CS 334: Homework #1

Submission Instructions: The homework is due on Gradescope. A part of your homework will be automatically graded by a Python autograder. The autograder will support Python 3.10. Additional packages and their versions can be found in the `requirements.txt`. Please be aware that the use of other packages and/or versions outside of those in the file may cause your homework to fail some test cases due to incompatible method calls or the inability to import the module. We have split homework 1 into 2 parts on Gradescope: the coding portion and the written answer portion. If *either of the two parts is late, then your homework is late.*

1. **Upload PDF to HW1-Written Assignment**: Create a single high-quality PDF with your solutions to the non-coding problems. The solutions must be typed and make sure to *tag the section* appropriately on your PDF! Your written solutions should complement the code you submit. For example, if you are asked to generate a plot, the written document should have the plot itself and should not have a refer to the code in a.py.

2. **Submit code to the HW1-Code Assignment**: Your submitted code must contain the following files: 'circle.py', 'rectangle.py', 'sumsquare.py', 'palmer.py', 'README.txt'. You should submit *all Python files (no data files please)* to get partial credit for written problems. Note the autograder will only copy the files listed above when running the test cases so make sure they are self-contained (i.e., capable of running standalone). Make sure you always upload *ALL* of these files when you (re)submit. The `README.txt` file *must contain a* ***SIGNED*** *honor statement* that contains the following words:

   ```
   /* THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING CODE
   WRITTEN BY OTHER STUDENTS OR LARGE LANGUAGE MODELS SUCH AS CHATGPT.
   Your_Name_Here */

   I collaborated with the following classmates for this homework:
   <names of classmates>
   ```

!! Important Notes !! The methods should match the **exact** description in the handout and template code, including the **case sensitivity** of any method's name; otherwise, our Gradescope testing scripts will fail and you risk getting 0 points. Do not round and keep as much precision as you can.

1. **Basic Python Programming** (3+3+4+3+2+3+3+4+3+2=30 points): For this problem, you will familiarize yourself with functions and classes in Python using the topic of *shapes*. You will complete the implementation of simple Python classes by filling in the missing methods in each class.

   - Class Circle is defined inside `circle.py` and is meant to represent a circle shape.
   - Class Rectangle is defined inside `rectangle.py` and is meant to represent a rectangle shape.

   (a) (Code) Implement the `area` function in `circle.py`. This should calculate the area of a Circle instance. The return should be a float with the area value. You might find the `math` module helpful for defining π.

(b) (Code) Implement the `circumference` function in `circle.py`. This should calculate the circumference of a Circle instance. The return should be a float with the circumference value.

(c) (Code) Implement the `contains_point` function in `circle.py`. Given an input coordinate `(xc, yc)`, the method should determine whether this point lies within the Circle instance.

(d) (Code) Add 3 print statements in the `main` method of `circle.py` where there is a comment with the `# TODO` tag. The first print statement should print your name. The next 2 print statements should call `contains_point` on the `circle1` instance. The first print statement should pass in coordinates to the function such that `circle1.contains_point` returns `True`. The second print statement should pass in coordinates to the function such that `circle1.contains_point` returns `False`.

(e) (Written) Run `circle.py` from the command line and **copy the output from the terminal into your written document** to obtain full credit. You can run the file as:

```
$python circle.py
```

(f) (Code) Implement the `area` function in Rectangle. This should calculate the area of a Rectangle instance. The return should be a float with the area value. Do not round and keep as much precision as you can.

(g) (Code) Implement the `circumference` function in Rectangle. This should calculate the circumference of a Rectangle instance. The return should be a float with the circumference value. Do not round and keep as much precision as you can.

(h) (Code) Implement the `contains_point` function in Rectangle. Given an input coordinate `(xc, yc)`, the method should determine whether this point lies within the Rectangle instance.

(i) (Code) Add 3 print statements in the `main` method of `rectangle.py` where there is a comment with the `# TODO` tag. The first print statement should print your name. The next 2 print statements should call `contains_point` on the `rectangle1` instance. The first print statement should pass in coordinates to the function such that `rect1.contains_point` returns `True`. The second print statement should pass in coordinates to the function such that `rect1.contains_point` returns `False`.

(j) (Written) Run `rectangle.py` from the command line and copy the output from the terminal into your written document. You can run the file as:

```
$python rectangle.py
```

2. **Vectorization Comparison** (3+4+4+6+3+5=25 points): For this problem, we will perform a speed comparison of two types of code, comparing the non-vectorized code (for-loop) and a vectorized version (Numpy). The two functions should return approximately the same value (there may be subtle differences past 5 decimal places due to floating point differences). The template for the problem is in `sumsquare.py`.

(a) (Code) Implement `gen_random_samples` which given an input parameter `n`, generates a `numpy array` of `n` random numbers using a normal (Gaussian) distribution of mean 0 and variance 1. Hint, you might find `numpy.random.randn` useful.

(b) (Code) Fill in the function `sum_squares_for`. This function must compute the sum of squares using a `for loop` for each element of the array.

(c) (Code) Fill in the code for `sum_square_np`. Compute the sum of squares using `numpy.dot`.

(d) (Code) Implement the function `time_ss` which given an input list of integers (e.g., [1, 50, 100]), will yield a Python dictionary where the keys are 'n', 'ssfor', 'ssnp' and the values are lists. The list associated with 'n' should be the same as the input. The lists associated with 'ssfor' and 'ssnp' should be the time in seconds for the `sum_squares_for` and `sum_square_np` method calls, respectively. In other words, if the input is the list [1,5,100], the return will have the format of {'n': [1,5,100], 'ssfor': [5,6,7], 'ssnp': [2,3,4]}.[1] You will find it useful to use the `timeit` module where you can calculate the execution time of a function in the following manner:

```
start = timeit.default_timer()
function_to_time()
elapsed = timeit.default_timer() - start
```

(e) (Code) Implement the function `timess_to_df` which given an input dictionary formatted as specified by the output in 2d yields a `pandas.DataFrame` that contains 3 columns `n, ssfor, ssnp` (must follow this order and naming convention) that contains the time comparison between `sum_squares_for` and `sum_square_np`.

(f) (Written) Use 2d and 2e to compares the two types of code for n ranging from 10 to 10M. Using the dataframe, create a *single plot* using Python that shows the data points for the two methods. Each set of points (related to either the for loop or the numpy loop) should have a different shape and color, with the legend visible in the plot. Make sure to label both axes of your plot as well. For credit, you **must include your plot in the Written Gradescope submission**. It is not sufficient to submit only the code used to generate the plot in the Code submission.

3. **Pandas Frame** (2+4+4+5=15 points): For this problem, you will become familiar with some of the basic functions and manipulations of `pandas.DataFrame`. We will be using the Palmer Penguins dataset, `penguins.csv`, for this and the following problem. The template code can be found in `palmer.py`. The dataset contains measurements from 344 penguins collected from the Palmer Archipelago in Antarctica. There are 3 species of penguins: Adelie, Chinstrap, and Gentoo. More details of the dataset can be found at `https://github.com/allisonhorst/palmerpenguins?tab=readme-ov-file`. For this problem, make sure you are not hardcoding your solution to the Palmer Penguins dataset, it should work for any comma-separated value (CSV) file.

(a) (Code) Fill in `load_csv` that imports the file located at `inputfile` as a pandas frame. *Do not hardcode the filename*, so if inputfile is a different file, your function will load it as a pandas dataframe.

(b) (Code) Implement the function `remove_na` that given an input dataframe and column name removes the rows with the NaN values and returns the dataframe. If there are no rows with a na for the column, your function should still return a valid dataframe.

[1] These are not actual times from the solution, simply just an example with made-up numbers.

(c) (Code) Fill in the function `onehot` that will take in a dataframe and a column name and performs one-hot encoding of the column specified. The function should return a dataframe that does not contain the specified column and includes the one-hot encoding of that column. One-hot encoding takes a categorical variable with k categories and converts it to k new binary variables where the value in the column represents the boolean (is it that value or not). For example, one-hot encoding of the feature `sex` with 2 categories, `male` and `female`, will result in 2 new features, `male`, `female` where (1,0) will be the values for a male penguin and a (0, 1) will be the values for the female penguin in the 2 new columns.

(d) (Code) Implement the function `to_numeric` where given an input dataframe will drop the non-numeric features and convert the dataframe to a `numpy.ndarray` object. You can assume that NaNs have been removed and any categorical variable that should be kept has been one-hot encoded.

4. **Visualization Exploration** (6+8+10+6=30 points): For this problem, we will explore the visualization capabilities in Python using the Palmer Penguins dataset and the functions you wrote in the above problem. Create your own Python file that contains the commands to load the dataset and plot the features. Make sure to submit this file to HW1-Code and that it is not a part of any of the files above to avoid breaking the autograder. Make sure your code is well-documented. You may want to consider `pandas.DataFrame.boxplot` and `pandas.DataFrame.plot` for the visualization in this problem. Other alternative packages are `matplotlib` or `seaborn`. Each plot should contain sufficient information alone to be able to interpret it. This means there should be appropriate axis names, labels (i.e., axis tick marks with labels at the tick locations), titles, and legends if there are different colors or shapes.

   (a) (Written) Create 2 histograms for the counts of the species: (1) Use the original data (from 3a) and (2) Drop the rows with a NaN in the numeric features (from 3b). Does dropping data with incomplete measurements (i.e., NaN) change the species distribution? Make sure you justify your answer. For full credit, your written assignment **must have the 2 histograms, the answer to the question, and a justification for your answer based on the 2 histograms**.

   (b) (Written) For each numeric feature (i.e., culmen length, culmen depth, flipper length, body mass), plot the boxplot of the distribution of the feature as a function of the species type. In other words, for culmen length, there should be one plot with 3 boxes (i.e., Adelie, Chinstrap, and Gentoo). For full credit, your written assignment **must have 4 boxplots** that have appropriate axes labels, legends (if applicable), and titles.

   (c) (Written) Explore the correlation between all the different pairs of the 4 numeric features by plotting a scatterplot of each feature versus the other 3 features with each species containing a different color or shape (and the legend appropriately labeled). Note that you only need to produce 6 different scatterplots. For full credit, your written assignment **must have 6 plots (or subplots)** that have appropriate axes labels, legends (if applicable), and titles.

   (d) (Written) Based on your exploration of the data in parts 4b and 4c, come up with a set of "rules" that could classify the species type. You **must justify your rules based on the results from 4b and 4c** to receive full credit.