PS303 - Week 2

Describing data (pp.126-139; 151-169)

From vectors to data frames

Last week we spoke about how to create *vectors* to contain series of values. We also discussed some basic operations on vectors. When we have multiple data series of interest, it can be useful to create multiple vectors and combine them into *data frames*, which are two-dimensional data structures. Data frames ensure vectors are organized, in a single structure, and can be appropriately labeled for easy reference.

One of our goals today will be to generate a data frame and estimate some summary statistics using *R*'s built-in functions. The data for today's exercise has been taken from StatsFiji. We will also provide a gentle introduction to the underlying (statistical) calculations, along with the symbols they use, to have a clearer idea of how estimates are computed.

USP enrollment rates

In the image below, we have **3 columns** and **4 rows** of data. Each column represents total students for each year. Each row represents total students for each faculty.

Our first task today will be to re-create this table in *R*.

Equivalent Full-Time Students (EFTS) by Faculty	2015 EFTS	2016 EFTS	2017 EFTS
Faculty			
Faculty of Arts, Law &Education	3,448.5	3,461.9	3,765.4
Faculty of Business-Economics	5,004.5	5,106.6	5,374.4
Pacific TAFE	4,689.6	4,860.2	5,352.0
Faculty of Science, Technology & Environment	3,479.0	3,484.7	3,915.0

Source: University of the South Pacific

Lets first create the three **column** vectors representing the enrolment frequency of students each year.

```
# Creating vectors for each of the years
ef.2015 <- c(3448.5,5004.5,4689.6,3479)          # For 2015
ef.2016 <- c(3461.9,5106.6,4860.2,3484.7)        # For 2016
ef.2017 <- c(3765.4,5374.4,5352,3915)            # For 2017
schools <- factor(c("FALE","FBE","TAFE","FSTE")) # School names (as factor)

# We have to inform R that 'schools' is a factorial (categorical) variable, otherwise any non-numerical
```

```r
# We can generate columns using the function 'cbind.data.frame' and store the output in a variable call
my.df <- cbind.data.frame(schools,ef.2015,ef.2016,ef.2017)

my.df
```

```
##   schools ef.2015 ef.2016 ef.2017
## 1    FALE  3448.5  3461.9  3765.4
## 2     FBE  5004.5  5106.6  5374.4
## 3    TAFE  4689.6  4860.2  5352.0
## 4    FSTE  3479.0  3484.7  3915.0
```

We have our first data frame! Whenever creating data frames, ensure that:

- each row represents a participant/identity variable
- each column should represent a task/condition variable

Suppose we want to re-name columns...

```r
# Let's look at the column names present...
colnames(my.df)
```

```
## [1] "schools" "ef.2015" "ef.2016" "ef.2017"
```

We can assign names to each of the column labels

```r
# When changing labels, the ORIGINAL label should be on the left-hand side and the ALTERNATIVE label on

colnames(my.df)=c("schools"="Schools",
                  "ef.2015"="2015",
                  "ef.2016"="2016",
                  "ef.2017"="2017")

my.df
```

```
##   Schools   2015   2016   2017
## 1    FALE 3448.5 3461.9 3765.4
## 2     FBE 5004.5 5106.6 5374.4
## 3    TAFE 4689.6 4860.2 5352.0
## 4    FSTE 3479.0 3484.7 3915.0
```

We could also alter `row.names` if necesary...

Let's try answering the following questions using *R*:

1. What was the total number of students enrolled in *FALE*?
2. What was the total number of students enrolled for the 2016 academic year?
3. What were the mean and standard deviation of students enrolled in 2017?

Q1: What was the total number of students enrolled in *FALE*?

The data for each faculty is provided in individual rows (specifically, `FALE` is the first level under the `Schools` variable inside the data frame `my.df`). Our goal is to:

- Select all numeric values associated with the `FALE` factor in `my.df`
- Sum all the values together.

First, let's look at how we select variables (columns) in R using the `$` operator.

```
# If you type in `my.df$`, you will see the variables present inside the df. For our exercise, we want

# Selecting the Schools variable
my.df$Schools
```

```
## [1] FALE FBE  TAFE FSTE
## Levels: FALE FBE FSTE TAFE
```

We only want to select the `FALE` level

```
my.df$Schools=="FALE" # This functions as an INDEX
```

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

Now we can apply the index to the data frame

```
my.df[my.df$Schools=="FALE",] # Values before the comma describe ROW indices
```

```
##   Schools   2015   2016   2017
## 1    FALE 3448.5 3461.9 3765.4
```

We only want the 2^{nd}, 3^{rd} and 4^{th} values of the row

```
my.df[my.df$Schools=="FALE",2:4] # Values after the comma describe COLUMN indices
```

```
##     2015   2016   2017
## 1 3448.5 3461.9 3765.4
```

Almost there!

Now we can add up these values by using the `sum` function on the entire argument

```
sum(my.df[my.df$Schools=="FALE",2:4])
```

```
## [1] 10675.8
```

We could also store part of the argument inside a variable on which we could run operations

```
fale.st <- (my.df[my.df$Schools=="FALE",2:4]) # Assign the operation to a variable, then
sum(fale.st)   # Apply a function to the newly created variable
```

```
## [1] 10675.8
```

A1: The total number of students enrolled in FALE across the years 2015, 2016 and 2017 was **10675.8**.

Q2: Total students enrolled for the year 2016?

Recall that the years were stored as COLUMN variables. We can look at the STRUCTURE of our dataset by using the `str` argument

```
str(my.df)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ Schools: Factor w/ 4 levels "FALE","FBE","FSTE",..: 1 2 4 3
##  $ 2015   : num  3448 5004 4690 3479
##  $ 2016   : num  3462 5107 4860 3485
##  $ 2017   : num  3765 5374 5352 3915
```

It's comparatively easier to run operations on column variables as there's no need to specify an index...

```
my.df[,"2016"] # Remember that values AFTER a comma specify the column
```

```
## [1] 3461.9 5106.6 4860.2 3484.7
```

Now the sum...

```
sum(my.df[,"2016"])
```

```
## [1] 16913.4
```

When working with columns, it can be easier to use the `$` operator

```
# To refer to a single column with a data frame, use the '$' operator
sum(my.df$`2016`)
```

```
## [1] 16913.4
```

Back-ticks are required to call in a variable that begins with a **number** when using the `\$` operator (e.g., '2015' instead of 2015). It is good practice to use variable names that DO NOT begin with numbers and/or DO NOT contain spaces/special characters

A2: The total number of students enrolled during 2016 was **16913.4**.

As you may have surmised by now, we can refer to specific rows/columns frame by indexing within **square brackets**. This allows us to refer to rows/columns by their position in the data frame.

Suppose you are interested in knowing the number of students enrolled in 2017 for FALE only.

```
my.df[my.df$Schools=="FALE","2017"] # We have to specify the ROW index, not the column
```

```
## [1] 3765.4
```

What about all the students in 2017?

```
my.df[,"2017"]
```

```
## [1] 3765.4 5374.4 5352.0 3915.0
```

Indexing is a powerful tool for extracting values!

Q3: What were the *mean* and *standard deviation* of students enrolled in 2017?

The mean is the *average* point estimate of a numerical data series. Standard deviation describes the dispersion of the mean point estimate.

R has base functions for estimating both quantities (called `mean()` and `sd()` respectively)

```
# Let's estimate the mean of 2017 enrolment rates
mean(my.df$`2017`)
```

```
## [1] 4601.7
```

```
# And the standard deviation
sd(my.df$`2017`)
```

```
## [1] 881.4704
```

We can also estimate the `median` (or middle value)

```
median(my.df$`2017`)
```

```
## [1] 4633.5
```

For most quantitative analyses that we will cover, the point estimates of interest are typically the `mean` (for parametric tests) and `median` (for nonparametric tests).

Beyond point estimates, we also require *range* estimates to determine the level of variance associated with our point estimate. A common variance estimate is *standard deviation* (σ), which describes how 'far' a given value from a series is likely to be from the mean estimate

Let's use *R*'s built-in function for estimating standard deviation for the year 2016 and store it in a variable called `sd.2016`

```r
sd.2016 <- sd(my.df$`2016`)
sd.2016
```

```
## [1] 877.6898
```

We can round this to one decimal place using the `round` function...

```r
round(sd.2016,digits=1) # We can instruct R how many digits we want out estimated to be rounded up by
```

```
## [1] 877.7
```

We can also summarize our column data with the `summary` function

```r
summary(my.df)
```

```
##  Schools       2015           2016           2017
##  FALE:1   Min.   :3448   Min.   :3462   Min.   :3765
##  FBE :1   1st Qu.:3471   1st Qu.:3479   1st Qu.:3878
##  FSTE:1   Median :4084   Median :4172   Median :4634
##  TAFE:1   Mean   :4155   Mean   :4228   Mean   :4602
##           3rd Qu.:4768   3rd Qu.:4922   3rd Qu.:5358
##           Max.   :5004   Max.   :5107   Max.   :5374
```

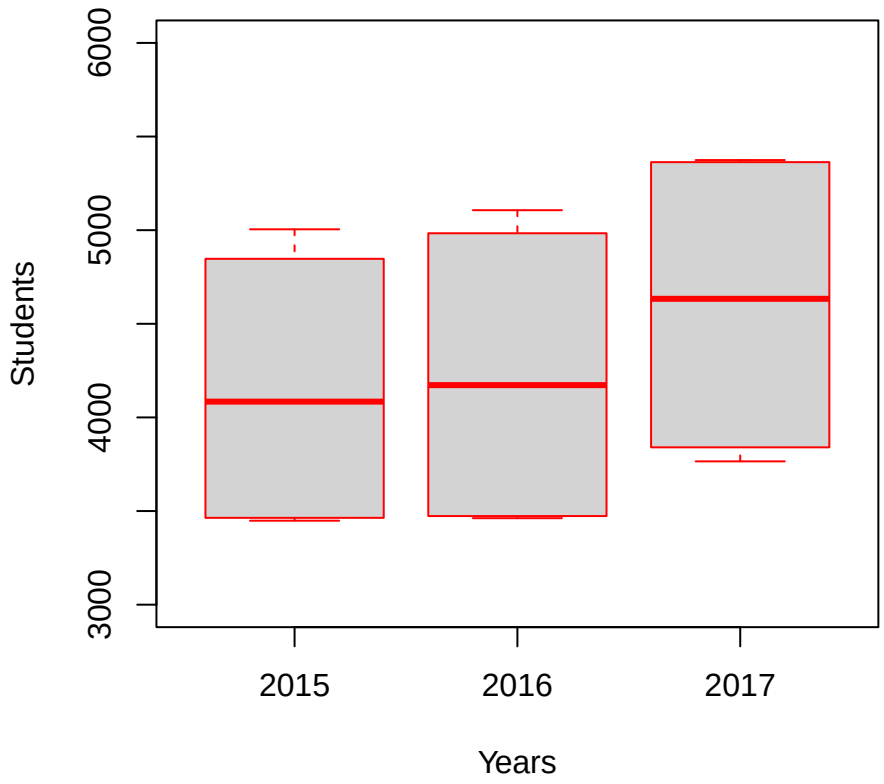
Note the minimum and maximum values within the range, the mean and median estimates, along with the limits of the inter-quartile range (IQR). We will return to what these estimates mean near the end of the class.

For now, we can visually illustrate these summaries using a **boxplot**.

Boxplot summary

Let's visualize the data with a boxplot

```r
boxplot(x = my.df[,-1],         # Name of your data frame (we exclude the first column, 1)
        xlab = "Years",         # x-axis label
        ylab = "Students",      # y-axis label
        ylim = c(3000,6000),    # Limit of y-axes (from 3000 to 6000 students)
        border = "red",         # boxplot border colour
        frame.plot = TRUE       # Draw a frame around the plot
)
```



Try to visually render your data whenever possible. It will help communication of your results to both specialists and non-specialists

Student enrolment for 2017 is higher than 2015 and 2016

Some more boxplot options

```
>  boxplot( formula =  margin ~ year,    # the formula
+           data = afl2,                 # the data set
+           xlab = "AFL season",         # x axis label
+           ylab = "Winning Margin",     # y axis label
+           frame.plot = FALSE,          # don't draw a frame
+           staplewex = 0,               # don't draw staples
+           staplecol = "white",         # (fixes a tiny display issue)
+           boxwex = .75,                # narrow the boxes slightly
+           boxfill = "grey80",          # lightly shade the boxes
+           whisklty = 1,                # solid line for whiskers
+           whiskcol = "grey70",         # dim the whiskers
+           boxcol = "grey70",           # dim the box borders
+           outcol = "grey70",           # dim the outliers
+           outpch = 20,                 # outliers as solid dots
+           outcex = .5,                 # shrink the outliers
+           medlty = "blank",            # no line for the medians
+           medpch = 20,                 # instead, draw solid dots
+           medlwd = 1.5                 # make them larger
+ )
```

Manually estimating *mean* and *standard deviation*

Each value within a data series represents an **iteration** (i). Assuming $i = 1$, we can write the enrolment numbers for 2017 as

$$3765.4 + 5374.4 + 5352 + 3915 = x_i + x_{i+1} + x_{i+2} = \sum x_i$$

We can divide the sum of values ($\sum x_i$) by the total number of iterations (n) to give us the mean ($\hat{x}$). We may now compute the average/mean estimate for the year 2017:

$$\frac{3765.4 + 5374.4 + 5352 + 3915}{4} = 4601.7$$

> The sum ($\sum$) of each iteration (x_i) divided by the total number of iterations (n) gives us the mean estimate ($\hat{x}$). Symbolically, $\frac{\sum x_i}{n} = \hat{x}$.

Next, let's go over the formula for *sample* standard deviation (σ) that is used by R (this varies slightly from the formula for the *population* standard deviation). Recall that σ measures how values are 'spread' around the mean estimate - it is a measure of data variance (larger σ's mean wider spread of data).

The *population* standard deviation is written as:

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x})^2}{N}}$$

The *sample* standard deviation (σ), which is the one we will use, is written as:

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x})^2}{N-1}}$$

We typically use the population standard deviation when we have the entire population of interest at hand (rarely the case) *or* we only want to explain features in a sample and not the entire population. However, we typically *want* to explain features of the population based on findings from our sample, for which the sample deviation is more appropriate.

One reason for using the divisor $N - 1$ instead of N is because the observed values within a series lie, on average, closer to the **sample mean** than to the **population mean**. A standard deviation calculated based on deviations from the sample mean *underestimates* the desired standard deviation of the population. By using $N - 1$ instead of N, we control for a (potentially) non-representative sample by making the outcome of the division a little larger.

Additional point and range estimates

Suppose we've collected age data from a student population which looks like the following:

`32, 27, 29, 41, 28, 22, 23, 19, 32, **5**, 31, **3**, 27, 22, 18, 19, 20, 31, 34, 29`

There appears to be at least two values that do not make sense (in **bold**). These may be outliers that skew our series average.

```
st.ages <- c(32, 27, 29, 41, 28, 22, 23, 19, 32, 5, 31, 3, 27, 22, 18, 19, 20, 31, 34, 29)

mean(st.ages)
```

```
## [1] 24.6
```

One way to address this concern is to derive *robust mean* estimates, which involve 'trimming' the edges of our series. For instance, we can remove 10% of the values at either end of the series...

3, 5, 18, 19, 19, 20, 22, 22, 23, 27, 27, 28, 29, 29, 31, 31, 32, 32, 34, 41

```
mean(st.ages,trim = .1) # How much of the series do we want to 'trim'?
```

```
## [1] 25.5625
```

We could trim 20% of the series...

```
mean(st.ages,trim=.2)
```

```
## [1] 25.66667
```

Trimming 10% and 20% of the series give us roughly equivalent values, so we can stick with the former (as less of the original data is lost).

Recall that the inter-quartile range, or *IQR*, describes the difference between the 25^{th} and 75^{th} percentiles of the data. This is conventionally used as the measure of variance when we're interested in estimating group medians.

```
range(st.ages) # Min and Max values within a series
```

```
## [1]  3 41
```

We can estimate the total range

```
max(st.ages) - min(st.ages)
```

```
## [1] 38
```

We can estimate the IQR in one step

```
IQR(st.ages)
```

```
## [1] 11.25
```

Once you become familiar with what these estimates describe, you can use R's built-in `summary` function to view all the descriptives

```
summary(st.ages)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3.00   19.75   27.00   24.60   31.00   41.00
```

Missing values

SUppose your research assistant provides you a new set of age data that you have to provide summary statistics for

28, 22, 23, 19, 32, 5, 31, 3, 27, **NA**, **NA**, 22, 18, 19

For some reason, two values were not entered/collected. This will be a problem when we're trying to run operations on the variable. For example,

```
st.ages2 <- c(28, 22, 23, 19, 32, 5, 31, 3, 27, NA, NA, 22, 18, 19)
mean(st.ages2)
```

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

R does not know what to do if (even) one value is missing from the data. Instead of re-constructing the entire series, we can instruct R to simply *remove* all missing values (described as *NA*)

```
mean(st.ages2,na.rm = T)
```

```
## [1] 20.75
```

We can use this with other functions as well...

```
summary(st.ages2,na.rm = T)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    3.00   18.75   22.00   20.75   27.25   32.00       2
```

Note that `summary` additionally describes the number of NAs in the dataset (which were removed)

Lab activity 2

Cause of death	2018	2019	2020
Heart disease	655381	659041	696962
Cancer	599274	599601	598932
Injuries	167127	173040	192176
Respiratory	159486	156979	156979
Stroke	147810	150005	150005
Alzheimers	122019	121499	133182
Diabetes	83564	87647	101106
Influenza	55672	69081	95219
Kidney diseases	50633	52260	91779

Cause of death	2018	2019	2020
Self-harm/suicide	48344	47511	44834

Have a look at the top ten causes of mortality in the world from 2018, 2019 and 2020, then go through the following instructions to complete this week's lab activity:

1. Create 3 numerical vectors (for the years 2018, 2019, and 2020) with the **raw values** from the **top 7 causes** reported (*not* the percentages).
2. Create 1 factorial vector with each of the **top 7 causes** functioning as a factor level.
3. Combine all vectors into a single table using `cbind.data.frame()`.
4. Report the mean number of deaths & standard deviation for each year, rounded to one decimal value.
5. Construct a boxplot with a BLUE border and a YELLOW background (*hint* - see the `boxfill` argument earlier).
6. Describe whether the average number of deaths (across all causes) has increased or decreased from 2018 to 2020.

Submit all your lab activities into the dropbox before the following class.