3031_A4_Complete_Solutions_v2

October 25, 2021

1 Assignment 4

Instructions:

a. You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and not as a group activity.

b. Do not write your name on the assignment. (1 point)

c. Please include each question (or question number) followed by code and your answer (if applicable). Write your code in the 'Code' cells and your answer in the 'Markdown' cells of the Jupyter notebook. Ensure that the solution is written neatly enough to understand and grade. (1/2 point value of each question)

d. Export your Jupyter notebook as a PDF file. If you get an error, make sure you have downloaded the MikTex software (for windows) or MacTex (for mac). Note that after installing MikTex/MacTex, you will need to check for updates, install the updates if needed, and re-start your system. Submit the PDF file. (1 point)

This assignment is due at 11:59pm on Wednesday, November 3rd. Good luck!

(54 points overall – 52 points for code & answers, 2 points for anonymity and proper formatting)

1.1 Part 1

1.1.1 Use the dataset *movies_cleaned.csv*. We wish to find the 95% confidence interval of Profit for the movies of genre 'Action'. We will use a method known as Bootstrapping to do that.

Bootstrapping is a non-parametric method for obtaining confidence interval. The method is as follows.

(a) Find the profit for each of the Action movies. Suppose there are N such movies. You will have a Profit column with N values.

(b) Randomly sample N values with replacement from the Profit column

(c) Find the mean of the N values obtained in (b)

(d) Repeat steps (b) and (c) 1000 times

(e) The 95% Confidence interval is the range between the 2.5% and 97.5% percentile values of the 1000 means obtained in (c)

Use the following two methods to answer this question:

(1) Without using NumPy, compute the:
(a) 95% Confidence interval of profit for 'Action' movies, and (b) Time taken to execute the code for obtaining the confidence interval

(3 points for code, 2 points for answers)

(2) Using NumPy, and without using loops, compute the:
(a) 95% Confidence interval of profit for 'Action' movies, and (b) Time taken to execute the code for obtaining the confidence interval

(3 points for code, 2 points for answers)

Note that Profit = Worldwide gross – Production budget.

[1]:
```
import pandas as pd
import numpy as np
import time as tm
import random
```

[2]:
```
d = pd.read_csv('movies_cleaned.csv')
```

[3]:
```
d.loc[:,'Profit'] = d.loc[:,'Worldwide Gross'] - d.loc[:,'Production Budget']
```

[4]:
```
profit_vec = d.Profit
```

(1) Without using NumPy

[5]:
```
M=1000
rows,col = d.shape
start_time = tm.time()
mean_vals=pd.Series(M)
for i in range(M):
    sum_inner_loop = 0
    for j in range(rows):
        ind = random.randint(0,rows-1)
        sum_inner_loop += profit_vec[ind]
    mean_vals[i] = sum_inner_loop/rows
mean_vals_sorted = mean_vals.sort_values()
end_time = tm.time()
print(end_time - start_time)
```

```
4.850481986999512
```

[6]:
```
print(mean_vals_sorted.iloc[25]/1e6)
mean_vals_sorted.iloc[975]/1e6
```

```
82.82197102040817
```

[6]:
```
100.61220310204081
```

(a) Confidence interval is [82.6, 100.6], and
(b) Time taken is 3.45s

(2) Using NumPy

[7]:
```
f = lambda x:profit_vec[x]
start_time = tm.time()
sim_values = np.random.randint(0,rows,size = (M,rows))
profit_vals = np.apply_along_axis(f, 1, sim_values)
mean_vals = profit_vals.mean(axis = 1)
mean_vals.sort()
print(mean_vals[25]/1e6)
print(mean_vals[975]/1e6)
end_time = tm.time()
print(end_time - start_time)
```

```
82.93176540306122
100.08956932244898
0.47502803802490234
```

(a) Confidence interval is [82.19, 99.87], and
(b) Time taken is 0.25s

1.2 Part 2

Download the datasets 'percent-bachelors-degrees-women-usa.csv' and 'percent-bachelors-degrees-women-usa-complete.csv' from Canvas.

[8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.impute as skl
import sklearn.preprocessing as pp
```

[9]:
```
#Importing data with missing values
data_missing = pd.read_csv('percent-bachelors-degrees-women-usa.csv')
all_data = pd.read_csv('percent-bachelors-degrees-women-usa-complete.csv')
```

a. Report the number of missing values for each column in 'percent-bachelors-degrees-women-usa.csv'. Which three columns have the most missing values? ***(1 point for code, 1 point for answer)***

[10]:
```
#Finding number of missing values for each column
data_missing.apply(lambda x: sum(x.isnull())).sort_values(ascending=False)
```

[10]:
```
Biology                        6
Engineering                    5
Computer Science               4
Public Administration          1
Psychology                     1
```

```
Physical Sciences                1
Foreign Languages                1
Social Sciences and History      1
Education                        0
Agriculture                      0
English                          0
Communications and Journalism    0
Health Professions               0
Math and Statistics              0
Business                         0
Art and Performance              0
Architecture                     0
Year                             0
dtype: int64
```

Biology, Egineering and Computer Science

b. For which years in the 'Year' column are the values of the 'Biology' column missing? ***(1 point for code, 1 point for answer)***

[11]:
```
data_missing[data_missing.Biology.isnull()].Year
```

[11]:
```
28    1998
29    1999
30    2000
31    2001
32    2002
33    2003
Name: Year, dtype: int64
```

The values of the 'Biology' column are missing from 1998 to 2003

c. Make a scatter plot showing the values of the 'Biology' column on the vertical axis and the 'Year' on the horizontal axis. Make a trendline representing the points on the scatter plot ***(3 points for code)*** Hint: (i) Refer to Lec3_dataViz notes for the trendline, (ii) While making the trendline, do not consider the observations with missing values in the 'Biology' column.

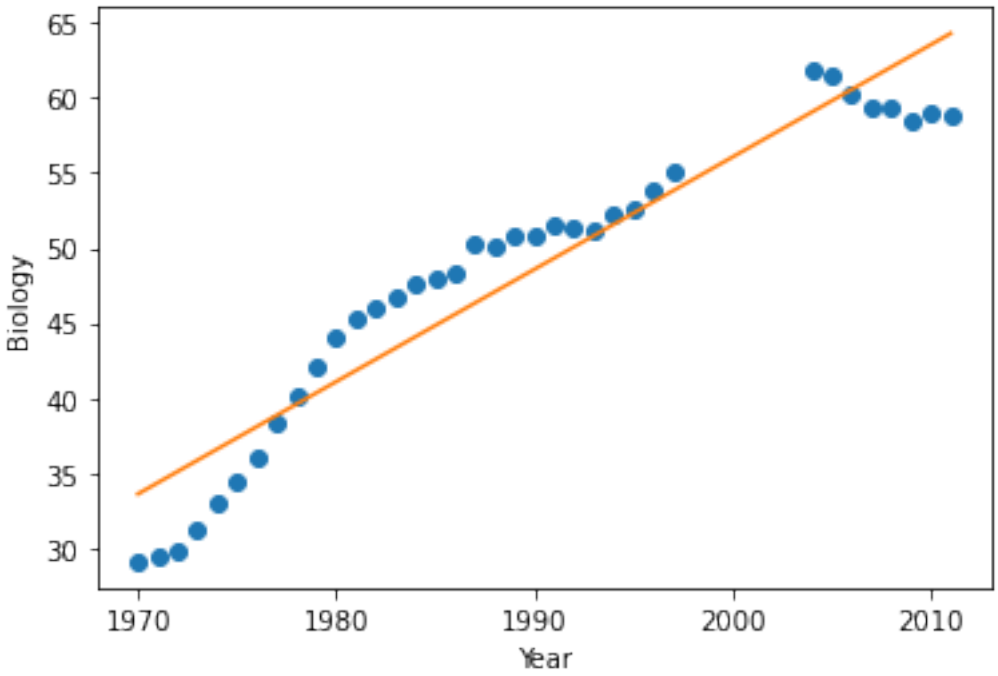
[12]:
```
import matplotlib.pyplot as plt
```

[13]:
```
x = data_missing[~data_missing.Biology.isnull()].Year
y = data_missing[~data_missing.Biology.isnull()].Biology
plt.plot(x,y,'o');
plt.xlabel('Year');
plt.ylabel('Biology');
z = np.polyfit(x, y, 1)
p = np.poly1d(z)

#Plot a trendline
```

```
plt.plot(x,p(x))
```

[13]:
```
[<matplotlib.lines.Line2D at 0x7fbe7b3a9730>]
```



d. Create a copy of the missing data, and call it "imputed_data". Using the trendline, impute the missing values for the 'Biology' column. Fill in the missing values of the Biology column with the imputed values, in the "imputed_data". ***(2 points for code)*** Hint: (i) Refer to Lec3_dataViz notes for the trendline (ii) The function p(x) will provide you an estimate of the value of the 'Biology' column for Year 'x', based on the trendline.

[14]:
```
imputed_values = p(data_missing[data_missing.Biology.isnull()].Year)
```

[15]:
```
data_missing.loc[data_missing.Biology.isnull(),'Biology'] = imputed_values
```

e. Make a scatter plot as follows:

(i) Plot the values of the 'Biology' column on the vertical axis and 'Year' on the horizontal axis *(1 point for code)*

(ii) Color the points for the non-missing values of the 'Biology' column as 'lightgrey' *(1 point for code)*

(iii) Plot the trendline (as in part (c) above) *(1 point for code)*

(iv) Plot the imputed values of the Biology column. Color the points as 'red' *(1 point for code)*

(v) Suppose the actual values for the 'Biology' column for the years 1998-2003 are 56.35, 58.23, 59.39, 60.71, 61.89 and 62.17 respectively. Plot these values, and color the points as 'green' *(1 point for code)*

[16]:
```
actual_values = [56.35, 58.23, 59.39, 60.71, 61.89, 62.17]
```

[17]:
```
imputed_values
```

[17]:
```
array([54.56947821, 55.31855023, 56.06762225, 56.81669428, 57.5657663 ,
       58.31483832])
```

[18]:
```
actual_values
```

[18]:
```
[56.35, 58.23, 59.39, 60.71, 61.89, 62.17]
```

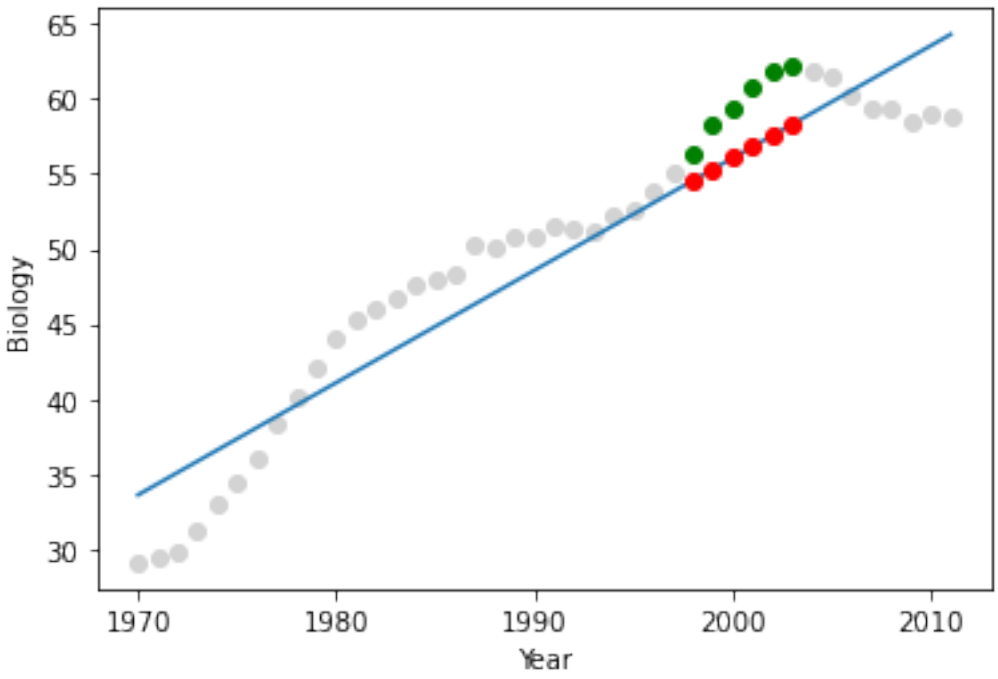
[19]:
```
imputed_values - actual_values
```

[19]:
```
array([-1.78052179, -2.91144977, -3.32237775, -3.89330572, -4.3242337 ,
       -3.85516168])
```

[20]:
```
x = data_missing[~data_missing.Biology.isnull()].Year
y = data_missing[~data_missing.Biology.isnull()].Biology
plt.plot(x,y,'o',color = 'lightgrey');
plt.xlabel('Year');
plt.ylabel('Biology');
z = np.polyfit(x, y, 1)
p = np.poly1d(z)

#Plot a trendline
plt.plot(x,p(x))
plt.plot(range(1998,2004),imputed_values,'o', color = 'red')
plt.plot(range(1998,2004),actual_values,'o', color = 'green')
```

[20]:
```
[<matplotlib.lines.Line2D at 0x7fbe7b5e5430>]
```



f. Compute the RMSE (Root mean squared error) and the MAE (Mean absolute error) for the imputed values. ***(2 points for code)***

[21]:
```
RMSE = np.sqrt((np.square(imputed_values - actual_values)).mean())
print("RMSE = ",RMSE)
MAE = (np.absolute(imputed_values - actual_values)).mean()
print("MAE = ",MAE)
```

```
RMSE =  3.449736454612534
MAE =  3.3478417357563317
```

1.3 Part 3

Find and load the data set "weather_stations.csv" on Canvas. The data is drawn from the city of Chicago's automated sensors at beach weather stations. Each row is a distinct record. Some of the data has been altered to reflect the fact that weather sensor data can sometimes be unpredictable or unreliable.

If a question asks you to alter the data set, please use the updated data set for the following questions. *(24 points overall)*

[53]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

wsd0 = pd.read_csv("weather_stations.csv", index_col=0)
```

1. In this data set the string "-" represents a NaN value. Print all the column names that have at least one "-" as one of their values *(2 pts for code)*

[54]:
```
def check_hyphen(colname):
    a=(wsd0[colname =='-'].shape[0])
    return(a)
b=wsd0.apply(check_hyphen)
wsd0.columns[b>0]
```

[54]:
```
Index(['Station Name', 'Wet Bulb Temperature', 'Rain Intensity', 'Total Rain',
       'Precipitation Type', 'Heading'],
      dtype='object')
```

2. Replace all "-" values with np.nan. Drop all rows where "Station Name" is null and report how many rows were dropped. *(1 pt for code, 1 pt for answer)*

[55]:
```
wsd1 = wsd0.replace("-",np.nan)
wsd1 = wsd1[wsd1["Station Name"].notnull()]

print(wsd0.shape[0] - wsd1.shape[0])
```

```
42
```

Forty-two rows were dropped from the DataFrame.

3. Wind direction shouldn't be above 360. For records with wind direction values above 360, replace the wind direction values with the median wind direction corresponding to the record's weather station. Plot a histogram of wind direction values. *(3 points for code incl. visualization)*

[56]:
```
wsd1.groupby("Station Name")["Wind Direction"].median()
```

[56]:
```
Station Name
63rd Street Weather Station    205
Foster Weather Station         199
Oak Street Weather Station     116
Name: Wind Direction, dtype: int64
```

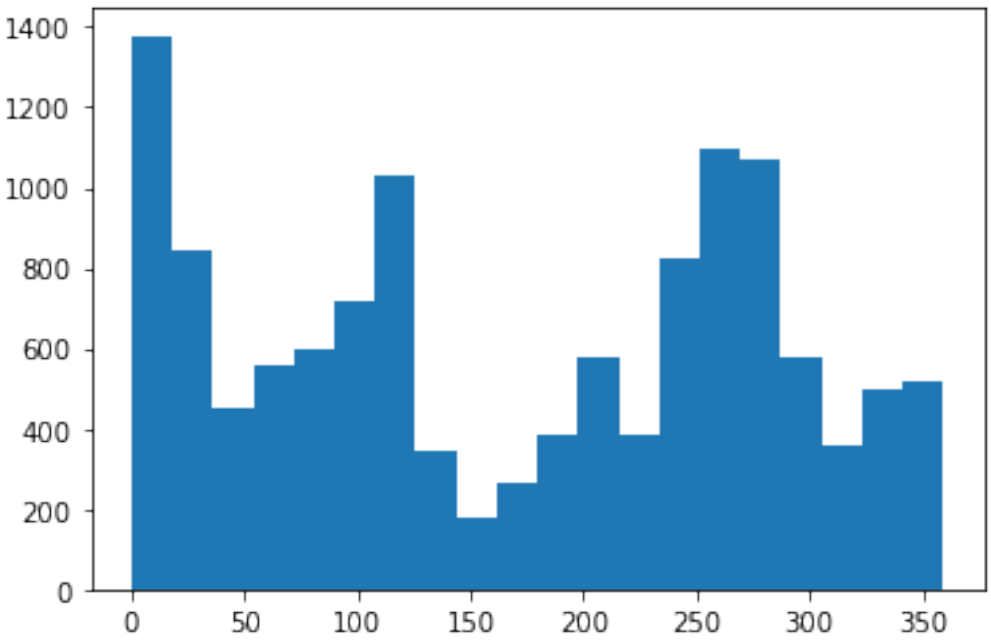
[57]:
```
wsd1.loc[wsd1["Wind Direction"]>360]["Station Name"].value_counts()
```

[57]:
```
Oak Street Weather Station    4
Name: Station Name, dtype: int64
```

[58]:
```
wsd1.loc[wsd1["Wind Direction"]>360,"Wind Direction"] = 116
```

[59]:
```
plt.hist("Wind Direction", data=wsd1, bins=20)
plt.plot()
```

[59]:
```
[]
```



4. Humidity values should not be negative. Calculate the mean humidity using the original data, then drop all rows with sub-zero humidity records. Recalculate and report mean humidity. ***(2 points for code)***

[60]:
```
print(np.mean(wsd1["Humidity"]))
wsd2 = wsd1[wsd1["Humidity"] < 0]
print(np.mean(wsd2["Humidity"]))
```

```
60.2251085669167
-5972.214285714285
```

5. Clearly the mean wind speed during a given hour ("Wind Speed" column) shouldn't exceed that hour's maximum wind speed. Where this discrepancy occurs, consider the mean wind speed an outlier and cap wind speed at its hourly maximum value (i.e. if maximum wind speed is 4.0, and the corresponding mean wind speed is 4.2, cap the mean wind speed by setting it to 4.0). Report the new overall mean wind speed at each weather station. ***(2 points for code)***

[61]:
```
wsd3 = wsd2.copy()
wsd3.loc[:,"Wind Speed"] = np.where(wsd2["Wind Speed"] <= wsd2["Maximum Wind␣
↪Speed"],wsd2["Maximum Wind Speed"],
                                    wsd2["Wind Speed"])
```

[62]:
```
wsd3.groupby("Station Name")["Wind Speed"].mean()
```

[62]:
```
Station Name
63rd Street Weather Station    5.17
```

```
Oak Street Weather Station     3.40
Name: Wind Speed, dtype: float64
```

6. Are there any outliers based on Barometric Pressure? Use the interquartile range rule to find and report how many outliers exist. ***(2 points for code, 1 point for answer)***

[63]:
```
wsd3["Barometric Pressure"].describe()
```

[63]:
```
count      14.000000
mean      994.592857
std         5.052869
min       988.200000
25%       989.975000
50%       994.900000
75%       998.000000
max      1003.600000
Name: Barometric Pressure, dtype: float64
```

[52]:
```
print(wsd3[wsd3["Barometric Pressure"]>(998 + 1.5 * (998-989.975))]["Barometric␣
↪Pressure"].shape)
print(wsd3[wsd3["Barometric Pressure"]<(989.975 - 1.5 * (998-989.
↪975))]["Barometric Pressure"].shape)
```

```
(0,)
(0,)
```

There are no outliers.

7. We'll take a more conservative approach to finding outliers in terms of air temperature. Identify records where air temperature falls more one standard deviation from its mean value. How many such outliers are there? ***(2 points for code, 1 pt for answer)***

[34]:
```
wsd2["Air Temperature"].describe()
```

[34]:
```
count    14.000000
mean     16.535714
std       6.197097
min       6.100000
25%      11.650000
50%      15.500000
75%      21.700000
max      28.500000
Name: Air Temperature, dtype: float64
```

[35]:
```
wsd2[np.abs(wsd2["Air Temperature"]-15.988945)>8.043672].shape
```

[35]:
```
(2, 16)
```

In this case there are 4,857 outliers.

8. We're interested in classifying solar radiation, given in Watts per square meter. Define any value at or below 8 as "Low Solar Radiation," any value above 8 and up to 200 as "Moderate Solar Radiation," and any value above 200 as "High Solar Radiation." Use binning to create a new column classifying each record's solar radiation. What percentage of records are in each bin? ***(2 points for code, 1 point for answer)***

[36]:
```
wsd2["Solar Radiation"].describe()
```

[36]:
```
count     14.000000
mean     146.857143
std      277.712873
min       -5.000000
25%        2.000000
50%        3.500000
75%       70.500000
max      860.000000
Name: Solar Radiation, dtype: float64
```

[37]:
```
bins = [-10,8,200,1000000]

solar_cats = pd.cut(wsd2["Solar Radiation"],bins)

solar_cats.value_counts()/len(solar_cats)
```

[37]:
```
(-10, 8]           0.571429
(8, 200]           0.214286
(200, 1000000]     0.214286
Name: Solar Radiation, dtype: float64
```

49.87% are low solar radiation, 23.87% medium solar radiation, and 26.27% high solar radiation.

9. Using the "Measurement Timestamp" column, create new columns based on month, day of month, year, and time. ***(2 points for code)***

[46]:
```
wsd3["Time"] = (wsd3["Measurement Timestamp"]).str.split(expand=True)[1]
wsd3["Month"] = (wsd3["Measurement Timestamp"]).str.split(expand=True)[0].str.
 ↪split("/",expand=True)[0]
wsd3["Day"] = (wsd3["Measurement Timestamp"]).str.split(expand=True)[0].str.
 ↪split("/",expand=True)[1]
wsd3["Year"] = (wsd3["Measurement Timestamp"]).str.split(expand=True)[0].str.
 ↪split("/",expand=True)[2]

wsd3.head(3)
```

[46]:

Column1	Station Name	Measurement Timestamp	Air Temperature	\
1259	63rd Street Weather Station	5/10/20 14:00	6.1	
1402	Oak Street Weather Station	5/13/20 13:00	11.4	
1519	63rd Street Weather Station	5/16/20 1:00	10.7	

```
        Wet Bulb Temperature  Humidity Rain Intensity  Interval Rain  \
Column1
1259                     5.4       -92              0            0.0
1402                     7.8       -61              0            0.0
1519                    10.7      -100              0            0.0

        Total Rain Precipitation Type  Wind Direction  …  \
Column1                                                   …
1259          8.20                  0              16  …
1402          4.60                  0              23  …
1519         73.10                  0              17  …

         Maximum Wind Speed  Barometric Pressure  Solar Radiation  Heading  \
Column1
1259                   13.6                991.7               54      350
1402                    1.4               1001.6              860      357
1519                    8.1                995.3                3      350

        Battery Life   Time Month Day Year  Season
Column1
1259            11.9  14:00     5  10   20  Spring
1402            12.0  13:00     5  13   20  Spring
1519            11.8   1:00     5  16   20  Spring

[3 rows x 21 columns]
```

10. Suppose we're interested in comparing beach weather during different seasons. Let's call March/April/May "Spring," June/July/August "Summer," September/October/November "Fall," and December "Winter." Create dummy variables for the four seasons. ***(2 points for code)***

[48]:
```
wsd3.loc[:,"Season"] = wsd3.loc[:,"Month"].map({"3": "Spring","4":"Spring","5":
↪"Spring","6":"Summer","7":"Summer",
                  "8":"Summer","9":"Fall","10":"Fall","11":"Fall","12":
↪"Winter"})
pd.get_dummies(wsd3["Season"],prefix="Season")
```

[48]:
```
         Season_Fall  Season_Spring  Season_Summer
Column1
1259               0              1              0
1402               0              1              0
1519               0              1              0
1743               0              1              0
1923               0              1              0
2124               0              1              0
2184               0              1              0
2273               0              0              1
```

2467	0	0	1
3386	0	0	1
4382	0	0	1
5202	0	0	1
8943	1	0	0
8961	1	0	0