STAT 231 Fall 2023 Introductory R Tutorial

This document provides a brief introduction to using R for the assignments in STAT 231 this term. The information here should be enough to get you started, and assumes that you have downloaded R and RStudio and used it before. Please review the Introduction to R and RStudio on LEARN for a more comprehensive overview of using R.

Downloading Your Sample

First, follow the instructions in the file titled Assignment Dataset Information. You should have a sample of data stored as a CSV file, with a name of the form stat231f23datasetSTUDENTID.csv (with STUDENDID replaced by your 8-digit student ID number).

Importing the Data

You should follow the instructions to import the data as given in the Introduction to R and RStudio. I use the `read.csv()` command as follows:

```r
mydata <- read.csv("stat231f23dataset12345678.csv")
```

Syntax Note: In R we use `<-` to create objects. Here I am creating the object `mydata` to store my datset.

Syntax Note: This command assumes that my data file is stored in my current R working directory. We could also specify the path to the file's location when using this function.

Inspecting the Data

It's always a good idea to check a few properties of your dataset, for example:

```r
dim(mydata)
```

```
## [1] 894  13
```

This tells me the dimensions of my dataset: 894 rows and 13 columns, corresponding to 894 traffic stops and 13 variates.

I can also check the names of the variates in my dataset with:

```r
colnames(mydata)
```

```
##  [1] "city"         "date"         "day.of.week"  "time"         "time.hour"
##  [6] "lat"          "lng"          "subject.age"  "subject.race" "subject.sex"
## [11] "violation"    "vehicle.make" "outcome"
```

The object `mydata` is a data frame, which can be thought of as a table of values. This therefore has rows and columns and we can access specific entries in the table using square brackets. For example:

```r
mydata[1, 3]
```

```
## [1] "Tuesday"
```

This tells me the entry in the first row and the third column (the day of the week on which the first traffic stop in my sample took place), and can be modified for any row or column.

We can also specify ranges of values:

```
mydata[1:5, 11:12]
```

```
##                                    violation vehicle.make
## 1                 DISPLAY ST REG-FRONT/REAR        other
## 2   IVC - NO TURN SIGNAL LIGHTS/ 1ST & 2ND       toyota
## 3 OBSTRUCTION DRIVER'S VIEW/TINTED WINDOWS    chevrolet
## 4     HEADLIGHT TWO REQUIRED-MOTOR VEHICLE        other
## 5                           STOP AT STOP SIGN        other
```

This gives me the entries for columns 11 and 12 (the violation or reason for the stop, and the make of vehicle stopped) for the first five stops in my sample.

It may be more intuitive to refer to variates by their names, rather than the number of their column, and we can do this using $. For example:

```
mydata$violation[1:5]
```

```
## [1] "DISPLAY ST REG-FRONT/REAR"
## [2] "IVC - NO TURN SIGNAL LIGHTS/ 1ST & 2ND"
## [3] "OBSTRUCTION DRIVER'S VIEW/TINTED WINDOWS"
## [4] "HEADLIGHT TWO REQUIRED-MOTOR VEHICLE"
## [5] "STOP AT STOP SIGN"
```

```
mydata$vehicle.make[1:5]
```

```
## [1] "other"     "toyota"    "chevrolet" "other"     "other"
```

Checking for Missing Values

Very important: upon importing your dataset you should check for missing values (usually indicated by 'NA'). The only missing values should occur in the variates `vehicle.make`, and `outcome`, as these are not available for San Francisco, and Chicago, respectively. One useful function is `is.na()`:

```
mydata$vehicle.make[5]
```

```
## [1] "other"
```

```
is.na(mydata$vehicle.make[5])
```

```
## [1] FALSE
```

```
mydata$vehicle.make[500]
```

```
## [1] NA
```

```
is.na(mydata$vehicle.make[500])
```

```
## [1] TRUE
```

Syntax Note: `is.na()` checks if a value is missing, returning 'TRUE' if it is missing, and 'FALSE' if it isn't.

Here we can see that the `vehicle.make` variate is present for the 5^{th} stop in my dataset, but missing for the 500^{th} stop in my dataset.

We can quickly check whether there are any missing values in the sample by combining `is.na()` with `sum()`:

```
sum(is.na(mydata[, 1:11]))
```

```
## [1] 0
```

Syntax Note: `sum()` returns the sum of a list of numbers, treating the value 'FALSE' as 0, and 'TRUE' as 1.

Here we have summed over all elements in my sample excluding columns 12 and 13 (where missing values are expected as these correspond to `vehicle.make` and `outcome`). The fact this command returned the value 0 confirms my sample has no missing data in the relevant variates and has therefore been imported correctly. If there were missing data, this command would return a non-zero value.

If You Have Missing Values

The primary dataset contains no missing values in the first 11 variates. If the above command indicates you have missing values then there is a problem with your data import which must be remedied before you complete any assignment work.

Our first suggestion is to use the `read.csv()` function as used in this tutorial. If this doesn't work, then please follow the following steps:

1. Load the `data.table` package by running the command

```
library(data.table)
```

2. Then import your dataset using the read.table() function:

```
mydata <- data.table::fread("stat231f23datasetSTUDENTID.csv")
```

replacing STUDENTID with your ID number. For this function to work you must either have your file saved in your current R working directory, or instead of specifying the file name you can specify the path to the file name, for example:

```
dataset <- data.table::fread("/path/to/file/stat231f23datasetSTUDENTID.csv")
```

Once you have carried out these steps, check that you have no missing values as per the above commands. If you still have problems, please ask for help on Piazza!

Probability Calculations

We'll end this tutorial with some more general commands, not specific to the assignment dataset. You should familiarize yourself with these functions, especially in preparation for exams.

One of the most common ways we will use R this term is for calculations related to probability distributions. You may have used probability tables to find such probabilities (and examples of these are included in the STAT 231 Course Notes). The use of probability tables is not required for STAT 231.

We will begin with an illustration involving the Gaussian (Normal) distribution.

Probability Calculations: Gaussian

First, let's generate a random observation from the $G(0, 1)$ distribution:

```r
set.seed(1)
rnorm(1, 0, 1)
```

```
## [1] -0.6264538
```

Syntax Note: The `set.seed()` function specifies a random seed which will be used for random number generation. This ensures that we will get the same results each time we run the commands.

The `rnorm()` command generates random numbers from a Gaussian distribution. The first number indicates how many observations we want, the second number indicates the mean of the distribution, and the third number indicates the standard deviation of the distribution.

PROTIP: Don't forget that the final number specifies the standard deviation, and *not* the variance!

Here, we have therefore generated 1 random observation from a $G(0, 1)$ distribution.

Instead, we could generate 5 observations from a $G(1, 2)$ distribution as follows:

```r
set.seed(1)
rnorm(5, 1, 2)
```

```
## [1] -0.2529076  1.3672866 -0.6712572  4.1905616  1.6590155
```

The default values of the mean and standard deviation for this command are 0 and 1. Therefore, if we do not specify these values R will assume we want a $G(0, 1)$ distribution. For example, to generate 5 observations from a $G(0, 1)$ distribution we can simply use:

```r
set.seed(1)
rnorm(5)
```

```
## [1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

Note that the first of these values matches the observation we first generated because we have set the same seed.

If we want to conduct probability calculations using the Gaussian distribution we can use `pnorm()`:

```r
pnorm(5, 1, 2)
```

```
## [1] 0.9772499
```

This gives us $P(Y \leq 5)$ where $Y \sim G(1, 2)$. We can change the first number to generate different probabilities.

We can combine multiple `pnorm()` calls to compute more complex probabilities, for example:

```r
pnorm(3, 0, 5) - pnorm(2, 0, 5)
```

```
## [1] 0.07032514
```

This gives $P(2 \leq Y \leq 3)$ where $Y \sim G(0, 5)$.

Instead of probabilities, we can generate quantiles by using `qnorm()`:

```r
qnorm(0.975, 1, 2)
```

```
## [1] 4.919928
```

This returns the value a such that $P(Y \leq a) = 0.975$ where $Y \sim G(1, 2)$.

Finally, we can use the `dnorm()` function to evalute the value of the probability density function of the Gaussian distribution for specific values:

```r
dnorm(5, 1, 2)
```

```
## [1] 0.02699548
```

PROTIP: You should be familiar with the concept of a probability density function from STAT 230. If you are unsure what this R output is telling you then this indicates you need to review your STAT 230 notes!

Probability Calculations: Other Distributions

Notice that for the Gaussian distribution we had four functions:

1. `rnorm()` generated $\underline{\text{r}}$andom observations.
2. `pnorm()` calculated $\underline{\text{p}}$robabilities.
3. `qnorm()` calculated $\underline{\text{q}}$uantiles.
4. `dnorm()` calculated the value of the probability $\underline{\text{d}}$ensity function.

R has built in functions for many other probability distributions, and each one follows the same naming convention.

Probability Calculations: Exponential

For example, for the Exponential distribution we would use `rexp()`, `pexp()`, `qexp()`, and `dexp()`.

PROTIP: Be careful when using the Exponential functions in R. We use the notation Exponential(θ) in the course, meaning an Exponential distribution with mean θ. In R, we specify the *rate* parameter $\frac{1}{\theta}$. Therefore:

```r
set.seed(1)
rexp(5, 1/2)
```

```
## [1] 1.5103637 2.3632856 0.2914135 0.2795905 0.8721373
```

generates 5 observations from an Exponential(2) distribution. One way to verify this is to generate a large sample and check the sample mean:

```r
set.seed(1)  # set the seed
expsample <- rexp(10000, 1/2)  # generate a sample of size 10,000 from Exponential(2)
mean(expsample)  # calculate the sample mean
```

```
## [1] 1.996722
```

Probability Calculations: Poisson

R also has analogous commands for discrete random variables. However, take care when computing probabilities:

```r
ppois(2.5, 1)
```

```
## [1] 0.9196986
```

```r
ppois(3, 1)
```

```
## [1] 0.9810118
```

```r
ppois(3.5, 1)
```

```
## [1] 0.9810118
```

These commands generate, respectively, $P(Y \leq 2.5), P(Y \leq 3)$, and $P(Y \leq 3.5)$ where $Y \sim$ Poisson(1). Note that $P(Y \leq 3) = P(Y \leq 3.5)$ because Y is discrete. This is another concept you should be very comfortable with from STAT 230/240!

Note also that when working with a discrete probability distribution the density functions analogous to `dnorm()` give the value of the probability function (not the value of the probability density function). For example:

```r
dpois(3, 1)
```

```
## [1] 0.06131324
```

gives $P(Y = 3)$ where $Y \sim$ Poisson(1).

Probability Calculations: Binomial

Finally, let's look at the Binomial distribution. R's functions for this distribution can sometimes be a little tricky. For example:

```r
set.seed(1)
rbinom(3, 100, 0.5)
```

```
## [1] 52 46 60
```

generates 3 observations from a $Y \sim$ Binomial$(100, 0.5)$ distribution. 1 observation from such a distribution can be thought of as conducting an experiment where you flip a fair coin 100 times and counting the number of heads. The output of the preceding command is therefore similar to performing this experiment 3 times.

For Binomial probabilities we use `pbinom()`:

```r
pbinom(5, 10, 0.5)
```

```
## [1] 0.6230469
```

This gives $P(Y \leq 5)$ where $Y \sim \text{Binomial}(10, 0.5)$. This is (for example) the probability of 5 or fewer heads if we flip a fair coin 10 times.

As the Binomial distribution is discrete, `dbinom()` returns the value of the probability function, for example:

```r
dbinom(4, 10, 0.6)
```

```
## [1] 0.1114767
```

returns $P(Y = 4)$ where $Y \sim \text{Binomial}(10, 0.6)$. You can verify this calculation from the Binomial probability function directly by:

```r
choose(10, 4) * 0.6^4 * 0.4^6
```

```
## [1] 0.1114767
```

Syntax Note: `choose(n, r)` returns the value of n choose r.

PROTIP: Experiment using R for generating observations and probability calculations for different distributions. For discrete distributions, compute the probabilities directly to the probability functions you have studied in STAT 230, to ensure you are familiar with how these commands operate.