Module 6 Lab

This lab uses the eleven Month-XX.csv files provided via the class Canvas module page.

Exercise 1

Iterate over each of the **Month-XX.csv** files using a list comprehension or `for` loop and (i) import the data as a DataFrame and (ii) compute the total number of observations in each DataFrame. Which DataFrame has the most observations? Which has the least?

Exercise 2

Import the **Month-XX.csv** files into your current Python session. Do so using a `for` loop or list comprehension. Rather than have 11 separate data frames (one for each month), combine these so that you have one data frame containing all the data. Your final data frame should have 698,159 rows and 10 columns.

What is the `Account_ID` value for the very last transaction in this combined DataFrame? **Note**, the last transaction in the combined DataFrame should be the last transaction in the Month-11.csv data set.

Exercise 3

How many unique values exist in each column?

Exercise 4

Create a function `convert_to_qtr()` that converts monthly values to quarters. This function should assess a month value ("Jan", "Feb", ... , "Dec") and convert to "Q1", "Q2", "Q3", or "Q4". Do it such that:

- If the month input is Jan-Mar, then the function returns "Q1"
- If the month input is Apr-Jun, then the function returns "Q2"
- If the month input is Jul-Sep, then the function returns "Q3"
- If the month input is Oct-Dec, then the function returns "Q4"

Apply this function to the imported data (note the `Month` column contains month values in the form of "Jan", "Feb", ... , "Dec") and create a new column that contains the quarter that each observation is aligned to. Compute how many observations fall into each quarter.

Exercise 5

Import the **Month-XX.csv** files into your current Python session. Do so using a `for` loop or list comprehension. However, this time I want you to only import those data sets that have 60,000 or more observations. You'll need to include a conditional statement within your `for` loop or list comprehension to make this happen. Once you've imported the data sets that meet this condition combine them into one single DataFrame. How many observations does this DataFrame have?

Exercise 6

Create a function (`get_data()`) that performs the task in exercise 5. However, this function should have two arguments:

1. `files` which you can use to feed the list of file names to import and
2. `min_req_obs` which you can specify the number of observations required in a data set for it to be imported.

The output of the function should be a single DataFrame containing all observations from those data sets that meet the required observation threshold. Consequently, you should be able to use this function as below to get the same combined DataFrame that you got in exercise 5:

```
get_data(files, min_req_obs=60000)
```

Now use this function to import all **Month-XX.csv** files that contain at least 75,000 observations and combine them into one single DataFrame. How many observations is in this DataFrame?

Exercise 7

Now use the `get_data()` function to iterate over the following values for `min_req_obs` : 25000, 50000, 75000, 90000. How many observations are in the final DataFrames based on these `min_req_obs` values?

Exercise 8

For this exercise and the ones that follow you will not be using any external data sets.
Create a function `divisible(a, b)` that accepts two integers (`a` and `b`) and returns `True` if `a` is divisble by `b` without a remainder. For example, `divisible(10, 3)` should return `False` , while `divisible(6, 3)` should return `True` .

Once you have created this function, apply it to the randomly generated integers below. Be sure to use the same seed value. What is your result?

```
import random

random.seed(123)
a = random.randint(10, 100)
b = random.randint(1, 10)

divisible(a, b)
```

Exercise 9

Create a function `lucky_sum()` that takes all the integers a user enters and returns their sum. However, if one of the values is 13 then it does not count towards the sum, nor do any values to its right.

For example, your function should behave as follows:

```
lucky_sum(1, 2, 3, 4)
10

lucky_sum(1, 13, 3, 4)
1

lucky_sum(1, 3, 13, 4)
4

lucky_sum(13)
0
```

Once you've created this function, apply it to the following randomly sampled list of numbers. Be sure to run the code below exactly as you see it. What is your result?

```
random.seed(18)
my_values = random.choices(range(1, 14), k=26)
lucky_sum(*my_values)
```