Grading Feedback Cell

IST 718: Big Data Analytics

- Professors: Akit Kumar akumar32@syr.edu
- Faculty Assistant: David Antonio Garcia Flores dgarciaf@syr.edu

General instructions:

- You are welcome to discuss the problems with your classmates but **you are not allowed to copy any part of your answers from your classmates. Short code snippets are allowed from the internet. Code from the class text books or class provided code can be copied in its entirety.**
- **Do not change homework file names.** The professor use these names to grade your homework. Changing file names may result in a point reduction penalty.
- There could be tests in some cells (i.e., `assert` and `np.testing.` statements). These tests (if present) are used to grade your answers. **However, the professor and FAs could use additional test for your answer. Think about cases where your code should run even if it passess all the tests you see.**
- Before submitting your work, remember to check for run time errors with the following procedure: `Kernel` $\rightarrow$ `Restart and Run All`. All runtime errors will result in a minimum penalty of half off.
- Data Bricks is the official class runtime environment so you should test your code on Data Bricks before submission. If there is a runtime problem in the grading environment, we will try your code on Data Bricks before making a final grading decision.
- All plots shall include a title, and axis labels.
- Grading feedback cells are there for graders to provide feedback to students. Don't change or remove grading feedback cells.

```
#Name: Saikumarreddy Pochireddygari
#SUID: 367190390
```

```
# import all needed packages in this cell
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
```

Question 1 (10 pts) Write a function named reverse_by_three that receives a numpy array, python list, or tuple as an input argument and starting at the 2nd from last element, returns every 3rd element of the list in reverse order. It is legal to return an empty python list, numpy array, or tuple if the input. Use slicing syntax exclusively to perform the operations. Your code should be efficient as possible and not use a lot of extraneous un-needed code in the solution.

```
# Create the function reverse_by_three here
def reverse_by_three(input):

  #using isinstance method to check for dtype and returning as o/p per question
```

```
  if isinstance(input, np.ndarray):
    return (input[-2::-3])
  elif isinstance(input, list):
    return (input[-2::-3])
  elif isinstance(input, tuple):
    return (input[-2::-3])

  else:
    if isinstance(input, np.ndarray) and (len(input) == 0):
      return np.array()
    elif isinstance(input, list) and (len(input) == 0):
      return []
    elif isinstance(input, tuple):
      return ()
    else:
      return ("Please correct the input to the function, Accepted input types are Numpy-array or List or Tuple")

```

```
# For grading use only (question 1)
reverse_by_three(np.array([1,2,3,4,5,6]))
```

```
array([5, 2])
```

Grading Feedback Cell

Question 2 (10 pts) Create a python class named my_statistics. The my_statisitics class should require a single numpy array argument in its constructor. Implement the following statistical methods in the my_statistics class: get_mean, get_pop_std, get_sample_std, get_min, and get_max. The get_pop_std method shall return the standard deviation assuming the constructor argument is a complete population. The get_sample_std method shall return the standard deviation assuming that the constructor argument is a sample of a population. The get_min and get_max methods shall return the min and max of the constructor argument respectively. The get_mean method shall return the mean. Use numpy functions to perform the statistical calculations.

```
# create the my_statistics class here
# YOUR CODE HERE
class my_statistics():
  def __init__(self, input_numpy_array):
    self.input_numpy_array = input_numpy_array

  def get_mean(self):
    return np.mean(self.input_numpy_array) #getting mean

  def get_pop_std(self):
    return np.round(np.std(self.input_numpy_array),5) #getting standard deviation-population


  def get_sample_std(self):
    return np.round(np.std(self.input_numpy_array),5) #getting standard deviation-sample

  def get_min(self):
    return np.min(self.input_numpy_array) #getting min of array

  def get_max(self):
    return np.max(self.input_numpy_array) #getting max of array
```

```
# For grading use only (question 2)
stats_data = np.array([12, 17, 9, 13, -5, -7, -9, -11, 39, 42, 17, 2, 99, 1, -310])
my_stats = my_statistics(stats_data)
print(my_stats.get_mean())
print(my_stats.get_pop_std())
print(my_stats.get_sample_std())
print(my_stats.get_min())
print(my_stats.get_max())
```

```
-6.066666666666666
85.59086
85.59086
-310
99
```

Grading Feedback Cell

Question 3 (30 pts) Create a function named monte_hall that takes as an argument the number of times to iterate a monte-carlo simulation of the Monte Hall problem with 4 doors and 1 prize. (https://en.wikipedia.org/wiki/Monty_Hall_problem).
For simplicity, you can assume that the game player always initially chooses door A and the host will open one door. The prize may be behind any of the 4 doors. The monte_hall function should print answers to the following 2 questions: Based on the simulation, what is the probability of winning if you switch doors, and what is the probability of winning if you keep door A? Your function should return these values in a tuple (prob_win_if_switch, prob_win_if_keep).

```
# monte_hall function here
def monte_hall(no_of_times):
    switch_wins = 0
    keep_wins = 0
    for i in range(no_of_times): #looping for iterations
        prize_door = random.randint(1,4) #using random to generate first chance for door
        if prize_door == 1: #checking for chance 1
            host_door = random.choice([2,3,4])
        elif prize_door == 2: #checking for chance 2
            host_door = random.choice([1,3,4])
        elif prize_door == 3: #checking for chance 3
            host_door = random.choice([1,2,4])
        else:
            host_door = random.choice([1,2,3])
        if prize_door == 1: # if the chance 1 and prize is 1 we r switching
            switch_wins += 1
        else:
            keep_wins += 1 # if not we are not switching the door
    prob_win_if_switch = switch_wins / no_of_times #calculating probability of win if we switch
    prob_win_if_keep = keep_wins / no_of_times #calculating probability of win if we dont switch
    return (prob_win_if_switch, prob_win_if_keep)


```

```
# For grading use only (question 3)
print(monte_hall(10000))
```

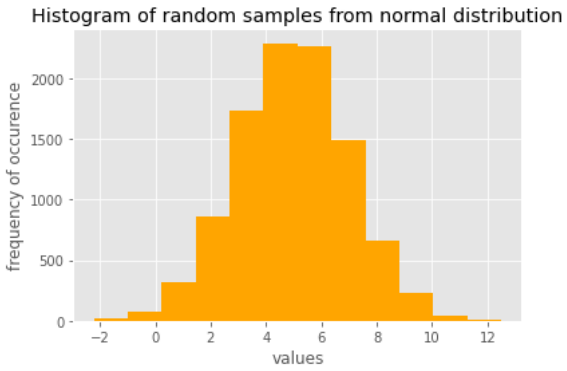
```
(0.2495, 0.7505)
```

Grading Feedback Cell

Question 4 (10 pts) Create a numpy array containing 10,000 samples of random normal data with a mean of 5 and a variance of 4. Plot a histogram of the data using matplotlib.

```
# your histogram plot code here

plt.style.use('ggplot') #using ggplot style
plt.hist(np.random.normal(5,2,10000), bins=12, color='orange') #generating hist plot for normal values
plt.xlabel('values') #labling x axis
plt.ylabel('frequency of occurence') #labling y axis
plt.title('Histogram of random samples from normal distribution') #adding title
plt.show() #showing the result plot
```



Grading Feedback Cell

Question 5 (10 pts) The below cell creates a pandas dataframe called iris_df. Write a function named plot_iris_grid that uses matplotlib to create a grid of 16 scatter plots of all combinatations of the columns in the iris_df data frame. For example, the first row should be sepal len vs sepal len, sepal len vs sepal width, sepal len vs petal len, sepal len vs petal width.

```
from sklearn.datasets import load_iris

# plot_iris_grid code here
iris = load_iris() #loading dataset
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names) #using dataframe method to convert it to dataframe
display(iris_df.head()) #using head function to print 5 rows
```

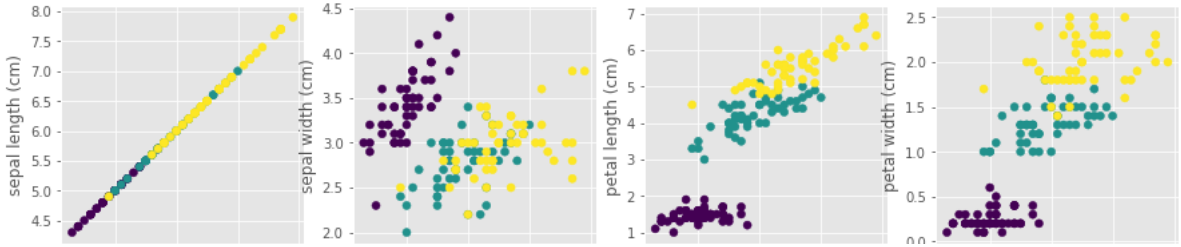
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)

```
iris_df.columns #showing the columns of dataset
```

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

```
# plot_iris_grid code here
def plot_iris_grid(iris_df):
    features = iris_df.columns[:4] #taking features
    fig, axs = plt.subplots(4, 4, figsize=(15, 15)) #defining figure and axis
    for i in range(4): #iteration of 0to3
        for j in range(4): #iteration of 0to3
            axs[i,j].scatter(iris_df[features[i]], iris_df[features[j]], c=iris.target) #plotting two features at i,j axis
            axs[i,j].set_xlabel(features[i]) #adding x label at i axis
            axs[i,j].set_ylabel(features[j]) #adding y label at y axis
    plt.show() #showing plots

plot_iris_grid(iris_df)
```



Grading Feedback Cell

Question 6 (20 pts) Pandas

6a (4 pts): Read the supplied potholes_2016.csv file into a pandas dataframe named potholes. Save the shape of the potholes dataframe in a variable named potholes_shape. Display the potholes_shape variable. Display the head of the potholes dataframe.

```
# the path contains the potholes_2016.csv files
data_file_name = "https://raw.githubusercontent.com/akitkumar24/IST718/main/potholes_2016.csv"

# Create potholes dataframe, potholes_shape variables here
potholes_dataframe = pd.read_csv(data_file_name, na_values=" ") #reading csv file using read csv function
potholes_shape = potholes_dataframe.shape #storing the shape of dataframe using shape method

print(potholes_shape) #printing the shape

print(potholes_dataframe.head()) #printing the head of the dataframe
```

```
(7488, 10)
  StreetNumber StreetName StreetNamePostType Directional  \
0          215  COMSTOCK                AVE         NaN
1          700   MIDLAND                AVE         NaN
2         1604     GRANT               BLVD         NaN
3          261    HOPPER                 RD         NaN
4         1821    VALLEY                 DR         NaN

                        strLocation           dtTime  streetID VehicleName  \
0      215 COMSTOCK AVE & HARRISON S   4/14/2016 8:57  12578124         DP2
1   700-06 MIDLAND AVE & CASTLE ST W   4/15/2016 9:01  12573231         DP1
2  1604-08 GRANT BLVD & WOODRUFF AVE  4/15/2016 13:03  12580306         DP1
3                     261 HOPPER RD  4/18/2016 10:39  12571704         DP2
4     1821 VALLEY DR & CHAFFEE AVE E  4/18/2016 10:52  12571710         DP2

    Latitude  Longitude
0 -76.130140  43.044159
1 -76.154074  43.031314
2 -76.138284  43.072356
3 -76.159681  42.998028
4 -76.152482  42.997837
```

```
# for grading use only (question 6a)
```

Grading Feedback Cell

6b (4 pts): Count the total number of NAN values in the potholes dataframe and store in a variable named total_nan. Print the total_nan variable.

```
# Your nan count code here
total_nan_values = 0 #using a placeholder variable and defining it to 0
for value in potholes_dataframe.isnull().sum(): #iterating over sum of nulls
  total_nan_values += value #updating the placeholder value

print(f"total nan values in data set are :-> {total_nan_values}") #printing the result
```

```
total nan values in data set are :-> 5495
```

```
# for grading use only (question 6b)

```

Grading Feedback Cell

6c (4 pts): Count the number of unique street names in the dataframe and store in a variable named unique_street_name_count. Print unique_street_name_count.

```
# your unique street name count here
print(f"Unique count of street names are :-> {len(set(potholes_dataframe.StreetName.values))}")
#using set to remove duplicates street names and using len function to get length
```

```
Unique count of street names are :-> 413
```

```
# for grading use only (question 6c)
```

Grading Feedback Cell

6d (4 pts): Use the pandas groupby feature to create a new dataframe called street_pothole_sum which summarizes the total number of potholes by street. You are essentially counting the number of rows by street name. The rows of street_pothole_sum should be the labeled with street name. There should be a single column in street_pothole_sum dataframe named num_potholes. Print the head and shape of the street_pothole_sum dataframe.

```
# street_pothole_sum code here
street_pothole_sum = potholes_dataframe.groupby('StreetName') #grouping by streetname

holes_by_street_values = street_pothole_sum.count()['StreetNumber'].values #getting the holes by street
indexes_of_names_holes_by_street = street_pothole_sum.count()['StreetNumber'].index

street_pothole_sum_dataframe = pd.DataFrame({'StreetName':indexes_of_names_holes_by_street, 'num_potholes':holes_by_street_values})
#in above line we are constructing the dataframe of streetname and counts of holes

#In below lines we are printing the reults
print(street_pothole_sum_dataframe.head())
print(street_pothole_sum_dataframe.shape)

```

```
  StreetName  num_potholes
0   ACADEMY             76
1  ACKERMAN              5
2     ADAMS             17
3   AINSLEY              7
4   ALANSON              4
(413, 2)
```

```
# for grading use only (question 6d)
```

Grading Feedback Cell

6e (4 pts) Save the number of potholes on Comstock Ave in a variable named num_potholes_comstock and display the variable. The num_potholes_comstock variable should be an integer type. Print num_potholes_comstock.

```
# num_potholes_comstock code here

potholes_dataframe['StreetName'] = potholes_dataframe['StreetName'].str.strip()
#using strip method because street name has spaces at end
no_potholes_comstock = potholes_dataframe[potholes_dataframe['StreetName']=='COMSTOCK'].groupby('StreetName').count()['StreetNumber']
#in above line we are grouping dataframe on street comstock and geting the count and returing the answer
no_potholes_comstock
```

```
StreetName
COMSTOCK    10
Name: StreetNumber, dtype: int64
```

```
# for grading use only (question 6e)
```

Grading Feedback Cell

Question 7 (10 pts): Create a function named my_corr_coef that takes as input 2 numpy single dimensional arrays and returns the correlation coefficient to the caller. Note that your function does not have to work for matrices. The correlation coefficient is a number of type float between -1 and 1. The my_corr_coef function should not use any built in numpy functions to calculate the correlation. For example, don't use the built in numpy corrcoef function, don't use np.mean, etc. Make sure to check for input error conditions and return the Python 'None' type if the correlation coefficient cannot be computed due to problems with the input data. See equation 3 as a reference on how to compute correlation.

```
# my_corr_coef here
def my_corr_coef(numpy_array_one, numpy_array_two):
  if numpy_array_one.ndim != 1: #checking for dimentionality not equal to 1 for array 1
    return None
  elif numpy_array_two.ndim != 1: #checking for dimentionality not equal to 1 for array 2
    return None
  elif (numpy_array_one.ndim != 1) and (numpy_array_two.ndim != 1): #checking for dimentionality not equal to 1 for array 1,2
    return None
  else:
    if len(numpy_array_one) == len(numpy_array_two): #making sure both array are same size
      return np.corrcoef(numpy_array_one, numpy_array_two) #then we are calculating co-relation
```

```
    else:
      return None

  # in above function we are returing none of conditions arent met as per question
```

```
# for grading use only (question 7)
a = np.array([1,2,3,4,6,7,8,9])
b = np.array([2,4,6,8,10,12,13,15])
my_corr_coef(a,b)
```

```
array([[1.        , 0.99535001],
       [0.99535001, 1.        ]])
```

Grading Feedback Cell

✓ 0s completed at 11:05 PM