Exercise 2

Due Friday May 26 2023.

Some files are provided that you need below: E2.zip. As with last week, **you may not write any loops** in your code.

Plotting Wikipedia Page Views

For this question, we will use some data on the number of times individual Wikipedia pages were viewed in two particular hours. These were extracted from Wikipedia's page view counts *(https://dumps.wikimedia.org/other/pagecounts-ez/)* .

To get the two provided `pagecounts-*.txt`, I selected some English pages at random from the first full data set. Then I selected those pages from the second data set. The result is that some of the pages in the first data set are not in the second: those pages weren't viewed in the second hour

We will produce two plots of the data provided with a stand-alone **Python program `create_plots.py`**. The filenames you operate on must be taken from the command line. Your program must run with a command like this:

```
python3 create_plots.py pagecounts-20190509-120000.txt pagecounts-20190509-130000.txt
```

To get the command line arguments (as strings), you can use the built-in `sys` module:

```
import sys
⋮
filename1 = sys.argv[1]
filename2 = sys.argv[2]
```

The files contain space-separated values for the language, page name, number of views, and bytes transferred. You can get the data out of this file format something like this:

```
pd.read_csv(filename, sep=' ', header=None, index_col=1,
        names=['lang', 'page', 'views', 'bytes'])
```

We will produce a single plot with two subplots on the left and right. Matplotlib can do that with a skeleton like this:

```
import matplotlib.pyplot as plt
⋮
plt.figure(figsize=(10, 5)) # change the size to something sensible
plt.subplot(1, 2, 1) # subplots in 1 row, 2 columns, select the first
plt.plot(…) # build plot 1
plt.subplot(1, 2, 2) # ... and then select the second
plt.plot(…) # build plot 2
```

Plot 1: Distribution of Views

For the first plot, we will use **only the first data set**. Based on statistics knowledge gained from blog posts and YouTube videos, I believe the distribution of page views should be a Pareto distribution *(https://en.wikipedia.org/wiki/Pareto_distribution)* .

Let's have a look: using only the first input file, sort the data by the number of views (decreasing). [Hint: `sort_values`.] If you give `plt.plot` a single data set, it will be plotted against a 0 to n-1 range, which will be what we want.

But, if we give matplotlib a Pandas Series (like `data['views']` will be), it will try to use its index as the x-coordinate. To convince it to do otherwise, we have two options: (1) pass the underlying NumPy array (`data['views'].values`), or (2) create a range to explicitly use as the x-coordinates (with `np.arange`).

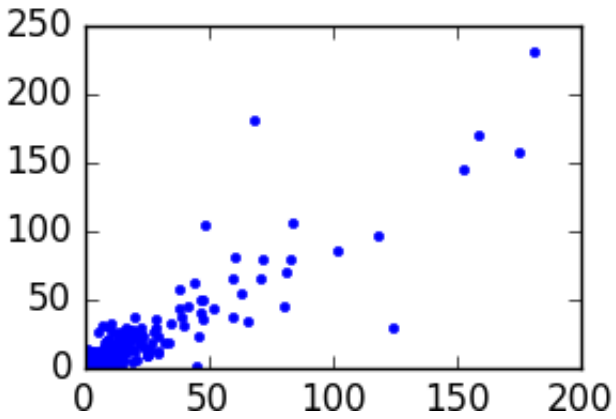
Plot 2: Hourly Views

The second plot we want to create is a scatterplot of views from the first data file (x-coordinate) and the corresponding values from the second data file (y-coordinate). It's fairly reasonable to expect a linear relationship between those values.

To do that, you'll need to get the two series into the same DataFrame. If you used the hint above to read the file, the page name will be the index.

You can then use these indexes: if you copy a Series from one DataFrame to another, elements are identified *by their index*. Since the DataFrames have the page name as their index, you can just put the two Series representing page views from both days into a single DataFrame and view counts for each page will end up alongside each other.

With default settings, I get a scatterplot like this (on different randomly-chosen data):



Because of the distribution of the values, the linear axes don't make much sense. Change this plot to log-scale on both axes using plt.xscale *(https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.xscale)* and plt.yscale *(https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.yscale)* .

Final Output

You can use `plt.show()` to see the figure as the program runs. That's probably easiest for testing, but as a final result, don't `show()`, but **create a PNG file** `wikipedia.png` like this:

```
plt.savefig('wikipedia.png')
```

Use the functions `plt.title`, `plt.xlabel`, and `plt.ylabel` to give some useful labels to the plots.

A sample `wikipedia.png` (with different input data) is included in the ZIP file.

Pup Inflation: Analysing Tweets

This question is heavily inspired by David H. Montgomery's Pup Inflation *(http://dhmontgomery.com/2017/03/dograte/)* post. His analysis is an excellent data science task, and we will ask the same question here: has there been grade inflation on the @dog_rates *(https://twitter.com/dog_rates)* Twitter, which rates the cuteness of users' dog pictures?

I scraped the @dog_rates feed with tweet_dumper.py *(https://gist.github.com/yanofsky/5436496)* . The result it produced is provided in the `dog_rates_tweets.csv` file, so we don't all have to scrape the data. (Yes, there's a gap in the data: data collection is hard.)

Do this analysis in a **Jupyter notebook `dog-rates.ipynb`**. To look for score inflation, we'll first have to make sense of the data. The steps I think are necessary to do this:

- Load the data from the CSV into a DataFrame. (Assume a `dog_rates_tweets.csv` file is in the same folder as the notebook file.)
- Find tweets that contain an "***n***/10" rating (because not all do). Extract the numeric rating. Exclude tweets that don't contain a rating.
- Remove outliers: there are a few obvious ones. Exclude rating values that are too large to make sense. (Maybe larger than 25/10?)
- Make sure the 'created_at' column is a datetime value, not a string. You can either do this by applying a function that parses the string to a date (likely using strptime *(https://docs.python.org/3/library/datetime.html#datetime.datetime.strptime)* to create a datetime object), or by asking Pandas' `read_csv` function to parse dates in that column with a `parse_dates` argument.
- Create a scatter plot of date vs rating, so you can see what the data looks like.

[The question continues, and there are a few hints below. You may want to do this part of the question and make sure things are working before continuing.]

Linear Fitting

One analysis Montgomery didn't do on the data: a best-fit line.

The `scipy.stats.linregress` function can do a linear regression for us, but it works on numbers, not datetime objects. Datetime objects have a `.timestamp()` method that will give us a number (of seconds after some epoch), but we need to get that into our data before using it. If you write a function `to_timestamp` then you can do one of these (if it's a normal Python function, or if it's a NumPy ufunc, respectively):

```
data['timestamp'] = data['created_at'].apply(to_timestamp)
data['timestamp'] = to_timestamp(data['created_at'])
```

You can then use `linregress` to get a slope and intercept for a best fit line.

Produce results like those found in the provided screenshot, `dog-rates-result.png`. **At the end of your notebook** (so the TA knows where to look), show the data itself, the slope and intercept of the best-fit line, and a scatterplot with fit line.

Hints

This Python regular expression *(https://docs.python.org/3/library/re.html)* will look for "***n***/10" strings in the format they seem to occur in the tweets. If this is found by searching in a tweet, then the resulting match object *(https://docs.python.org/3/library/re.html#match-objects)* can be used to get the numeric rating as a string, which can then be converted to a float.

```
r'(\d+(\.\d+)?)/10'
```

I think the easiest way to "exclude" some rows from the DataFrame is to return `None` for rating values that aren't valid ratings, and then use Series.notnull *(http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.notnull.html#pandas.Series.notnull)* to create a boolean index. There are certainly other ways to do the job as well.

To plot the best-fit line, the x values must be datetime objects, not the timestamps. To add the best-fit line, you can plot `data['created_at']` against `data['timestamp']*fit.slope + fit.intercept` to get a fit line (assuming you stored the results of `linregress` in a variable `fit`).

Here are some hints to style the plot as it appears in my screenshot, which seems to look nice enough:

```
plt.xticks(rotation=25)
plt.plot(???, ???, 'b.', alpha=0.5)
plt.plot(???, ???, 'r-', linewidth=3)
```

Questions

Answer these questions in a file `answers.txt`. [Generally, these questions should be answered in a few sentences each.]

1. In the hint above, what is the result of the calculation `data['timestamp']*fit.slope + fit.intercept`? What is the type, and describe the values.
2. In the same hint, why does this produce a fit line on the graph? Why are the `created_at` values and `timestamp` values paired correctly to make points on the plot?

Submitting

Submit your files through CourSys for Exercise 2.

Updated Fri April 28 2023, 09:13 by ggbaker.