Data Dive 2: Loading and Summarizing Data

Part 2: Scraping the Web for Unique Data

Web scraping (https://en.wikipedia.org/wiki/Web_scraping) is the process of extracting data from html code on the internet.

Resources on web scraping:

- Digital Ocean Tutorial (https://www.digitalocean.com/community/tutorials/how-to-scrape-web-pages-with-beautiful-soup-and-python-3) (requests, Beautiful Soup)
- DataCamp Tutorial (https://www.datacamp.com/community/tutorials/web-scraping-using-python) (urllib, Beautiful Soup)
- Hitchhiker's Guide to Python (https://docs.python-guide.org/scenarios/scrape/) (requests, lxml)

Important Note: This is for demonstration purposes only, and only harvests content from individual pages. When building a scraper to harvest large amounts of data from multiple pages, be mindful of legal (https://www.fastcompany.com/40456140/bots-are-scraping-your-public-data-for-cash-amid-murky-laws-and-ethics-linkedin-hiq) and ethical (https://towardsdatascience.com/ethics-in-web-scraping-b96b18136f01) issues in web scraping.

Today's Exercise

Say we want to learn more about where Google's offices are located. Helpfully, the provide a list of all of their campuses globally at google.com/about/locations (https://www.google.com/about/locations). However, copying this list by hand to do data analysis on would be frustrating and time-consuming. Let's take a look at how web scraping can make this process easier.

First, let's import all of the packages we'll need for today's exercise. There are a wide variety of packages that can be helpful, but today we'll be using *requests* and *Beautiful Soup* to pull the contents of these websites.

In [ ]:
```
import re
import pandas as pd
from bs4 import BeautifulSoup as soup
import requests
```

First, we use the *requests* package to get the content of the google site we'd like to extract office location information from:

In [ ]:
```
url = 'https://www.google.com/about/locations/'
site_source = requests.get(url)
```

This is going to give us an enormous amount of content - everything we would get if we looked directly at the source code in the browser.

In [ ]:
```
print(site_source.text)
```

In [ ]:
```

```

We could parse this ourselves, but fortunately scraping packages make this much easier

We'll use Beautiful Soup's built in functionality to extract info on the individual offices.

First, we parse the full site content.

In [ ]:
```
site_content = soup(site_source.content, "html.parser")

type(site_content)
```

Next, we pick out the office elements

In [ ]:
```
offices = site_content.select(".office-info")

len(offices)
```

In [ ]:
```

```

In [ ]:
```

```

Now that we've isolated the office elements, let's extract the location name and address for each.

In [ ]:
```
for o in offices:
    office = o.select(".office-name")[0].string.strip()
    address = o.select(".office-address")[0].string.strip()

    print(office)
    print(address)
    print()
```

If we look carefully at our extracted elements, we'll see we have some issues:

1. All elements appear twice.
2. The zip codes - which we're interested in - are part of broader strings.

These are trivial to handle, we'll just need to pass over the data carefully to handle both.

In [ ]:
```
us_offices = []
for o in offices:
    office = o.select(".office-name")[0].string.strip()
    address = o.select(".office-address")[0].string.strip()

    is_US = re.search(r'(United States)', address)

    if is_US:

        print(office)
        zip_code = re.search(r'(\d{5})', address)
        if zip_code:
            print(zip_code.group())
            if [office, zip_code.group()] not in us_offices:
                us_offices.append([office, zip_code.group()])
        print()
```

Now that we've extracted a list of offices and zip codes, we can load them into a data frame.

In [ ]:
```
office_df = pd.DataFrame(us_offices, columns=['Office', 'Zip Code'])

office_df.head(5)
```

In [ ]:
```

```

Exercise: In What Countries Does Google Maintain Offices?

In [ ]:
```
# todo: compile a dataframe the cities and countries in which Google mai
ntains international offices
```

Adding County Information

The Department of Housing and Urban Development makes a *crosswalk* of zip codes to counties available here (https://www.huduser.gov/portal/datasets/usps_crosswalk.html). We can load these into pandas and clean them up to find the county for each office.

In [ ]:
```
zip_df = pd.read_csv('https://grantmlong.com/data/ZIP_COUNTY_122016.csv'
)

zip_df.head(5)
```

A good rule of thumb: if two numbers cannot be added together to produce a logical result, they should be stored as strings. '

We can recast the zip and county ids as strings - with leading zeros - to make this dataframe easier to handle. To do this we can use the .astype() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.astype.html) and .zfill() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.str.zfill.html) methods.

In [ ]:
```
zip_df['Zip Code'] = zip_df['ZIP'].astype(str).str.zfill(5)
zip_df['County Number'] = zip_df['COUNTY'].astype(str).str.zfill(5)

zip_df.sort_values(by='COUNTY').head(5)
```

Of course we don't need all of these columns, but we do need to attach the ***County Number*** column to our Google office data in order to learn more about the data. The .merge() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) method allows us to do this easily in one line.

In [ ]:
```
office_df = office_df.merge(zip_df[['Zip Code', 'County Number']],
                            how='left')

office_df.sort_values(by='Office')
office_df.shape
```

In [ ]:
```
office_df.sort_values(by='County Number')
```

Merge Office Data with Census Data

First, we'll need to load the data extract we produced in the first data dive. We'll also need to make sure that the *County Number* - the variable we need to join our data on - is appropriately formatted as a string.

In [ ]:
```
census_df = pd.read_csv('https://grantmlong.com/data/census_counties_backup.csv')
census_df['County Number'] = census_df['County Number'].astype(str).str.zfill(5)

census_df.head(5)
```

Next, we'll use the .merge() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) method to attach the two data sets.

In [ ]:
```
full_df = census_df.merge(office_df, how='left')
full_df.loc[full_df['Office'].notnull(), ].head(5)
```

With our full data set, we can now begin to look at some interesting numbers, like the median income in counties where google has an office, and where they don't.

In [ ]:
```
print(full_df['Median Household Income'].mean())
print(full_df.loc[full_df['Office'].notnull(), 'Median Household Income'
].mean())
print(full_df.loc[full_df['Office'].isnull(), 'Median Household Income']
.mean())
```

In [ ]:
```
(full_df
 .sort_values(by='Median Household Income',
              ascending=False)
 .head(20))
```

Exercise: What Other Interesting Findings Can We Identify?

In [ ]:

In [ ]:

In [ ]: