, you can do so by running install.packages("readxl"). Once installed, load the package using library(readxl).

```
install.packages("readxl")
library(readxl)
```

Locate the Excel File: Ensure that your Excel file is located in your working directory, or provide the full path to the file.

Load the Excel File: Use the read_excel() function to load the Excel file into R. You'll need to specify the path to the file and the sheet name if it's not the first sheet.

```
# Load the entire Excel file
data <- read_excel("path_to_your_excel_file.xlsx")

# If the data is on a specific sheet, specify the sheet name
data <- read_excel("path_to_your_excel_file.xlsx", sheet = "Sheet1")
```

Filtering data

Using Base R:

You can filter data using base R functions like subset() and indexing. Here's how:

```r
# Create a sample data frame
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David", "Emma"),
  Age = c(25, 30, 35, 40, 45),
  Gender = c("Female", "Male", "Male", "Male", "Female")
)

# Using subset() function
subset_data <- subset(data, Age > 30)
print(subset_data)
```

```
     Name Age Gender
3 Charlie  35   Male
4   David  40   Male
5    Emma  45 Female
```

```r
# Using indexing
indexed_data <- data[data$Age > 30, ]
print(indexed_data)
```

```
     Name Age Gender
3 Charlie  35   Male
4   David  40   Male
5    Emma  45 Female
```

Using dplyr package:

The dplyr package provides a concise syntax for data manipulation. You can use the filter() function from dplyr to filter data based on conditions. Here's how:

```
# Install and load the dplyr package
install.packages("dplyr")
library(dplyr)

# Filter data using dplyr
filtered_data <- filter(data, Age > 30)
print(filtered_data)
```

Using subset() with logical conditions:

You can also use logical conditions directly within the subset() function to filter data. Here's an example:

```
# Filter data using subset() with logical conditions
subset_data <- subset(data, Age > 30 & Gender == "Male")
print(subset_data)
```

```
     Name Age Gender
3 Charlie  35   Male
4   David  40   Male
```

Using which() function:

The which() function in R returns the indices of the elements in a logical vector that are TRUE. You can use it to filter data based on conditions. Here's an example:

```
# Filter data using which() function
index <- which(data$Age > 30)
filtered_data <- data[index, ]
print(filtered_data)
```

```
     Name Age Gender
3 Charlie  35   Male
4   David  40   Male
5    Emma  45 Female
```

These are some of the common methods to filter data in R. Depending on your preference and the complexity of your filtering conditions, you can choose the method that best suits your needs.

Basic Statistics

When it comes to calculating descriptive statistics, R can basically do it all. Let's start with functions that are included in the base installation. We will then look for extensions that are available through the use of user-contributed packages. For illustrative purposes, we will again use several of the variables from the Motor Trend Car Road Tests (mtcars) dataset provided in the base installation. We will focus on miles per gallon (mpg), horsepower (hp), and weight (wt):

```
myvars <- c("mpg", "hp", "wt")
head(mtcars[myvars])
```

```
                   mpg  hp    wt
Mazda RX4         21.0 110 2.620
Mazda RX4 Wag     21.0 110 2.875
Datsun 710        22.8  93 2.320
Hornet 4 Drive    21.4 110 3.215
Hornet Sportabout 18.7 175 3.440
Valiant           18.1 105 3.460
```

Descriptive statistics in R provide summaries of the main characteristics of a dataset. These summaries include measures of central tendency, dispersion, and shape of the distribution. Here's a step-by-step guide on how to use descriptive statistics in R:

Before you can perform descriptive statistics, you need to load your data into R. As we saw above, you can load data from various sources such as CSV files, Excel files, databases, or built-in datasets. For this example we will be using the iris dataset. We can load this by completing the following command.

```
# Load a dataset using the datasets package
library(datasets)
df = data(iris)
```

This loads the data, and assigns this to the variable df.

Summary statistics:

Use the summary() function to get a summary of your data, including minimum, 1st quartile, median, mean, 3rd quartile, and maximum for numeric variables. For factors, it shows the count of each level.

```
# Get summary statistics
summary(df)
```

```
   Length     Class      Mode 
        1 character character 
```

Measures of central tendency:

Use functions like mean(), median(), and mode() (from the DescTools package) to calculate measures of central tendency such as mean, median, and mode.

```
# Calculate mean
mean(my_data$column_name)

# Calculate median
median(my_data$column_name)

# Calculate mode (requires the DescTools package)
library(DescTools)
Mode(my_data$column_name)
```

Measures of dispersion:

Use functions like sd() for standard deviation, var() for variance, range() for range, IQR() for interquartile range, and quantile() for quantiles.

```
# Calculate standard deviation
sd(my_data$column_name)

# Calculate variance
var(my_data$column_name)

# Calculate range
range(my_data$column_name)
```

```
# Calculate interquartile range
IQR(my_data$column_name)

# Calculate quantiles
quantile(my_data$column_name)
```

Frequency distributions:

Use the table() function to create frequency distributions for categorical variables.

```
# Frequency distribution for a factor variable
table(my_data$factor_column)
```

Correlations

Correlation coefficients are used to describe relationships among quantitative variables. The sign ± indicates the direction of the relationship (positive or inverse), and the magnitude indicates the strength of the relationship (ranging from 0 for no linear relationship to 1 for a perfectly predictable linear relationship).

In this section, we look at a variety of correlation coefficients, as well as tests of significance.

We will use the state.x77 dataset available in the base R installation. It provides data on the population, income, illiteracy rate, life expectancy, murder rate, and high school graduation rate for the 50 US states in 1977. There are also temper- ature and land-area measures, but we drop them to save space. In addition to the base installation, we'll be using the psych and ggm packages. R can produce a variety of correlation coefficients, including Pearson, Spearman, Kendall, partial, polychoric, and polyserial. The Pearson product-moment correlation assesses the degree of linear relationship between two quantitative variables. Spearman's rank-order correlation coefficient assesses the degree of relationship between two rank-ordered variables. Kendall's tau is also a nonparametric measure of rank correlation.

The cor() function produces all three correlation coefficients, whereas the cov() function provides covariances. There are many options, but a simplified format for producing correlations is

```
cor(x, use= , method= )
```

Where x is a matrix or a data frame, and use specifies the handling of missing data. The options are all.obs (assumes no missing data), everything (any correlation involving a case with missing values will be set to missing), complete.obs (listwise deletion), and pairwise. complete.obs (pairwise deletion). The method spec- ifies the type of correlation. The options are pearson, spearman, and kendall. The default options are use =“everything” and method= “pearson”. An example is provided in Table 4 The first call produces the variances and covariances. The second provides Pearson product-moment correlation coefficients, and the third produces Spearman rank-order correlation coefficients. You can see, for example, that a strong positive correlation exists between income and high school graduation rate and that a strong negative correlation exists between illiteracy rates and life expectancy. A partial correlation is a correlation between two quantitative variables, controlling for one or more other quantitative variables. You can use the pcor() function in the ggm package to provide partial correlation coefficients. Again, this package is not installed by default, so be sure to install it on first use. The format is

```
pcor(u, S)
```

where u is a vector of numbers, with the first two numbers being the indices of the variables to be correlated, and the remaining numbers being the indices of the conditioning variables (that is, the variables being partialed out), and S is the covariance matrix among the variables. An example will help clarify this:

```
library(ggm)
colnames(states)
pcor(c(1,5,2,3,6), cov(states))
```

In this case, 0.346 is the correlation between population (variable 1) and murder rate (variable 5), controlling for the influence of income, illiteracy rate, and high school graduation rate (variables 2, 3, and 6 respectively). The use of partial correlations is common in the social sciences.

Visualizing data:

Visualize your data using histograms, box plots, or other plots to get a better understanding of the distribution and spread of your data.

```
# Histogram
hist(my_data$numeric_column)

# Box plot
boxplot(my_data$numeric_column)
```

These are some basic steps to perform descriptive statistics in R. Depending on your specific requirements and the complexity of your data, you may need to explore additional functions and techniques.

Tasks

Exercise 1

Below are some functions that you can use to summarise some data.

```
mean()
sd()
var()
min()
max()
median()
length()
range()
quantile()
fivenum()
```

1. Load the iris dataset (see instructions above) and experiment with these functions.
2. You will see that there are 3 types of flowers Setosa, Versicolor, Virginica. Are there any differences in the descriptive statistics for each of these flowers?
3. Create a markdown document which includes writing and code, to show these differences.

Exercise 2

To load the nycflights13 dataset in R, you need to first make sure that the nycflights13 package is installed. This package contains a dataset about flights departing from New York City airports in 2013. Follow these steps:

Install and Load the nycflights13 Package: If you haven't already installed the package, you can do so by running install.packages("nycflights13"). Once installed, load the package using library(nycflights13).

```
install.packages("nycflights13")
library(nycflights13)
```

Load the Dataset: Once the package is loaded, you can load the dataset using one of the available datasets included in the package. Here are some common datasets available in the nycflights13 package:

flights: Information about all flights departing from New York City airports in 2013. weather: Hourly meteorological data for each of the three airports in 2013. airports: Airport metadata. planes: Aircraft metadata. airlines: Airline metadata. For example, to load the flights dataset, you can use:

```
data("flights")
```

This will load the flights dataset into your R environment.

1. Identify what type of data is in flights, weather, airports, planes and airlines.
2. Answer the following questions

a. What was the most popular destination on new years day?
b. What was the most popular destination overall?
c. There are numbers listed under arr_delay and dep_delay, which denote the number of minutes delay for the arrival and departure respectively. What does a - in front of the number mean?
d. Were flights on new years day delayed?
e. Which airlines were the most delayed?

3. Create a markdown document which includes writing and code, to answer these questions.

For each question, it would help to visualise what data you need before you start to try to code this.