Mini Project 1

Weather Forecast

`2 - DAL` `python`

Prerequisite

- Install following packages. If you're using Anaconda Python, some of them may have already been installed.
  - requests
    - Python HTTP library.
    - We use it for making RESTful API calls
  - pandas
    - Python data analysis library
    - We use it for tabular data handling and export to .csv
  - matplotlib
    - Python plotting library
- PyCharm is not a must, but is recommended. We will be using PyCharm as the default IDE for the rest of this tutorial.

> You may refer to this post for how to install packages on Windows / mac OS and how to import sample json file into your project.

Inside your project directory, create a empty file named `weather_forecast.py`. We'll put most of our codes in this file.

Get familiar with JSON

JSON (JavaScript Object Notation) is a kind of file that is heavily used in net work communications. You may think of it as the universal language for data exchange in the computer world.

The format of JSON file is similar to `list` and `dict` types in Python which makes it a lot easier for us to understand the meaning of the file. The very first thing we need to do is to analyze the content of JSON and extract necessary data from it. This procedure is also called *parsing*.

Load JSON from file

Python has an in-built package called `json` that can help us handle JSON easily. But before we use any functions from `json` package we'll have to import the package into our project. This can be done easily by calling `import json` at the beginning of `weather_forecast.py`.

> According to PEP-8, we should always put all `import` statements at the beginning of source file.

Now, we may use the powerful `load()` method provided by `json` package to read the content of a specific JSON file and put it into memory as a `dict` that Python recognize.

```
import json

def load_json_sample(path: str) -> dict:
    with open(path, encoding='utf-8') as json_file:
        return json.load(json_file)
```

In python 3, we may specify the type of arguments in a function definition as well as the return type of it. In the code above, we defined a function that takes one argument of type `str` and return a `dict`.

The first line of `load_json_sample` opens a local file at `path`. Since the `sample.json` is encoded in `utf-8`, we need to set the `encoding` argument as `utf-8`.

Parsing JSON

The next step is to parse JSON object and return necessary data for out task.

```
def daily_data_of_attributes(json_dict: dict, attributes: list) ->
dict:

    daily_attributes = {}
    for attr in attributes:
        daily_attributes[attr] = []

    daily_data = json_dict["daily"]["data"]
    try:
        for dict_data in daily_data:
            for attr in attributes:
                daily_attributes[attr].append(dict_data[attr])
    except KeyError:
        print("Key Not Found")
        return {}
    return daily_attributes
```

We declared a new function that takes a JSON object (`json_dict`) and a list of attributes that we're interested in (`attributes`). It will return a `dict` using attribute in `attributes` as key with corresponding data extracted from `json_dict`.

The format of returned object should be like:

```
{
    'temperatureMin': [0.0, 1.0, 2.0, ...],
    'temperatureMax': [3.0, 4.0, 5.0, ...]
    'humidity': [3.0, 4.0, 5.0, ...]
}
```

A better way to do that is using the `defaultdict`:

```
def daily_data_of_attributes_better(json_dict, attributes):
    daily_data = json_dict['daily']['data']

    from collections import defaultdict
    daily_attributes = defaultdict(list)

    try:
        for dict_data in daily_data:
            for attr in attributes:
                daily_attributes[attr].append(dict_data[attr])
    except KeyError:
        print("Key Not Found")
        return {}

    return daily_attributes
```

> In production, you may not need to do the exception thing, since the JSON format will always be the same. But it is always a good habit to avoid crushing.

Test Parsing

It's time to test whether the loading & parsing works well. We defined a `main` function to represent the flow of our program.

```
def main(remote=False):
    if remote:
        pass
    else:
        json_obj = load_json_sample('sample.json')

    attributes = ['temperatureMin', 'temperatureMax', 'humidity']
    daily_data = daily_data_of_attributes_better(json_obj, attribut
es)
```

It accepts a `remote` argument and by default it is set to `False`. This is how to control whether to load data from a local file (`remote = False`) or from remote server by API call (`remote = True`).

In the flow, we first load a JSON object from a local file, then we define a list of attributes and extract these attributes from the JSON object.

Before we hit the running button, we'll need to create an entry point for our program. So that Python interpreter knows where to start, i.e. which is the first line to interpret.

To do this, we'll add a special `if` statement to mark the entry point.

```
if __name__ == '__main__':
    main()
```

In order to print out the value of `daily_data`, we need to add a new utiliy function that can print it beautifully:

```
def pretty_print_dict_of_list(d: dict):
    indent = 4
    print("{")
    for k, l in d.items():
        print(indent * " " + k + ": ", end="")
        print(l)
    print("}")
```

We can pass `daily_data` to it and the output should be the same as below:

```
{
    temperatureMax: [64.57, 64.33, 66.61, 66.52, 64.52, 67.63, 73.3
7, 68.41]
    temperatureMin: [61.17, 59.39, 61.44, 63.46, 62.5, 63.15, 65.4
1, 65.62]
    humidity: [0.89, 0.89, 0.71, 0.67, 0.69, 0.69, 0.69, 0.7]
}
```

> You may also use `pprint`. But sometimes it will not print out in the same format as we expected.

DataFrame

So far we've extracted what we need from original data. The next step is to play with it, or to manipulate it. Here we introduced a new powerful package called Pandas. It is usually used to manage tabular data and do further analysis. The basic data type in Pandas is DataFrame. You may think of it as an Excel sheet. Later we will use it to plot the line chart.

It's easy to build a DataFrame object from dictionaries like `daily_data`. We'll just need to pass it to the constructor of DataFrame.

```python
import pandas as pd # Put this at the beginning
df = pd.DataFrame(daily_data)
```

Here we import **pandas** package and rename it to `pd` so that we may use it to represent `pandas` later in our code, to save some typings. To use any class / function of it, we will have to type `pd.` first. That's the standard format of using 3rd party packages.

> We may call `df.to_csv(file_path)` to export the DataFrame into a .csv file at given file path.

Plot

The last thing we'll do here is to plot 2 line charts, one with highest / lowest temperatures and another with the humidity levels in the next 7 days.

Add the following code to `main` function:

```python
df_temperature = df[["temperatureMin", "temperatureMax"]]
df_humidity = df[["humidity"]]

# Plot data
plt.style.use('ggplot')
_, axes = plt.subplots(nrows=2)

df_temperature.plot(ax=axes[0])
df_humidity.plot(ax=axes[1])

plt.show()
```

First we separate `df` into two data frames. We did this by passing a list of column names into `[]`. Then we did some configurations to the plotting style:

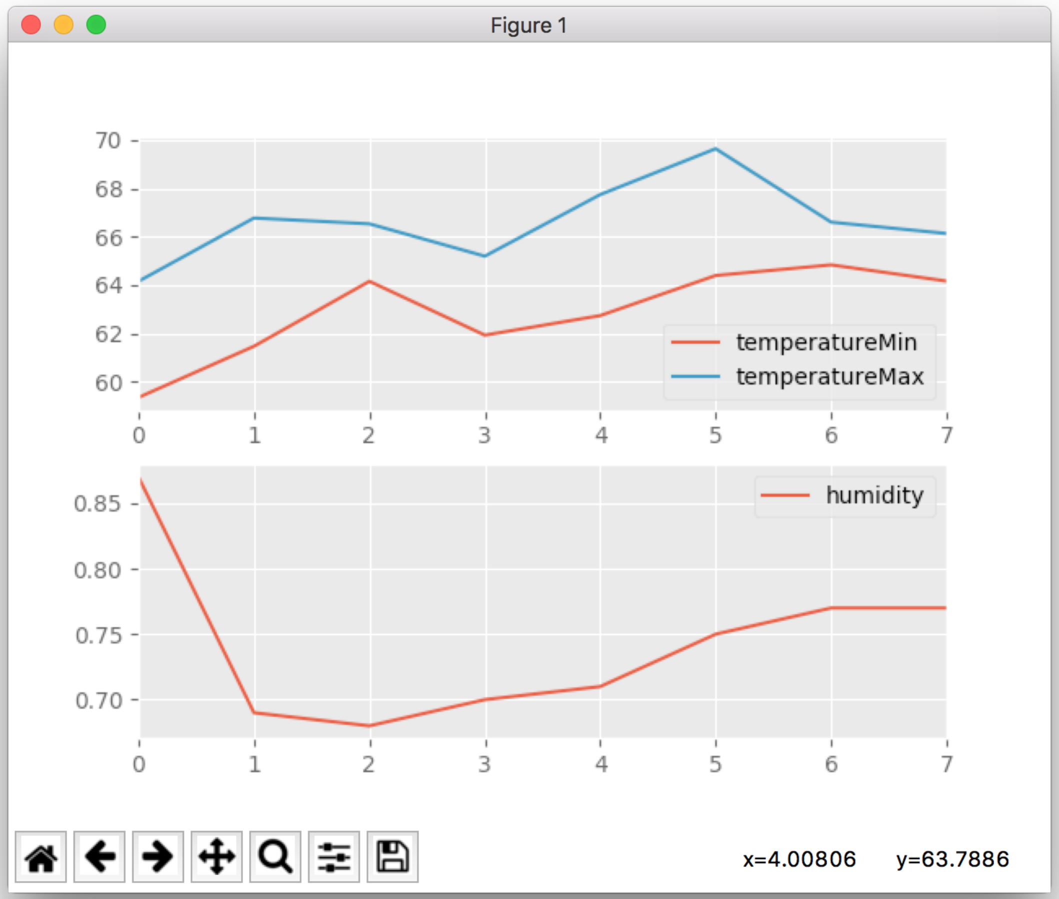
1. Use ggplot style
2. Divide the whole chart into two rows by calling `subplots` method. The syntax here is similar to MATLAB.

Then we did the actual plotting by calling `plot` method on our two `DataFrame` objects. By setting `ax` argument we can specify which chart appears at the top and which at the bottom.

The last thing we did here is to show the plot by calling `plt.show()`.

> If you cannot see the plot, minimize your PyCharm window since the plot window can be inactive.

You should see something like this:



Remote Request

So far we've done most of things for this project. The only thing missing is that we're not getting latest weather data from providers. To make the project really meaningful, let's write one last function to fetch data from DarkSky.

> Before you move on, you should register an account on DarkSky and get your API key ready.

Add the following function to your `weather_forecast.py` :

```
def request_data():
    lat = 37.7749
    long = -122.4194
    api_key = "YOUR_API_KEY"

    url = "https://api.darksky.net/forecast/%s/%s,%s" % (api_key, l
at, long)
    response = requests.get(url)
    return json.loads(response.text)
```

Here is what we did in the function:

1. Set the latitude and longitude of the place we are interested in. Here I'm using San Francisco as an example.
2. Replace `YOUR_API_KEY` with your own API key.
3. Build the API URL by providing base url, `api_key`, `lat` & `long`
4. Calling `get` method in `requests` package and store the response in `response`.
5. Like what we did in `load_json_sample`, we return a JSON object (dict) by passing JSON text to `json.loads()` function.

Change the first `if` statement in your `main` function into:

```
if remote:
    json_obj = request_data()
else:
    json_obj = load_json_sample('sample.json')
```

Change the entry point into the following code so that we'll get data remotely instead of loading from local JSON file.

```
if __name__ == '__main__':
    main(remote=True)
```

Build and run your project, you should be able to see the real time weather data in the next 7 days.

What's Next

There are a lot of improvements you can make to this project. For example, I would like my program read the weather report for me so that I don't have to stare at the screen. Fortunately, there's a system command in mac OS that will do text-to-speech, exactly what I need here.

> Windows does not provide similar command so the following tutorial is mac only.

Add a new function into your `weather_forecast.py`:

```
def say(text: str):
    subprocess.call('say ' + text, shell=True)
```

Then add the following code after where we get `daily_data`:

```
highest_temp = max(daily_data["temperatureMax"])
lowest_temp = min(daily_data["temperatureMin"])
report = "The highest temperature in the coming week will be " \
        + str(highest_temp) + " degrees, with the lowest of " + st
r(lowest_temp) + " degrees."
say(report)
```

Don’t forget to import `subprocess` at the beginning.

That’s it! Build and run your program, it will give you a brief weather report about the highest temperature and lowest temperature in the next 7 days. Pretty like what we get from Siri or Echo.

Play with the data a little bit and use your imagination to make your project better. This can be a good one on your resume.