DSCI 303 – HW 06 Instructions

General Instructions

Create a new notebook named **`HW_06_YourLastName.ipynb`**. Download the file **`auto_mpg.txt`** into the same directory as this notebook. Complete Parts 1 – 8 described below.

Any set of instructions you see in this document with an orange bar to the left will indicate a place where you should create a markdown cell. For each new part of this assignment, create a markdown cell that indicates the title of that part as a level 2 header.

Any set of instructions you see with a blue bar to the left will provide instructions for creating a single code cell.

Read the problem instructions carefully.

Assignment Header and Import Statements

Create a markdown cell with a level 1 header that reads: "DSCI 303 - Homework 06". Add your name as a level 3 header.

Import the following packages using the standard aliases: **`numpy`**, **`pandas`**, and **`matplotlib.pyplot`**. No other packages should be used in this project.

Part 1: Working with 2D Arrays

In this problem, you will be asked to create a randomly generated 5x8 array and then perform some operations on the rows and columns of this array.

Your code in Part 1 should use array operations and should not include any loops.

Perform the following steps in a single cell:

1. Set a seed of 1.
2. Use **`np.random.uniform`** to create a 5x8 array of elements sampled uniformly from the interval $[0,10]$. Name this array **Z**.
3. Round the elements of **Z** to two decimal places, storing the results back into **Z**.
4. Print **Z**.

Use a single cell to print the 3rd row and 6th column of **Z**. Display your results as shown below. The values replacing the xxxx characters should be left-aligned.

```
Row 3:    xxxx
Column 6: xxxx
```

Use **np.sum()** to calculate the column sum and row sum for **Z**, as well as the sum of the entire array. Display your result with messages in the format shown below. The values replacing the xxxx characters should be left-aligned.

```
Row Sums:    xxxx
Column Sums: xxxx
Array Sum:   xxxx
```

Part 2: Reshaping and Stacking Arrays

In this problem, you will use stacking and reshaping operations to create a 1000x4 array in which each column is drawn from a normal distribution with different parameters.

Your code in Part 2 should use array operations and should not include any loops.

Perform the following steps in a single cell:

1. Set a seed of 167.
2. Create arrays named **x1**, **x2**, **x3**, and **x4**. Each array should be a 1D array with 1000 elements drawn at random from a normal distribution. The parameters for the distributions are as follows:
    - The distribution used for **x1** should have a mean of 50 and a standard deviation of 10.
    - The distribution used for **x2** should have a mean of 20 and a standard deviation of 5.
    - The distribution used for **x3** should have a mean of 100 and a standard deviation of 30.
    - The distribution used for **x4** should have a mean of 10 and a standard deviation of 2.
3. Use **`reshape()`** to create column arrays based on **x1**, **x2**, **x3**, and **x4**. Use **`np.hstack()`** to combine the resulting column arrays into a 1000x4 array named **X**.
4. Round the values in **X** to two decimal places, storing the results back into **X**.
5. Print the shape of **X**.

Print the elements in the first 6 rows of **X**.

Part 3: Standardization

When working with tabular datasets, it is typical for each column in the dataset to represent measurements of a different quantity, with different columns following different distributions and being on different scales. There are many applications in Data Science for which it is useful to have a scaled version of the dataset where all of the columns are on similar scales.

In this problem, you will use numpy operations to create a scaled version of the array created in Part 2. In particular, we will be performing standard scaling, where each column is rescaled to have a mean of 0 and a standard deviation of 1. This is accomplished by subtracting from each column the mean of that column, and then diving the results by the standard deviation of the column.

Your code in Part 3 should use array operations and should not include any loops.

Perform the following steps in a single cell:

1. Use **`np.mean()`** to calculate the column means for **X**. Store the resulting array in a variable.
2. Use **`np.std()`** to calculate the column standard deviations for **X**. Store the resulting array in a variable.
3. Print the results of Steps 1 and 2 in the format shown below. The values replacing the xxxx characters should be left-aligned. The displayed results **(but not the values stored in the variables)** should be rounded to 2 decimal places.

```
Column means:               xxxx
Column standard deviations: xxxx
```

We will now use the arrays calculated in the previous cell to perform the standardization.

Perform the following steps in a single cell:

1. Create an array **W** by subtracting the column means of **X** from **X**, and then dividing the result by the column standard deviations of **X**. You can do this in one step, or break it up into two steps.
2. Use **np.mean()** to calculate the column means for **W**. Store the resulting array in a variable.
3. Use **np.std()** to calculate the column standard deviations for **W**. Store the resulting array in a variable.
4. Print the results of Steps 2 and 3 in the format shown below. The values replacing the xxxx characters should be left-aligned. The displayed results **(but not the values stored in the variables)** should be rounded to 2 decimal places.

```
Column means:               xxxx
Column standard deviations: xxxx
```

Part 4: Load Auto MPG Dataset

The remaining problems in this assignment will involve using pandas to work with the Auto MPG Dataset. This dataset contains information about 398 automobiles manufactured in 1983. You can find more information about this dataset here: Auto MPG Data Set

The data is stored in the tab-separated file **auto_mpg.txt**. Load the contents of this file into a pandas DataFrame named **auto**. Use **head()** to display the first 10 rows of this data frame.

Do not use the **print()** statement to display the DataFrame. When displaying a DataFrame in a notebook, the results will be better if the **print()** function is not used.

Print the shape of the **auto** DataFrame.

Call the **mean()** method of **auto** and print the results. This will display the means for each of the numerical columns in the DataFrame.

Part 5: Regional Counts and Means

Each automobile in this dataset was manufactured in one of three regions: Asia, Europe, or The United States. In this problem, we will calculate the number of observations associated with each region, as well as regional means for **mpg**, **cyl**, and **wt**.

Your code in Part 5 should use DataFrame and array operations and should not include any loops.

Use **np.unique()** to obtain an array of unique values appearing in the **region** column of **auto**. Store the resulting array in a variable named **regions**. Print this array.

Use boolean masking to create three new DataFrames named **asia_auto**, **eur_auto**, and **usa_auto**. Each of these new DataFrames should contain the rows of **auto** associated with one of the values of **region**.

Use **len()** or the **shape** attribute to determine the number of rows in each of the new DataFrames. Print the results in the following formats, with the values replacing the xxxx characters left-aligned.

```
Number of cars manufactured in Asia:   xxxx
Number of cars manufactured in Europe: xxxx
Number of cars manufactured in USA:    xxxx
```

Create three new arrays named **`asia_means`**, **`eur_means`**, and **`usa_means`**. Each array should have three elements, which should be the average values of **`mpg`**, **`cyl`**, and **`wt`** for the relevant region. You can use either the function **`np.mean()`** or the DataFrame method **`mean()`** to calculate these arrays.

Use the arrays to create a new DataFrame named **`mean_df`**. The DataFrame should have three columns named **`mpg`**, **`cyl`**, and **`wt`**. Each row of this DataFrame should represent one of the three regions, and the index for the DataFrame should be set to indicate the name of the region using the previously created **`regions`** array.

Display **`mean_df`** without using the **`print()`** function.

Part 6: Average Weight and MPG by Region

In this problem, we will visually represent the results for MPG and weight from Part 5 in the form of bar plots.

Perform the following steps in a single cell:

1. Create a list named **`colors1`** containing three named colors that display well and are easy to distinguish from one another. (The reason for the 1 in the name is that you will be asked to create a second array of colors in Part 8).

2. Use the DataFrame **mean_df** from Part 5 to create a single figure with two side-by-side bar charts.

   Each bar chart should have three bars, one for each region. The heights of the bars in the left plot should indicate the average MPG of vehicles in each region. The heights of the bars in the right plot should indicate the average weight of vehicles in each region.

   The figure should be created according to the following specifications:
   - The figure size should be `[8,4]`.
   - The bars should be labeled according to the region they represent.
   - The colors of the bars in each plot should be set using the **`colors1`** list.
   - Each bar should have a black border.
   - The title of the left plot should be **"Average MPG by Region"**. The title of the right plot should be **"Average Weight by Region"**.
   - The y-axes of the plots should be labeled **"Average MPG"** and **"Average Weight in Pounds"**.
   - The x-axis of the plots should be labeled **"Region"**.

3. Call **`plt.tight_layout()`** to prevent elements of your subplots from overlapping.
4. Use **`plt.show()`** to display the figure.

Part 7: Relationship between Weight and Miles Per Gallon

In this problem, you will create a scatter plot for each region displaying the relationship between MPG and weight for vehicles manufactured in each region.

Perform the following steps in a single cell:

1. Set a figure size of `[12,4]`.

2. Loop over the **`regions`** list. Each time the loop executes, perform the following steps:
   - Create a new subplot. Your subplots should be arranged in a 1x3 grid.
   - Add a scatter plot displaying the relationship between **`wt`** and **`mpg`** for vehicles manufactured in the region currently being considered in this iteration of the loop. Use **boolean masking** to select the relevant records from the **`auto`** DataFrame. Points should have a black border and a fill color that matches the colors used for the regions in the bar charts in Part 6. Set an alpha level of 0.8.
   - Set the limits of the x-axis to `[1200, 5000]` and the limits of the y-axis to `[0, 50]`.
   - Label the x-axis as **"Weight in Pounds"** and the y-axis as **"Miles Per Gallon"**.
   - Set the title of the subplot to be **"Weight vs MPG (xxxx)"**, with the **xxxx** symbols replaced with the abbreviation for the current region, with the first letter capitalized.

3. Call **`plt.tight_layout()`** to prevent elements of your subplots from overlapping.

4. Use **`plt.show()`** to display the figure.

Part 8: Cylinder Distribution by Region

In this problem, you will create a stacked bar chart displaying the distribution of the number of cylinders in vehicles manufactured in each region.

Use **`np.unique()`** to create a list of unique values appearing in the **`cyl`** column of the **`auto`** DataFrame. Store the result in a variable named **`cyl_values`**. Print this array.

In this cell, you will create a DataFrame with one row for each value of **`cyl`** and one column for each value of **`region`**. Each value in this DataFrame will represent the number of vehicles in the dataset corresponding to the relevant **`cyl`**/**`region`** pair. This can be accomplished by passing the **`cyl`** column and the **`region`** column (in this order) to the function **`pd.crosstab()`**. Store the value returned by this function in a variable named **`cyl_counts_by_region`**. Display this DataFrame (without using the **`print()`** function).

Perform the following steps in a single cell:

1. Start by converting the count information into proportions. Create a DataFrame named **`cyl_props_by_region`** by dividing **`cyl_counts_by_region`** by the column sums of **`cyl_counts_by_region`**. The column sums can be calculated using **`np.sum()`** or the DataFrame **`sum()`** method.

2. We will be creating a stacked bar chart, so we need to know where the bottom of each bar should be located. We can calculate this using the following line of code:
   `bar_bottoms = np.cumsum(cyl_props_by_region) - cyl_props_by_region`

3. Create a list named **`colors2`** containing five named colors that display well and are easy to distinguish from one another.

4. Create a Matplotlib figure, setting the figure size to `[6, 4]`.

5. Loop over the rows of **`cyl_props_by_region`**. Each time this loop executes, add a bar chart to the figure according to the following specifications.
   - The bars should be labeled according to the region they represent.
   - The height of the bars should be determined by the current row of **`cyl_props_by_region`**.
   - The bottom position of each bar should be determined by the current row of **`bar_bottoms`**.
   - Each bar should have a black border, and a fill color determined by the current value of **`colors2`**.
   - The label should be set to the number of cylinders associated with the current row.

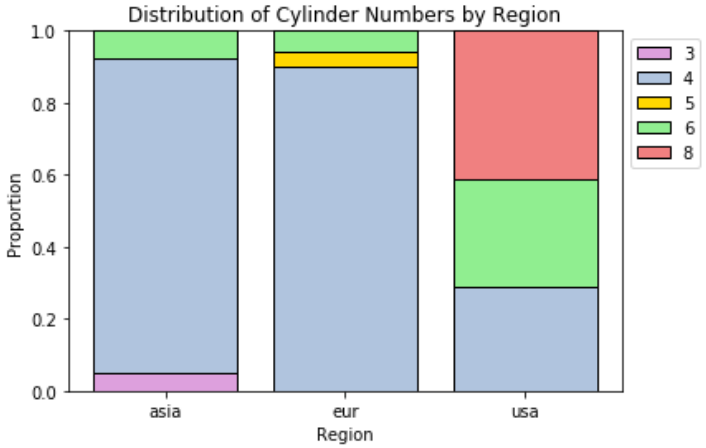
6. Set the labels for the x and y axes to be **"Region"** and **"Proportion"**. Set the title to be **"Distribution of Cylinder Numbers by Region"**.

7. Add a legend to the plot. Set the **`bbox_to_anchor`** parameter to place the legend to the right of the plot, near the top.

8. Display the figure using **`plt.show()`**.

You are not asked to print out **`cyl_props_by_region`**, but if you wish to do so temporarily to check your answer, you should get the DataFrame shown on the left below. Your final plot should look similar to the figure show on the right.

region	asia	eur	usa
cyl			
3	0.050633	0.000000	0.000000
4	0.873418	0.900000	0.289157
5	0.000000	0.042857	0.000000
6	0.075949	0.057143	0.297189
8	0.000000	0.000000	0.413655



Submission Instructions

When you are done, click **Kernel > Restart and Run All**. If any cell produces an error, then manually run every cell after that one, in order. Save your notebook, and then export the notebook as an HTML file. Upload the HTML file to Canvas and upload the IPYNB file to CoCalc, placing it in the **Homework/HW 06** folder.