CleanSpark Python Exercise: June 2019

This repository ('`clsk-python-exercise-june-2019`') is where we (CleanSpark) host the take-home Python exercise for the Junior Data Scientist/Analyst position. Please carefully read this entire document before continuing. There is a fair amount of scaffolding in this repo to ensure everyone starts on an even playing field. This scaffolding includes packages that you may not be familiar with; however, we outline their primary usage below. Learning to work with new open-source packages is a core skill required for any successful Python developer.

Getting started

Python version

This exercise will require that you have a version of Python 3 installed on your machine. If you do not have Python 3 installed on your computer, you will need to download and install it. Installation of Python 3 will vary depending on your platform and package manager. If you do not know how to install Python 3, please contact Rich Inman for help.

Clone this repo

Navigate to any directory where you would like to keep this repository. Once there, clone this repo into the directory using `git clone` followed by the repositories URL or by downloading a zip file. CleanSpark recommends using git to clone the directory.

Once you have cloned this directory, navigate into the directory by typing `cd clsk-test-june-2019`. You are now ready to set up your development environment.

Setting up your development environment (Pipenv)

CleanSpark uses Pipenv as its standard dependency manager for Python projects. Also, the Python Software Foundation (PSF) officially recommends Pipenv for application development in versions of Python greater than 3.6. Per the PSF documentation on Managing Application Dependencies:

> Pipenv is a dependency manager for Python projects. If you're familiar with Node.js' npm or Ruby's bundler, it is similar in spirit to those tools. While pip alone is often sufficient for personal use, Pipenv is recommended for collaborative projects as it's a higher-level tool that simplifies dependency management for common use cases.

First, make sure you also have `pip` installed on your machine. You can verify its installation by typing the following in a terminal/command prompt:

```
pip --version
```

You should see something like the following:

```
pip 19.1.1 from c:\Users\...
```

Now that we know `pip` is installed, we can use it to install `pipenv`:

```
pip install --user pipenv
```

> Note: The PSF recommends performing a user installation to prevent breaking any system-wide packages. If `pipenv` isn't available in your shell after installation, you'll need to add the user base's binary directory to your `PATH`. See

> Installing to the User Site for more information.

We can verify the installation of `pipenv` by running the following command:

```
pipenv --version
```

You should see something like the following:

```
pipenv, version 2018.11.26
```

Finally, now that `pipenv` is installed we can use it set up a development environment using the Pipfile provided in the repository. Run the following command, which will install all the required packages as well as the development packages such as `pytest` which we will use to test our code.

```
pipenv install --dev
```

The last step before beginning the coding challenge is to activate your new development environment by using the following command:

```
pipenv shell
```

> NOTE: Make sure to run the `pipenv shell` command every time you start working on this exercise. If you do not, there is no guarantee that you have all of the correct packages installed or that your code will run when it is tested at CleanSpark.

Running tests

It is commonly accepted that code without tests is bad code. For that reason, all production code is thoroughly tested at CleanSpark. Python code is verified using the `pytest` package. The good news is, we had already installed pytest and written tests for you (`pytest` was installed when we ran the `pipenv install --dev` command).

Next, we need to make sure you know how to run the tests we have written. Run the following command inside of your virtual environment:

```
python -m pytest
```

You should see that all of the tests were skipped. The last step before beginning to code is to make sure these tests are not skipped. Inside the `clsk\tests\` directory, there is a file called `billing_unit_test.py`. Open this file and completely delete all lines that say:

```
@pytest.mark.skip(reason='Waiting for candidate to remove.')
```

Save the file and rerun your tests. You should see that all the tests are now failing. It is your job to get these tests to pass by writing code. Keep reading for a description of the code you will need to write.

> NOTE: After deleting the lines mentioned above, you should not alter the `billing_unit_test.py` any more. These tests were written for you to pass and to expose you to how to write tests in Python.

Problem Description

Energy and Power

At CleanSpark we analyze electricity bills every day. In general, an electricity bill is composed of two primary parts: energy charges and maximum-power (demand) charges. In order to understand these two portions of the bill, we must first understand the difference between energy and power.

One helpful analogy is between electricity comsumption and driving your car. In order to understand this better, let's use a concrete example. Imagine you want to drive from San Diego to San Franciso this weekend. After checking Google Maps, you know that you need to travel roughly 500 miles.

Now, let's assume someone asked you how long it will take you to get to San Francisco. What would you say? ... Hopefuly, you thought to yourself, "I don't have enough information. I need to know how fast I am driving in order to know how long it will take."

This is because time and distance require a speed (or rate) to convert between the two. You may recall the familiar definition of a rate:

$$ {\rm speed} = \frac{\rm distance}{\rm time} $$

Now, assume I also told you that you will be traveling at an average speed of 60 mph over your 500 mile journey to San Francisco and then asked you how long it would take. Using our definition of speed (rate) we can easily calculate the time required to get to San Francisco:

$$ {\rm time} = \frac{\rm distance}{\rm speed} = \frac{\rm 500 \, miles}{\rm 60 \, mph} = \rm{8.\bar{3} \, h} $$

Extending the example above to electricity we can say that energy is to distance what power is to speed. However the units can be a bit tricky. Energy is measured in kWh and power is measured in kW. In order to provide some more context, let's assume a building consumed 50 kWh over a 30 minute period and I asked you what the averave power was. Well extending our formula from above:

$$ {\rm power} = \frac{\rm energy}{\rm time} = \frac{\rm 50 \, kWh}{\rm 0.5 \, h} = {\rm 100 \, kW} $$

Now that you understand energy and power we can discuss how energy bills are typically calculated.

Electricity bills

As mentioned before, energy bills have two primary components: energy and max-power (demand). These two portions are calculated in completely different ways. We will first discuss energy and then max-power (demand).

Energy rates have units of $/kWh and are calculated at every 15 minute interval. For example, if the energy rate is $0.01/kWh and the building consumed 100 kWh of energy over that 15 minute interval, they are charged $1.00. This occurs at every 15 minute interval in the month and all of the charges get summed up. This sum of energy charges is the energy portion of your bill.

> Note: This operation is very similar to a dot product.

Max-power (demand) rates have units of $/kW and are calculated only once per month. For example, for an entire month of data let's assume the maximum average power (demand) in a single interval was 100 kW and the demand rate is $20/kW. The customer would be they are charged $2000 once for that month.

Problem

You are given a month electricity consumption data (kWh) in 15 minute intervals for a comercial facility located in San Diego, CA. This data can be read into a pandas DataFrame using the pre-coded `_get_data` function in the `billing.py` module. Each interval is labeled with the time it began and all intervals are 15 minutes long. Your task is to add logic to the

two stubbed functions in the `billing.py` module. The first method is called `energy_charge` and the second is called `demand_charge`.

An example pricing scheme (electricity tariff) is outlined below. Your functions will need to use the electricity consumpation data and the pricing scheme to calculate the energy and demand charges. Please feel free to import any other third packages you think you will need, but make sure to install them using `pipenv`. In additions, feel free to write as many extra helper functions as you need and include them in the `billing.py` file as well.

Energy rates

Weekends: $0.05/kWh

Weekdays (12:00 am - 4:00 pm): $0.20/kWh

Weekdays (4:00 pm - 9:00 pm): $0.30/kWh

Weekdays (9:00 pm - 12:00 am): $0.10/kWh

Demand rates

Monthly max: $20/kW

Correct Solution

You will know if you have implemented the logic correctly when you run your tests and both of them pass.

Submission

Once you have finished working on your exercise. Please compress the entire repository into a .zip file and email to rinman@cleanspark.com. Good luck and happy coding!