Data Science Lab 4: Importing Data

Introduction

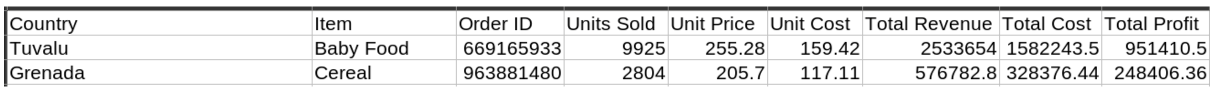
Introduction: In this lab we will examine a csv file of sales records, please download the 100_sales_records.csv file from the Moodle page. The CSV format (comma-separated values) (CSV) is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

fname, lnam
nancy, davo
erin , bora
tony , rapha
⋮
names.csv

Part 1: Visualise the Dataset

Importing Data

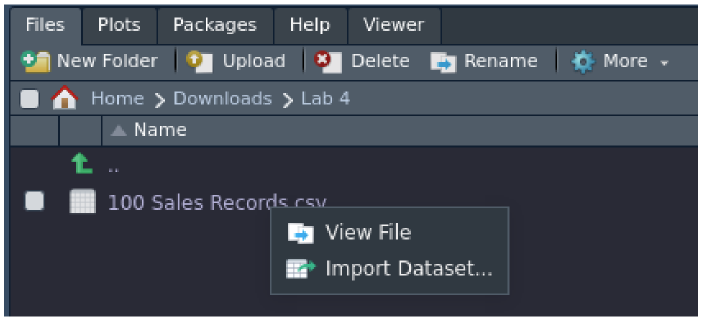
1.1 Use a spreadsheet application and open the csv file to get an idea of how the data is presented. You should see that the first row is the heading and the rest of the rows contain a sales entry with some associated records:



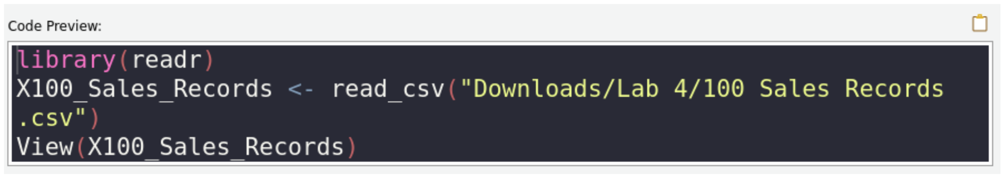
Country	Item	Order ID	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
Tuvalu	Baby Food	669165933	9925	255.28	159.42	2533654	1582243.5	951410.5
Grenada	Cereal	963881480	2804	205.7	117.11	576782.8	328376.44	248406.36

A CSV file is a raw text file so it won't have any formatting like bold text or column justification as you might see in a spreadsheet application. Let's import the data into R for analysis.

1.2 There are many ways to add a file to R, this is only one of them, you should place the csv file in a sensible location then in the lower right window click the files tab and navigate to the csv and click Import Dataset.



1.3 Ensure that the ‘First Row as Names’ box is ticked/checked, you should see a preview of the data along with the header and the data type in R. At this point it would be counter-intuitively tempting to click on import but do not do that, instead we want R to generate the import code for us, you should see a box called ‘Code Preview’. R is going to use a reading library called readr to perform this action, at the top right of the code preview there is a small icon that looks like a clipboard, click this and then close the preview window then paste the results into a new R Script in R studio. The advantage of this is that R will set the path and options and generate the command for us. Make sure the path to the file is correct:



```
library(readr)
X100_Sales_Records <- read_csv("Downloads/Lab 4/100 Sales Records
.csv")
View(X100_Sales_Records)
```

1.4 R has created a variable of the dataset and called it X100_Sales_Records, check the data type/class and structure with the following, we should see the data is a data.frame, this is an R data type that makes the data easy to work with, the str (structure) command will also give us a overview of the columns and the data types in each column.

```
typeof(X100_Sales_Records)
[1] "list"
class(X100_Sales_Records)
[1] "spec_tbl_df" "tbl_df"      "tbl"          "data.frame"
str(X100_Sales_Records)
$ Region          : chr [1:100] "Australia and  $
...
```

1.5 Use the extract operator $ to select a column, we can also get summary statistics using the summary function including mean, median and min cost using the summary command:

```
barplot(X100_Sales_Records$`Unit Cost`)
summary(X100_Sales_Records$`Unit Cost`)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   6.92   35.84  107.28  191.05  263.33  524.96
```

1.6 Use the `mean()` function on the Unit Price column to calculate the mean unit price, use the $ extract operator in your command. Use the class() function to determine the data type of the variables in the Unit Price column, check the type of the variables in the Region column, use the length() function to check the number of entries in the column, obviously the answer should be 100, but this is a handy function to know about.

1.7 So far we have worked with the whole data set and used the extraction operator ($) to pull out specific sections, now let us say we wish to pull out a specific part of the data, this is called subsetting. Subset the data to extract only specific parts of the dataset, to do this we use square brackets, extract all of the sales from the Asia region can work out the mean:

```
mean(X100_Sales_Records$`Unit Price` [X100_Sales_Records$Region ==
"Asia"])
```

Typing out the X100_Sales_Records$... Over and over again can become somewhat tedious. Let's attach the dataset into R's memory so we can just call the columns without the full dataset.

```
attach(X100_Sales_Records)
```

Now when we run Region with no arguments, R knows we are talking about the Region column, try with Country, you should see all 100 of them listed, this will make our commands shorter and easier to read, to remove the dataset from memory we just use the `detach()` function.

1.8 Calculate the mean sales unit price for the Europe region, you may run into some issues with the \`Unit Price\`, because the column name has a space you should enclose it in \`backticks\`.

1.9 Create a new variable with only the European sales records. We need to add the comma at the end of the square brackets to tell R to include all rows, how many records does the new european_sales subset contain, use the length function to check.

```
european_sales = X100_Sales_Records[Region == "Europe" ,]
```

1.10 Create a new variable called european sales high priority where the region is equal to Europe and the Priority is H:

```
european_sales_high_priority = X100_Sales_Records[Region == "Europe"
& `Order Priority` == "H" , ]
```

Now display columns 1 to 4 of the new variable and all columns:

```
european_sales_high_priority[1:4 ,]
```

1.11 Now that we know how to subset the data we can pull out specific sections based on a match or a number of matches but this still pulls every column, if we want to just examine one variable / column in the subset we can use the table command, subset the asian data by priority then pass to a table (there is a space between Order and Priority - it can be hard to see):

```
asian_sales_priority <- X100_Sales_Records[Region == "Asia" & `Order
Priority` == "L" ,]

table(asian_sales_priority$Country)
```

1.12 Now run the barplot to visualise the result, change the priority to M and then to H and resun these commands to see the barplot change:

```
barplot(table(asian_sales_priority$Country))
```

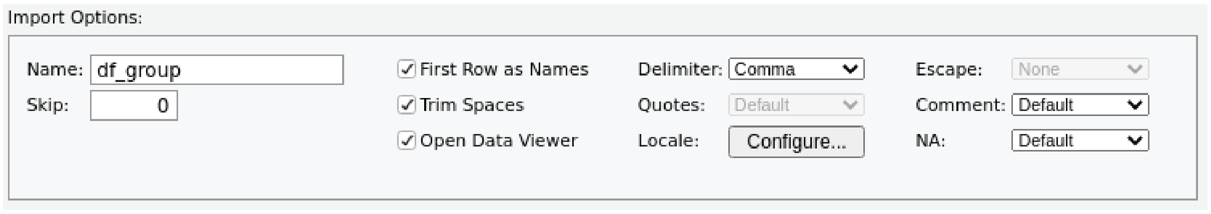
1.13 Create a new variable called european_sales_priority and pass all of the values of Low priority into this variable. Then create a table, and in the table add the country data from the new european_sales_priority variable. Run the plot command on this table, each item should have the same value. We can consider the european_sales_priority data to be a subset of the main data set.

1.14 With the table function, create a table using the whole csv file but only include the country and item type columns in the table, then create a barplot of this table.

Part 2: Group Scores CSV Row Operations and Column Operations

Obtain and load a text file - Data Acquisition / Pre-Processing

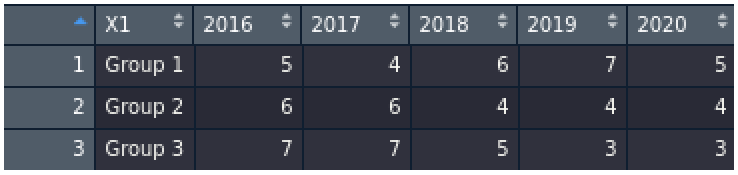
2.1 Download the group_scores.csv file from the moodle page, this is a very simple file that we can use to practice some operations on the dataset. As in Part 1, click on the file in the file explorer in R and choose to import. This time we want to specify the variable as df_group, the settings should look like this:



Import Options:

Name:	df_group	✓ First Row as Names	Delimiter:	Comma	Escape:	None
Skip:	0	✓ Trim Spaces	Quotes:	Default	Comment:	Default
		✓ Open Data Viewer	Locale:	Configure...	NA:	Default

2.2 Copy the import command and then place this at the top of a new R script. You should see the data in the preview window in R. There are 3 groups in column 1 and then years with numeric values in the rest of the columns.



	X1	2016	2017	2018	2019	2020
1	Group 1	5	4	6	7	5
2	Group 2	6	6	4	4	4
3	Group 3	7	7	5	3	3

2.3 Calculate the mean of columns, double square brackets, i.e. [[ ]], can be used to directly access columns. The number of the column can be placed inside the double square brackets, run the min max and range functions on columns 2 and 3.

```
mean(df_group[[2]])
mean(df_group[[3]])

median(df_group[[2]])
median(df_group[[3]])

sum(df_group[[2]])
sum(df_group[[3]])

range(df_group[[2]])
range(df_group[[3]])

min(df_group[[2]])
min(df_group[[3]])

max(df_group[[2]])
max(df_group[[3]])
```

2.4 To calculate the mean of rows is a little bit different, because we have the Group 1 character variable in column A we need to take this into consideration and calculate the mean of the first row of data. To do this we need to pass the values into a vector:

```
my_group_data = (as.numeric(df_group[1,]))
Warning message:
NAs introduced by coercion
```

2.5 Run the vector to see the contents first with no arguments, notice the character variable from column A has been converted to NA. Now we need to remove the NA values from the vector so we can perform calculations on it:

```
my_group_data

sum(my_group_data, na.rm = TRUE)
```

2.6 In practice this could be more annoying to type out than just removing the NA value, this can be done with:

```
my_group_data_new <- my_group_data[!is.na(my_group_data)]
```

Plot the Group CSV

3.1 Run the summary() function on my_group_data_new,then create a barplot of my_group_data_new. Change the colour of the chart:

```
barplot(my_group_data_new, col=“blue”)
```

3.2 Using names.arg, can we add the first row of the csv as names to the barchart?

Firstly lets try and get the names of the columns out to use in our bar chart calling the zeroth row with a comma to indicate for all columns, run this:

```
df_group[0 ,]
```

3.3 We can add this to the barplot command with the names.arg argument inside the barplot function:

```
barplot(my_group_data_new, names.arg = df_group[0 ,])
```

3.4 This creates a problem, with too many names for the number of bars, remember the NA value from column A we removed in part 2.6, we need to do this to our column names, first pass the column names to a vector, then use view to look at the contents:

```
my_name_vector = df_group[0 ,]
View(my_name_vector)
```

3.5 Each item in the vector has an index number, vector indexing in R starts from 1, unlike most programming languages where indexing starts from 0. Have a look at these with the following:

```
my_name_vector[2]
my_name_vector[3]
my_name_vector[2:4]
```

3.6 The previous examples were only to illustrate indexing and the use of the [ ] what we really need here is the column names function `colnames`:

```
barplot(my_group_data_new, col="gold1",
names.arg=colnames(df_group[-1]))
```