# 

# COMP 5070 Statistical Programming for Data Science

# Australian Energy Market

## Assessable Exercise 2

* This exercise is a part of the continuous assessment that is worth 25% of your overall grade.
* Your code should be submitted as a single file – R-script using LearnOnline. You should not submit anything else. Archived files will be ignored.
* Do not hardcode any paths on your computer in the code, as I should be able to load and run your code. All data files are stored on my computer in the working directory.
* Your code should load all required libraries BUT please don’t install anything on my computer. You can safely assume I have all packages already installed.
* The exercise is out of 100 marks. To obtain the maximum available marks you should aim to:

1. Code the requested program (60%). The correct code should use vectorization as much as possible.
2. Use a clear coding style (10%). Code clarity is an important part of your submission. Thus, you should choose meaningful variable names and adopt the use of comments - you don't need to comment every single line, as this will affect readability - however you should aim to comment at least each section of code.
3. Have the code run successfully (10%).
4. Output the information in a presentable manner as decided by yourself (10%).
5. Document code limitations including, but not limited to, the requested functionalities (10%).

This assessable exercise can be openly discussed within the group online and you are welcome to share tips and tricks (not entire programs, however).

Having said that, the ground rules are:

* This is an individual assessment; the code should be your own work. It is OK to use online examples and tutorials for information and inspiration. It is not OK to copy-paste other person’s code.
* If you submit a program cobbled together by other peoples’ code with no, or little, original input from yourself, you will automatically receive a zero mark. The idea is to develop your own programming style with (or without) the help of others, however any code used should support your approach to how you write the program, not replace your own efforts.

Late submission will be penalized by 10-point deduction for each day or part of it after the due date.

If you’re unsure at any point, you’re welcomed to check with me.

# Australian Energy Market

In this exercise you should write a code to load, analyse and create data visualisations for energy market in Australia in 2020.

The data set includes multiple CSV files containing the information about electricity demand and wholesale electricity prices in five Australian states (NSW, QLD, Vic, SA, TAS) in 2020. WA is not a part of Australian energy market, so it is not included in the analysis. Each file has information about one month, so there are 12 files per year per each of 5 states – in total, there are 60 files. The data were downloaded from the Australian Energy Market Operator (AEMO) website - https://www.aemo.com.au/.

All data files have the same structure with the following variables:

• REGION – state name.

• SETTLEMENTDATE – day and time for the recording. Until recently all electricity sales in Australia were executed on 30-minute basis. So, every observation represents a 30-minute block.

• TOTALDEMAND – total electricity demand over 30-minute block in measured megawatts

• RRP – average price per one megawatt of electricity during that block of time

• PERIODTYPE – type of recording; its value is the same everywhere and we don’t need it for analysis

The main purpose of this exercise is to test your understanding of programming in R: loading data, using vectorization, indexing, plotting. For this exercise, lectures 1 and 2 of R materials should be sufficient. However, lecture 3 material is highly recommended – it will make your life much easier.

While you do data analysis, no interpretations/discussions are required. This is mostly a programming exercise. Still, you should be confident that your results and/or data visualisations are meaningful.

**Your tasks are following:**

1. Load and prepare data for the analysis. You need to have one dataframe for all data. It is OK to use for-loop for loading data. Loading data is a slow process, so inefficiencies of for-loop are not important. It is NOT OK to load every file by manually typing their names. See below hints on working with multiple files.
2. Calculate **electricity “demand per capita”**, that is, demand normalised (divided) by state population.

|  |  |
| --- | --- |
| **State** | **Population ('000)** |
| New South Wales | 8,176.4 |
| Victoria | 6,648.6 |
| Queensland | 5,206.4 |
| South Australia | 1,771.7 |
| Tasmania | 542.0 |

1. Aggregate data to get “daily” information. That is, aggregate 30-minute blocks in one-day blocks: one observation – one day. See below hints on working with date and time data. Electricity demand should be summed up to get a total demand per day. Prices should be averaged as it does not make sense to sum up prices per megawatt.
2. Plot comparison graph of distributions of electricity demand per day for each of five states for original “raw” demand and then for demand per capita.
3. Build historical graph for each of five states for demand and then for demand per capita.
4. Study a relationship between prices and demand per capita for all five states.
5. Aggregate data by the hour of the day for each of five states. One might think that demand is higher during the day and lower during the night. You must investigate that. Make a plot of electricity demand for each state per hour. Then another graph for demand per capita per hour for each state.
6. Each graph should be nicely presented, that is, proper title, axis labels, colour legend. Each graph should be accompanied by appropriate numerical summary.
7. Think about the patterns in the data, differences in the graphs – no need to write anything. This is non-graded step.

**Hint on working with Date-Time variables:**

First, you can convert data to POSIX format – special format for datetime variables in R.

temp <- as.POSIXlt("2010/01/21 18:00:00", format = "%Y/%m/%d %H:%M:%OS")

The result is an object of class POSIXlt. After conversion, you can get access to elements of that object, for example

temp$hour # for hour

temp$year # for year. Count starts from 1900.

temp$mon # for month

attributes(temp) # check for all possible attributes

There are many different tools for working with date and time in R, for example package **lubridate**. However, the trick above should do the job for this assessment. If the variable is not a character but already some type of date-time class, you can change it by the same command as.POSIXlt()

Second, you can use function as.Date(). The result is an object of class Date. It keeps information about the date only and drop hours/minutes/seconds information. This is useful if you don't need time information for the analysis.

temp <- as.Date("2010/01/21 18:00:00", format = "%Y/%m/%d %H:%M:%OS")  
print(temp)

If you already have your variable in class POXISlt as before, then it is very easy to convert it to Date – no need for any extra parameters.

temp <- as.Date(your\_POSIXlt\_variable)  
print(temp)

**Hint on working with files:**

There are two options to get names for loading a lot of files:

1. Create names by string concatenation. This is week 11 topic but you can use the following example to create file name for any state and date:  
     
   state <- "NSW"  
   date <- "202001"  
   filename <- paste("file\_", state, "\_", date, ".csv", sep = "")  
   print(filename) # or you can load this file
2. Get the list of all files available in the working directory  
     
   myList <- list.files(pattern = "\\.csv$")  
   print(myList) # then load every file in the list  
     
   Parameter pattern allows you to get only file names ending on ".csv", which might be handy for this assessment. If you remove this parameter, then you get all files in the working directory.