MCIS6273 Data Mining (Prof. Maull) / Fall 2021 / HW2b

This assignment is worth up to 15 POINTS to your grade total if you complete it on time.

Points Possible	Due Date	Time Commitment (estimated)
15	Sunday, November 7 @ Midnight	*up to* 20 hours

- **GRADING:** Grading will be aligned with the completeness of the objectives.
- **INDEPENDENT WORK:** Copying, cheating, plagiarism and academic dishonesty *are not tolerated* by University or course policy. Please see the syllabus for the full departmental and University statement on the academic code of honor.

OBJECTIVES

- Continue practicing exploratory data analysis and visualization
- Perform a clustering analysis using k-means

WHAT TO TURN IN

You are being encouraged to turn the assignment in using the provided Jupyter Notebook. To do so, make a directory in your Lab environment called `homework/hw2b`. Put all of your files in that directory.

Then zip that directory, rename it with your name as the first part of the filename (e.g. `maull_hw2b_files.tar.gz`), then download it to your local machine, then upload the `.tar.gz` to Blackboard.

If you do not know how to do this, please ask, or visit one of the many tutorials out there on the basics of using in Linux.

If you choose not to use the provided notebook, you will still need to turn in a `.ipynb` Jupyter Notebook and corresponding files according to the instructions in this homework.

ASSIGNMENT TASKS

(25%) Continue practicing exploratory data analysis and visualization

In the last HW we explore some of the basic features of Pandas with graphic and data selection. This time we're going to go a but deeper into Pandas ans learn about MultIndices and grouping data in interesting and useful ways.

Power weightlifting (powerlifting) is an international sport that invites advanced amateurs and professionals alike. Fortunately, there are datasets for the multitude of powerlifting competitions around the world, and they are openly available for curious data scientists like ourselves who would like to ask interesting questions and find interesting relationships in the data. Whether you're into the sport or not, I think there are a variety of interesting phenomenon in the data that make it both tractable and interesting from just a data perspective.

DATA

OpenPowerLifting.org is a large set of data for a multitude of data related to powerlifting competitions around the world. The core data live at the following open source repository on gitlab.com/openpowerlifting/opl-data.

One of the things that we will learn from the data is that the majority of it are interesting over several dimensions. There are the years of competition, the sex of the competitors, the age the competitors, country of origin, among other things. With denser data like these, we want to understand some of the underlying groupings for easier access to the data. For example, one might want to understand how groupings by year and age bear out on the data to explore questions like "Has the number of competitors over 40 increased over the years?" This might be

an interesting question to ask to explore if powerlifters continue to compete as they age since the sport is very difficult on one's body and requires intense continuous training to stay competitive.

Some questions like these are also very useful to explore visually, so we'll dive into a few more graphical techniques to get at these answers and more. We're going to end up with a DataFrame that will group our data by year, age class and sex, so we can see some of the interesting annual trends along each of these dimensions within the last two decades.

§ BUILD THE DATASET

We've learned CSV is common file format for data and we will be working the files large text files in Gitlab to do the work we need. The task is to explore the repository and build up a dataset of 15 random lifting meets from 2019 using BeautifulSoup and the tools in Pandas to put these datasets together.

Oridinarily, we would use a technique often known as "crawling" and is consider by some to be a flagrant violation of good web etiquette. However, the technique is still often the only way to obtain data en masse from a single source. If this were an FTP server, the same pattern could be applied and would not be considered unusual to do so. Of course, you must use this with caution, as it can result in IP throttling and IP blocking, so please use it within the licensing terms of both the data and website you are obtaining data from. Good web citizens restore trust in providers and administrators alike, so throttling yourself after your own requests with code like time.sleep(2) (which will pause your code for 2 seconds), will show that you can behave responsibly.

Because we have to employ specific techniques when crawling datasets from dynamic web pages made with Javascript, and we're in a good position to just get the data we need from the ZIP file in the repository, we'll just assume we have already downloaded it and use the data provided in the `data/` folder which is just a random subset of 40 folders from the repository.

You will take the random data and build yet another dataset of just the data from 2019. See the supplemental notebook to see how to do this, but the easiest is to use the Python `glob` module.

Load all the 2019 datasets into a single Pandas DataFrame.

Your DataFrame should have around 21K rows.

§ FILTER AND EXPLORE THE DATA

Let's first get a feel for the data and filter it down. One of the main difficulties in dealing with large user-contributed data sets like these are *data consistency* and *data quality*. *Data consistency* refers to how data is represented over time. We can see how this becomes an issue when we look at the `Division` column of the dataset. We can see with a relatively untrained eye to the data, that something is very wrong with the consistency — there are nearly 400 Division designations!

When look at it more closely, there are groupings that overlap. For example, you will see `Masters 45-49` and `Masters 40-49` when you perform a `.value_counts()` on the `Divisions` column of the data (see supplemental notebook). What is the difference between these two since they obviously overlap? Coming from the outside, we might not easily answer that question.

We'll perform a few exploratory exercises to get a feel for the data.

Answer the following questions in your notebook:

1. What are the densities (counts) of participants in the top 5 `Division`s?
2. How many `M` and `F` sex values are in the dataset?
3. After dropping `NaN` values, what are the top 5 countries in the data, by number of datapoints?
4. For the country of *Belarus*, how many participants are there in this data?
5. Out all the data, how much of it is missing all of `Age`, `BirthYear` and `BirthDate`? Give both the raw number and the percentage.

§ CLEAN THE DATA

From the previous section, you learned that a good percentage of the data is lacking age information. You also learned that between `Age`, `BirthDate`, and `BirthYear`, there are quite a few gaps, but that the age coverage could be increased if we took the time to do so.

We are going to fill in those gaps the easy way – by making sure that `Age` has the data it needs as we are going to do analysis over that in the subsequent parts of the assignment (and it is one of the fields we can actually do something about).

In this part, you will write a function and use the `apply()` method in Pandas to fill in the missing data.

You will need to:

- find all data missing `Age`, but has at least `BirthYear` and `BirthDate`.
- calculate age by either subtracting `BirthYear` from 2019 **or** by subtracting the year part from `BirthDate`.
- in your final DataFrame, make sure Age is an `int` type.
- show all the steps in your notebook to get full credit.

After doing that answer the following questions:

1. How many new datapoints did you add to the dataset after filling in the missing Age values?

§ GROUP THE DATA

Pandas provides superior capabilities to slice and group data. We would like to build answer some questions of the 2019 data.

You will need to study the following resources to complete the questions in this section:

- `Dataframe.query()`
- `DataFrame.groupby()`. A useful hint when using this function is that you can pass to the first parameter of `DataFrame.groupby()` the list of the grouping in order of grouping, outer group first. Thus, `groupby(['Sex', 'Division]`) will return the MultiIndex DataFrame with `Sex` as the outermost group and `Division` the inner group.

NOTE: you will use the filtered data from the previous section – you do not want to include the missing `Age` data in your analysis

Answer the following question for full credit on this part:

1. How many `M` sex participants are there in the `Open Pro` Division?
2. What is the median age of all participants in the `Open` Division?
3. What is the mean `BodyweightKg` in the `Open` Division men (`M`)?
4. How does this compare with `Open` Division women (`F`)?
5. What is the correlation coefficient (Pearsons r is fine) between `Best3BenchKg`? Would you consider this to be a positive, negative or no correlation?

§ VISUALIZE

Now that we have the data segmented the way we'd like, let's visualize it in some interesting way.

With powerlifting there are a number of ways to express the *strength* of a competitor. There is *raw* strength, meaning how much total weight was lifted on a given lift, and there is *relative* strength. It is not fair to compare the raw lift of a 100lb 16 year old teenage to that of a 35 year old 300lb adult.

The 35 year old might lift ten times the weight yet the 16 year old may be *relatively* stronger, but how would we compare their *relative* strengths? Many competitors will be able to lift between 2 and 7 times their body weight depending on the lift, so we might expect a 100lb powerlifter to perhaps perform a 200lb bench press and maybe a 300lb squat, both impressive for their weight. To deal with comparing *strength* across age and weight variables a number of methods have been developed to create fair and accurate measures of *relative strength*. The OpenPowelifting dataset includes three such measures: *Wilks*, *McCulloch* and *Glossbrenner*, which give a numeric assignment of relative strength which factor age and weight into the computation. Exploring the details of each of these methods is beyond the scope of this homework, but the curious can learn more on the variety of sites which calculate these statistics.

We will restrict our interest to the *Glossbrenner* score, which takes into account age and weight to compute a normalized weight value. Consider three competitors, all 29 year olds, with one male and one female weighing

131.84lbs and the last male weighing 263.67 pounds. Assume they all lift 639.33 pounds total. The *Glossbrenner* score takes into account the age and weights and produces a relative score with the following:

competitor	sex	age (lbs)	weight (lbs)	lift (lbs)	score	γ-coefficient	γ_{age}-coefficient
1	F	29	100	639.33	457.28	0.7152	1.0
2	M	29	100	639.33	373.94	0.5849	1.0
3	M	29	100	639.33	373.94	0.5849	1.0
4	M	49	100	639.33	416.20	0.5849	1.113
5	F	49	100	639.33	508.95	0.7152	1.113

The *Glossbrenner* score is in the *score* column and the γ-coefficient is the constant calculated by the method. The γ_{age}-coefficient is a constant which the method factors in for the relative impact age has on the competitor. Thus, the *Glossbrenner* score Γ is:

$$\Gamma(age, sex, weight) = \gamma\text{-coefficient}_{weight,sex} \times \gamma_{age} \times weight$$

You will need to use the code in the OpenPoweLifting repository to complete the calculation.

You can download the `coefficient.py` file to the directory where your notebook lives. Then you can simply `import coefficient` and use it like in this example:

```
import coefficient

for i, d in enumerate(data):
    if d[0]:
        g_coeff = coefficient.glossCoeffMen(d[2])
    else:
        g_coeff = coefficient.glossCoeffWomen(d[2])

    gb_score = \
        g_coeff  *\
        coefficient.AGE_COEFFICIENTS[d[1]] *\
        d[3]
```

Now that we have that out of the way, let's visualize some data. Specifically, we'd like to plot the *Glossbrenner* score for the last 20 years over time. Are the scores going up, down or staying the same? One could expect any of these scenarios to occur, so let's dive in.

What we want to produce are two *area* plots of the annual *mean Glossbrenner* score from 1999 to 2018 for all age groups, one plot for males and the other for females as putting them all on one graph would most certainly be information overload. To do this we will need to slice the data in a way that makes a multi-index grouped by year, age group and sex.

You will the data in `data/1999_to_2018` to do this and make a visual of it.

Your area plot code will be invoked by:

```
DataFrame.plot.area()
```

You may optionally pass in the `figsize=(15,7)` (or whatever dimensions you'd like) to stretch the data out a bit so you can visually see what is going on, since the legend may get in the way of viewing the data.

Your plot will look something like this:

Please see the `DataFrame.plot.area()` method for full information on the area plots.

A final important note: You will need to use `droplevel()` and `unstack()` in order to prepare your DataFrame for final presentation. Basically, you'll need to drop the `Sex` level of your index (level 2) and immediately before you plot the area plot you will use `unstack()`.

Your notebook must show:

- the *Glossbrenner* score in a new column that implements the calculation for the parameters required. You can drop all other columns if you like.
- an area plot showing the male data grouped by age and year, that is the x-axis will show the year and the y-axis the *Glossbrenner* score,
- an area plot showing the female data grouped by age and year, that is the x-axis will show the year and the y-axis the *Glossbrenner* score.

You must answer the following question directly in your notebook:

1. What's the general trend you see in the area plots? Your answer can be in one or two sentences.

(50%) Perform a clustering analysis using k-means

The simplicity and power of k-means algorithm makes it one of the best to start with when performing *unsupervised learning* — that is the class labels of your data are not known *a priori* and you that will not be training the algorithm on labeled data. While this is a powerful and oft useful technique, use it with care as the initial conditions of the algorithm do not guarantee a global maximum and as such, running the algorithm with a number of initialization points will produce better and more reliable results.

Continuing with our OpenPowerlifting data, we're going to do some exploratory data analysis to examine this dataset in some interesting ways using unsupervised learning, namely clustering. The original dataset has over 1 million data points, but in order to get a good idea of what's in it, we will not need to go back through the entire dataset, and in fact, we will restrict the focus of our energy on just the last 2 decades from 1999.

REMEMBER TO MAKE SURE TO SHOW ALL YOUR WORK IN THE NOTEBOOK SO YOU CAN RECEIVE PARTIAL CREDIT WHERE APPROPRIATE!

§ PREPARE FOR CLUSTERING

You will need to complete part 1 of this homework to filter the data to the necessary subset for this part. The subset of features will just be the following:

```
features = [
    'Sex',
    'Age',
    'BodyweightKg',
    'Best3SquatKg',
    'Best3BenchKg',
    'Best3DeadliftKg',
    'TotalKg'
  ]
```

Final preparation for clustering will require you to turn all of *categorical* variables into *numeric* one's. One way to do this from directly within Pandas is to use `Pandas.get_dummies(your_dataframe)`. You can also study the `sklearn.preprocessing.OrdinalEncoder()` which will do something very similar. Either way, with the reduced set of features above, the only categorical variable will be `Sex` as all the others should already be numerical features.

In your notebook, you should show:

- clearly how many features are now in your dataframe?
- that you are using the **cleaned** data on `Age`
- you should also remove any rows where `NaN` is in any of required columns (hint: `dropna()`)

§ PERFORM SILHOUETTE ANALYSIS

In class we talked about the fact that the k number of clusters needs to be determined *a priori* — that is you will need to know how many clusters beforehand to run the algorithm. To find the optimal k, we will use a method called the *silhouette score*.

Adapt the following code to compute the silhouette scores on *only* the dataset filtered by the features from the prior step.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

Sum_of_squared_distances = []
K = range(2, 15)
for k in K:
    km = KMeans(n_clusters=k, n_init=20)
    km = km.fit(YOUR_OPENPOWERLIFTING_DATAFRAME_WITH_DUMMY_VARS)
    Sum_of_squared_distances.append(km.inertia_)

    silh_score = silhouette_score(YOUR_OPENPOWERLIFTING_DATAFRAME_WITH_DUMMY_VARS, km.labels_)
    print("k = {} | silhouette_score = {}".format(k, silh_score))
```

The largest score is typically the k you go with. If $k = 2$ is your largest score, we will ignore and use the next best score since 2 clusters is not usually an interesting number of clusters when dealing with a large set of data points.

Your notebook must show and answer the following:

1. What is the optimal k according the silhouette score?
2. What else is interesting about the scores?

§ CLUSTER INTERPRETATION

Now that you have clusters and optimal cluster, let's find out the characteristics of the features that dominate them.

Note that the k-means algorithm returns the cluster centers for each cluster, hence in that center each feature value is the *representative feature value* for that cluster. For example, the `TotalKg` would be the representative `TotalKg` for that cluster.

Using the optimal cluster size from the silhouette score in the prior section, please use adapt the following code to determine the cluster characteristics.

```
optimal_k = THE_OPTIMAL_SILH_K

km = KMeans(n_clusters=optimal_k, n_init=150)
km = km.fit(YOUR_OPENPOWERLIFTING_DATAFRAME_WITH_DUMMY_VARS)

for i in range(0, optimal_k):
    l = list(zip(YOUR_OPENPOWERLIFTING_DATAFRAME_WITH_DUMMY_VARS.columns, \
                 km.cluster_centers_[i]))
    l.sort(key=lambda x: x[1], reverse=True)

    print('CLUSTER : {}\n'.format(i))
    for attr, val in l[:]:
      print('\t{} : {}\n'.format(attr, val))
```

Your notebook must show and answer the following:

1. for each cluster, describe in real words what the cluster centers are telling you about the representative of that cluster. For example, your answer might look like: "for cluster 1, the representative for that cluster is a 24.7 year old female, with an average `Best3SquatKg` of 121 and a `TotalKg` of 721",
2. show the output of the cluster centers above.

NOTE: The order of the features in `km.cluster_centers_` are the same order as they exist in the DataFrame.