PROJECT REPORT

ns03444

IT4530

Contents

1 Product Description

This application is a content-aggregator built primarily with Python. It provides users specifically interested in the latest news for the Information Technology sector of the stock market to stay updated by aggregating the most recent new articles from 3 popular websites and displaying them on one webpage. Users will also be able analyze the top 25 stocks/stock prices in the technology industry. It will encompass different types of data visualizations as well as an in-depth financial analysis on 3 largest wireless network providers.

2 Web Scraping

2.1 Web Scraping: Stock Performance Data

The first portion of my project consisted of using Python to scrape each stock's ticker symbol, name, and current price from a table of the top 25 technology stocks listed on: https://finance.yahoo.com/sector/ms_technology?.tsrc=fin-srch.

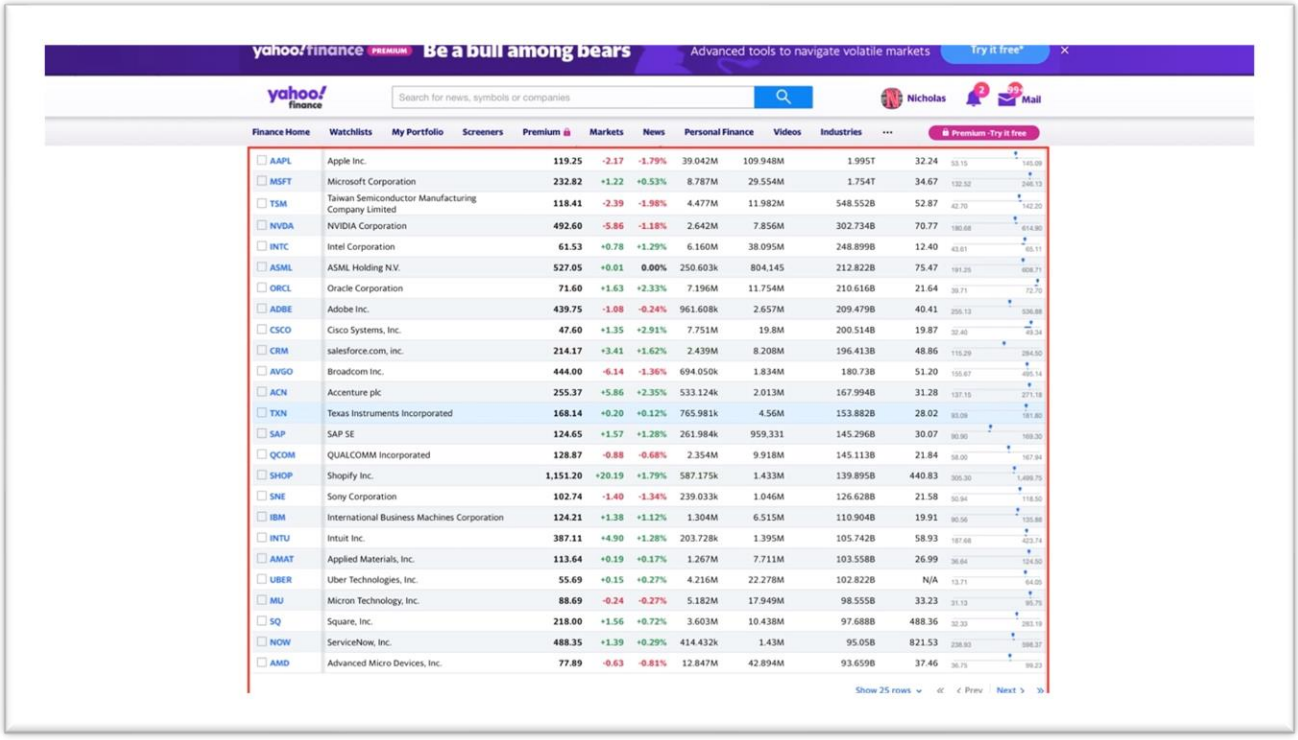


Figure1.1-1-Yahoo Finance Stock Data Table

2.1.1 Connecting to the Data

To begin the process, I had import and explore the basic nature and syntax of the "BeautifulSoup" and "requests" Python libraries to the "WebScraper1_tickers" script. The "requests" library is one of the most common libraries used for making HTTP requests in Python. The "requests" library and the "get()" method, combined, are used to send an HTTP request and receive a "response object" in return. The object contains the source code of the website and other descriptive data about the site (encoding, status, etc.). Then, by using the "content" method, the source code can be assigned to a variable. In the code shown below, the content variable now contains the source code of the website. However, since the content variable is of the data type "bytes", Python is still unable to do very much with the scrambled, unorganized source code. This is where the "Beautiful Soup" library comes into play. By creating a "BeautifulSoup object" and an "HTML parser", Python can process a copy of the source code data and return the HTML code using specific elements, tags, containers, classes, etc. in a very efficient way.

```
WebScraper1_tickers.py
1  import requests
2  from bs4 import BeautifulSoup
3  import csv
4
5  url = requests.get("https://finance.yahoo.com/sector/ms_technology?.tsrc=fin-srch")
6  c = url.content
7  print(c)
8  soup = BeautifulSoup(c,"html.parser")
```

Figure1.1.1-1-Creating Response Object

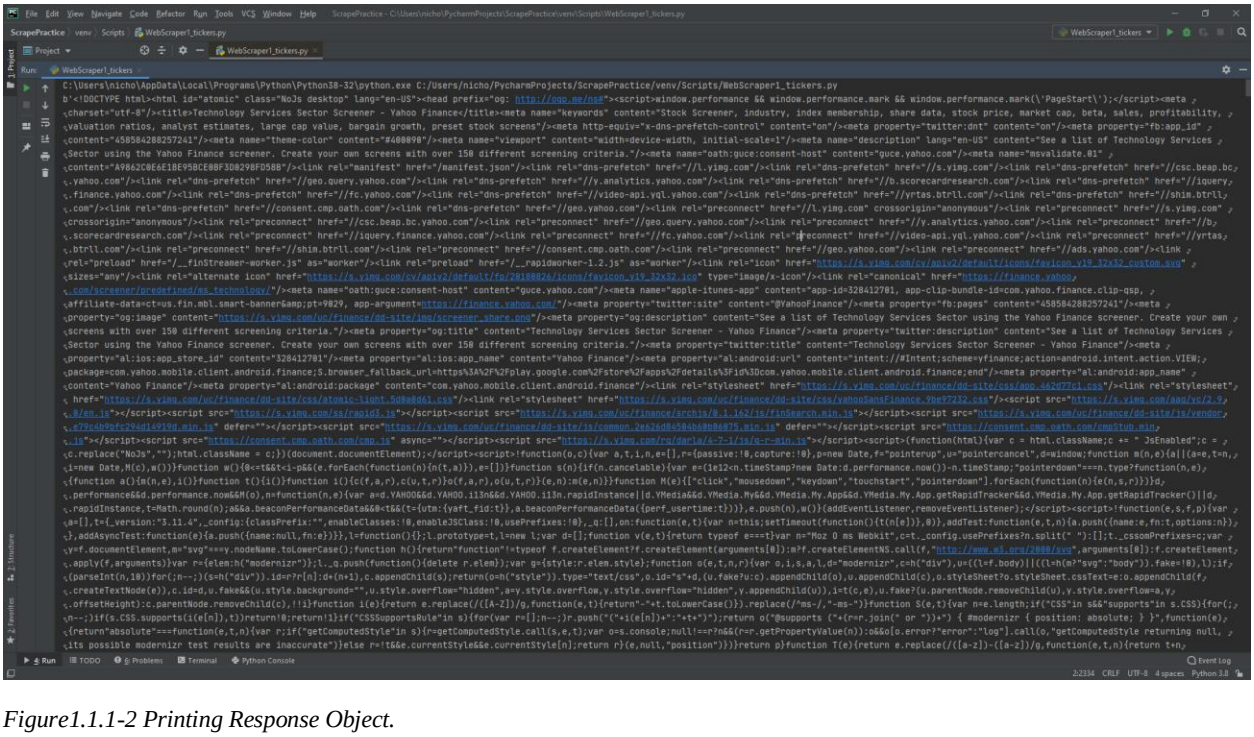


Figure1.1.1-2 Printing Response Object.

2.1.2 Scraping Ticker Symbols

By visiting the website through Google Chrome and using the “inspection” developer tool, I examined the overall structure of the website’s HTML code. Furthermore, since no obvious pattern was recognized, I used the “inspect element” developer tool to take a closer look at where the first ticker symbol was located (“APPL”). I discovered that the string for this ticker was stored in an <a> tag, but u discovered that the next ticker symbol in the list did not share the same attribute “id". Therefore, I knew this was not the element I wanted. So, I found the parent element was a tag. However, due to the odd parent-child relationship, there were more than one tags with the same attribute “id”, so I had to analyze each one to determine which one contained the data I needed. After printing all the items in each of the tags that shared this id, I found that the first contained the correct data (*figure 1.1.6* - hence the [0] on line 12). By using the find_all method for <a> tags with a certain “class” name, I was able to extract the HTML code that contained the Unicode text for each ticker symbol into a list. However, the data was still in HTML syntax while I only needed the plain text. So, I created a new list and used a for loop to iterate through the old list and used the “text” method to solely extract the ticker symbol text and append it to the new list.

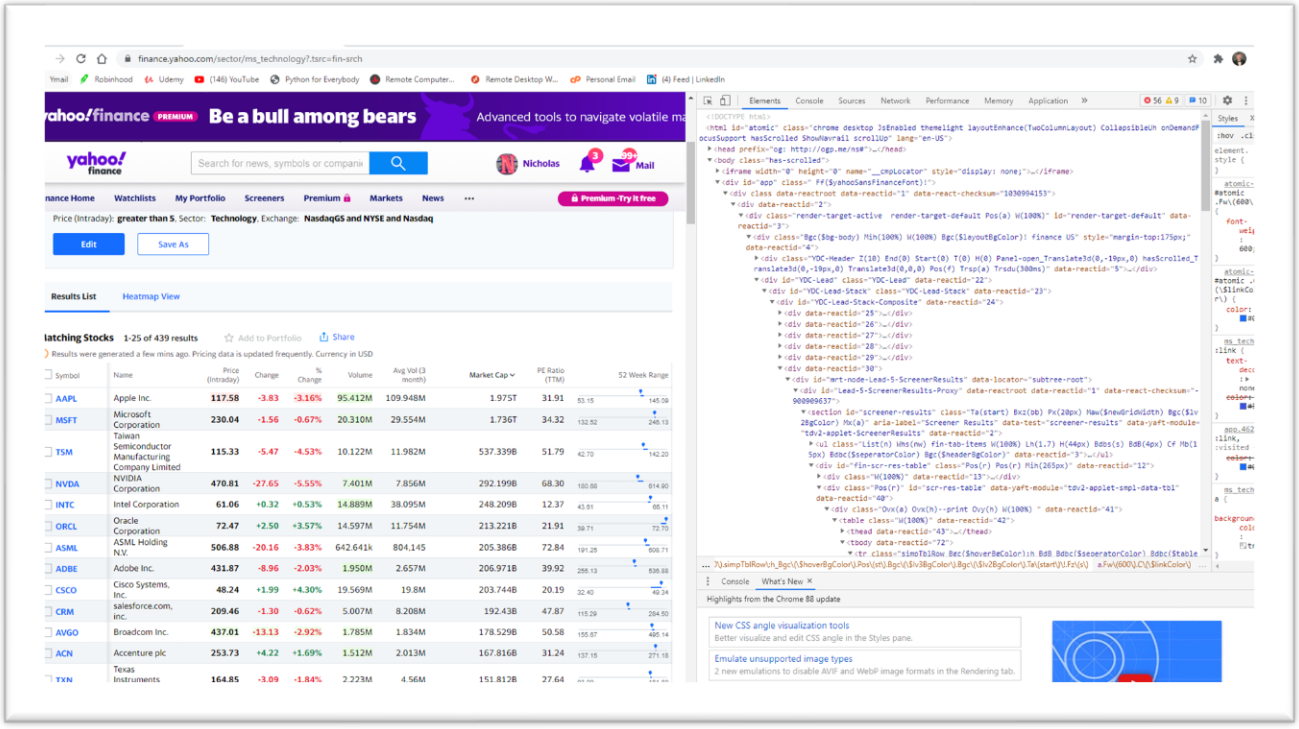


Figure1.1.2-1 Using Google Chrome's Inspection Tool

```
▶<label class="Ta(c) Pos(r) Va(tb) Pend(5px) D(n)--print" data-reactid="75">…</label>
  <a href="/quote/AAPL?p=AAPL" title="Apple Inc." class="Fw(600) C($linkColor)" data-reactid="79">AAPL</a> == $0
  <div class="W(3px) Pos(a) Start(100%) T(0) H(100%) Bg($pfColumnFakeShadowGradient) Pe(n) Pend(5px)" data-reactid="80"></div>
</td>
```

Figure1.1.2-2 Inspecting the 'APPL' ticker Element.

```
table = soup.find_all("div", {"id":"fin-scr-res-table"})
print(table[0])
```

```
ipt>\n<script>window.webpackPublicPath=\'https://s.yimg.com/uc/finance/dd-site/js/\';</script></bo
esultsAdd to PortfolioShareResults were generated a few mins ago. Pricing data is updated frequentl
geAAPLApple Inc.121.42+1.29+1.07%153.767M109.567M2.038T32.93MSFTMicrosoft Corporation231.60+4.87+2
997B54.17NVDANVIDIA Corporation498.46+3.65+0.74%13.571M7.819M309.045B72.24INTCIntel Corporation60.7
c.440.83+1.77+0.40%4.615M2.67M211.025B40.70ORCLOracle Corporation69.97+4.36+6.65%34.211M11.806M205
```

Figure1.1.2-3 Finding the correct div item.

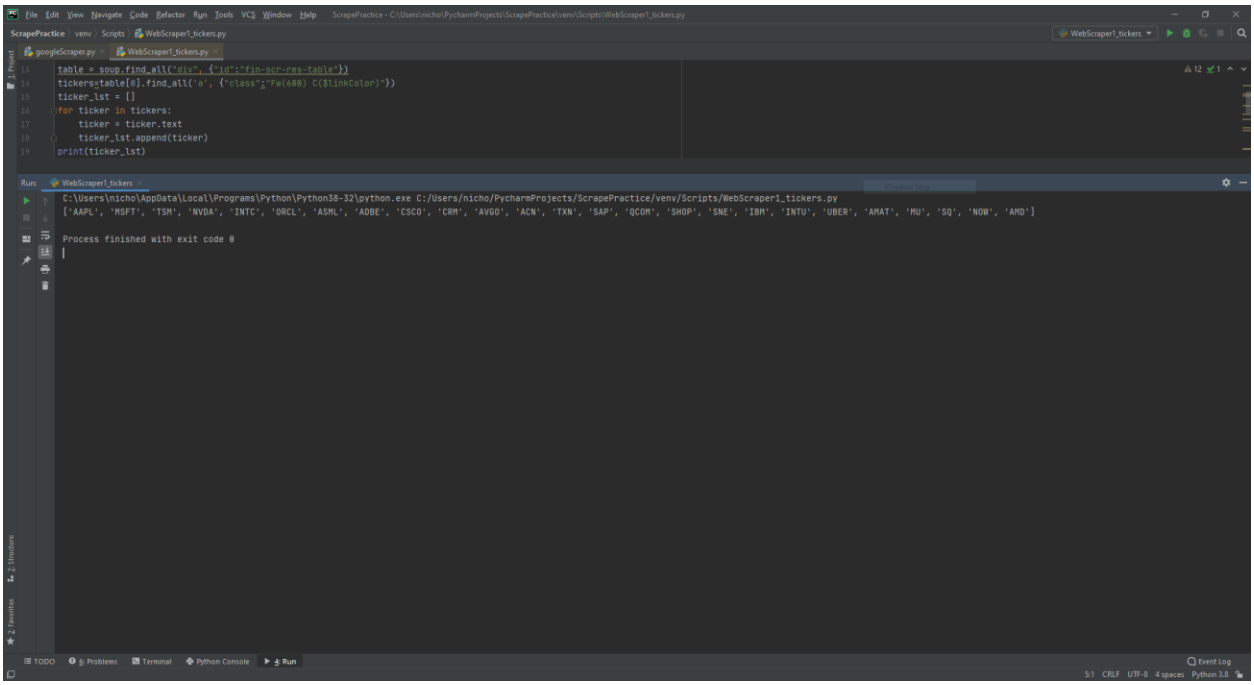


Figure 2.1.2-4 Creating & Printing the Ticker List

2.1.3 Scraping the Names and Prices

Since the data for the ticker symbols, names, and prices were all located in the first list of lists in the "table" variable I previously created all I had to do was find the individual elements that contained the Unicode text for the names and prices, as I did for the ticker symbols. Then, repeat the same process of looping through the list of html code, extracting the text and appending it to a new list. However, one exception came in the process of extracting the prices. While I did not have trouble extracting the string value for the price, I did have to convert the data type to float() during this iteration so the list would contain numeric values.

```
▼<td colspan class="Va(m) Ta(start) Px(10px) Fz(s)" aria-label="Name" data-
reactid="81"> == $0
    <!-- react-text: 82 -->
    "Apple Inc."
    <!-- /react-text -->
  </td>
```

Figure1.1.3-1 Inspecting the 'Apple Inc.' element.

```
▼<td colspan class="Va(m) Ta(end) Pstart(20px) Fw(600) Fz(s)" aria-label="Price
(Intraday)" data-reactid="83">
    <span class="Trsdu(0.3s) " data-reactid="84">121.42</span> == $0
  </td>
```

Figure1.1.3-2 Inspecting Apple's 'Intraday Price' element.

```
prices = table[0].find_all("td", {"aria-label":"Price (Intraday)"})
price_lst = []
for price in prices:
    price = price.text
    price = float(price.replace(',',''))
    price_lst.append(price)
```

Figure1.1.3-3 Converting the price data type.

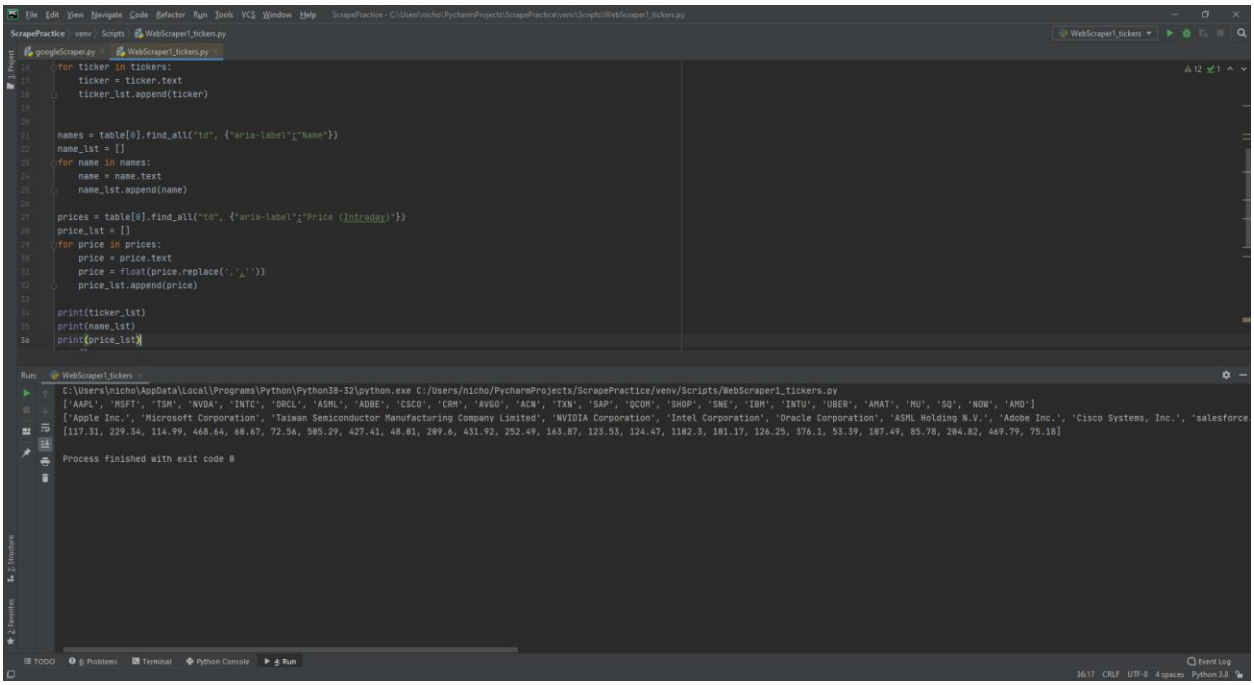


Figure1.1.3-4 All values stored into lists, respectively.

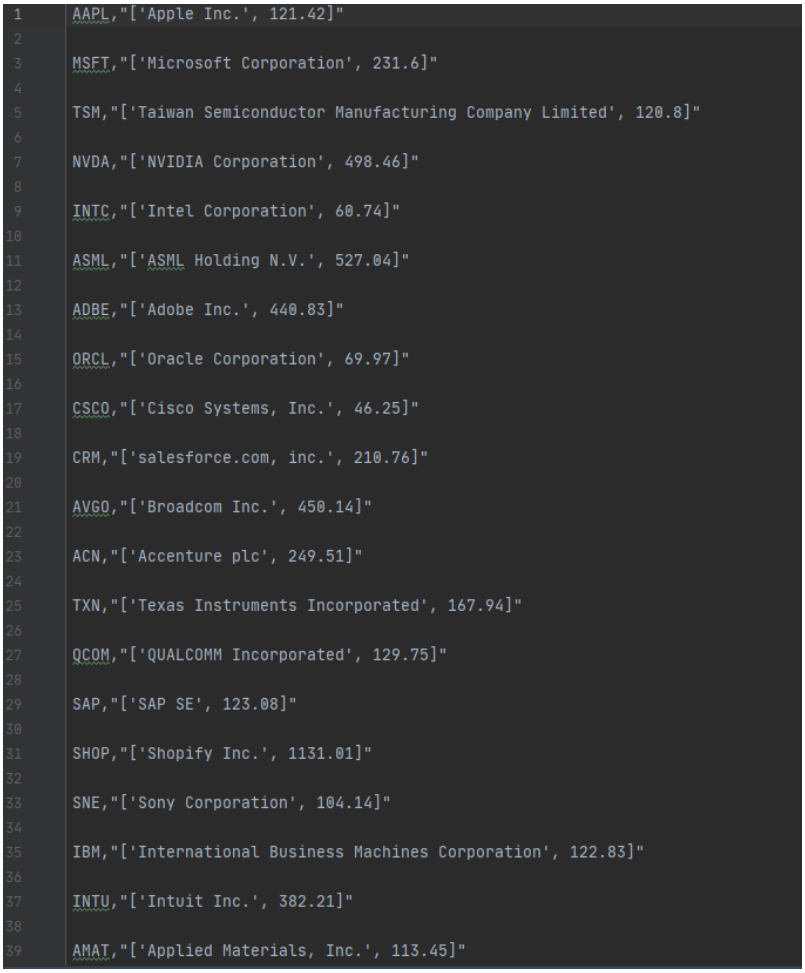
2.1.4 Storing Stock Performance Data

After storing all the data into python lists, I needed to find a way to save the data .csv file to later be imported into a database or elsewhere, if needed. Therefore, I created a dictionary and stored the ticker symbol as the dictionary keys and appended the name and price to a new list and stored the new list in the dictionary value. After the data was properly stored in the dictionary, I used the csv library to write the key, value pairs to .csv file.

```
dct={}
for i in range(len(ticker_lst)):
    lst=[]
    lst.append(name_lst[i])
    lst.append(price_lst[i])
    dct[ticker_lst[i]] = lst

w = csv.writer(open("stockData.csv", "w"))
for key, val in dct.items():
    w.writerow([key, val])
    write = csv.writer(f)
    write = csv.writerow(ticker_lst)
    write = csv.writerow(name_lst)
    write = csv.writerow(price_lst)
```

Figure1.1.4-1 Storing lists in dictionary and writing to csv file.



```
AAPL,"['Apple Inc.', 121.42]"

MSFT,"['Microsoft Corporation', 231.6]"

TSM,"['Taiwan Semiconductor Manufacturing Company Limited', 120.8]"

NVDA,"['NVIDIA Corporation', 498.46]"

INTC,"['Intel Corporation', 60.74]"

ASML,"['ASML Holding N.V.', 527.04]"

ADBE,"['Adobe Inc.', 440.83]"

ORCL,"['Oracle Corporation', 69.97]"

CSCO,"['Cisco Systems, Inc.', 46.25]"

CRM,"['salesforce.com, inc.', 210.76]"

AVGO,"['Broadcom Inc.', 450.14]"

ACN,"['Accenture plc', 249.51]"

TXN,"['Texas Instruments Incorporated', 167.94]"

QCOM,"['QUALCOMM Incorporated', 129.75]"

SAP,"['SAP SE', 123.08]"

SHOP,"['Shopify Inc.', 1131.01]"

SNE,"['Sony Corporation', 104.14]"

IBM,"['International Business Machines Corporation', 122.83]"

INTU,"['Intuit Inc.', 382.21]"

AMAT,"['Applied Materials, Inc.', 113.45]"
```

Figure1.1.4-2 StockData.csv view.

2.2 Web Scraping: Bloomberg News Article Data

The next process of my project dealt with scraping the 3 most recent news articles from 3 different websites. The first website I chose to scrape were the three articles stored in the “Latest Sector News” content block of https://www.bloomberg.com/markets/sectors/information-technology. Furthermore, I needed to scrape and store the title, publication date, link to the article, and description of each article. After gaining some experience from scraping the stock market data, I learned a more efficient way for looping through the HTML elements and parsing the data which I will explain.

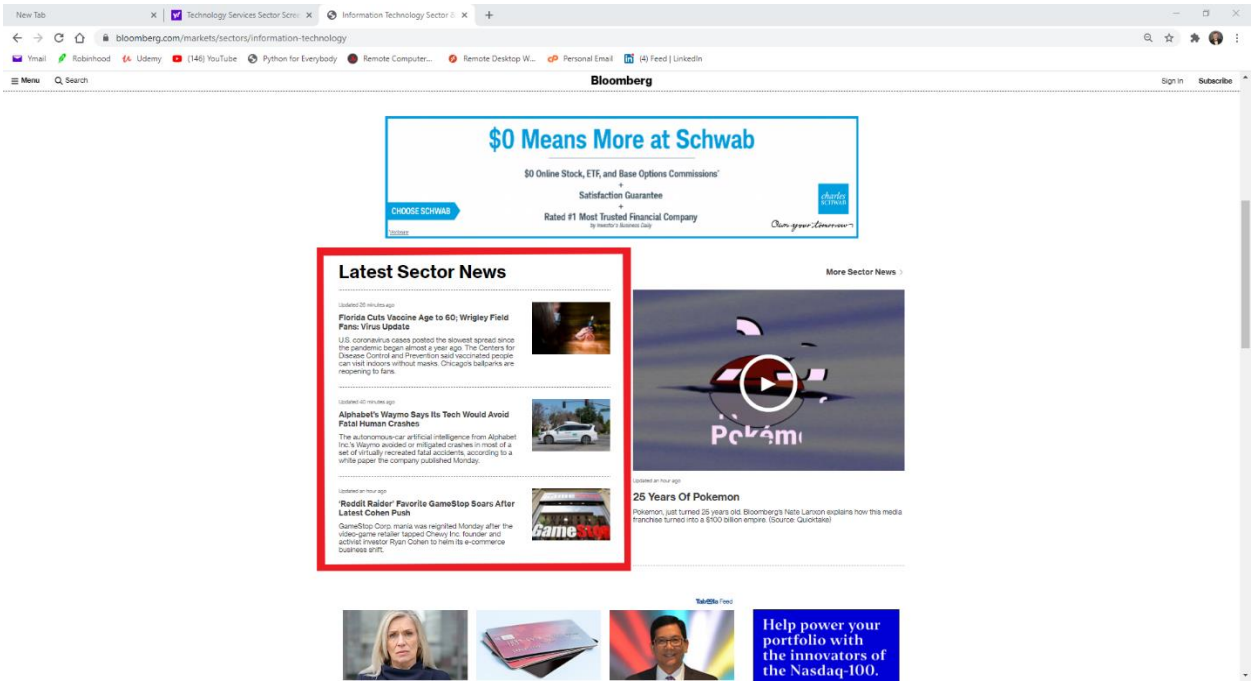


Figure1.2-1 Bloomberg's latest technology sector news

2.2.1 Connecting to the Data

First, I would like to note that no new libraries were needed for this process, however, I did import the html5lib library into the "bloombergScraper.py" file to use as my Beautiful Soup HTML parser because it seemed to work more accurately for all the websites. The reason html5lib worked more effectively was because it allows you to include "request headers" to give the request context and return a more accurate response. To begin with, I visited the link mentioned above through Google Chrome and used the inspection tool to locate the tags containing ***all*** the elements I needed to iterate through and extract. I found that the recent article section consisted of 3 tags with matching class names, indicating the containers for the 3 articles. Since only the elements I want to iterate through are stored in content variable, I can use a single for loop to extract the title, date, description, and link to article.

bloombergScraper.py

```
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup
import requests

link = "https://www.bloomberg.com/markets/sectors/information-technology"
req = Request(link, headers={'User-Agent':'Mozilla/5.0'})
webpage = urlopen(req).read()

with requests.Session() as c:
    soup = BeautifulSoup(webpage, 'html5lib')
    content = soup.find_all('div', attrs={'class': 'sector-related-content__item'})
```

Figure1.2.1-1 Creating Response & Beautiful Soup Objects.

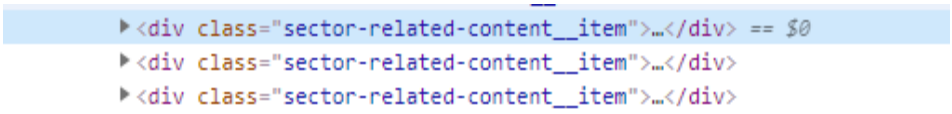


Figure1.2.1-2 Latest Sector News tags.

2.2.2 Scraping the title

To scrape the title from the Bloomberg website, I used the inspection tool in google chrome and found that the text for the title was found in an <a> tag. However, I had an issue with referencing the <a> tags by their class names. Since there were no other <a> tags in each of the 3 tags containing a hyperlink, I used the find method for the <a> tags where a hyperlink was present. Last, I added the test method to the end to return the Unicode string for each of the three titles.

```
for item in content:
    title = item.find('a', href=True).text
```

2.2.3 Scraping the description and publication date

Next, to scrape the publication date I repeated the process of locating the element tags containing the time, which were actual <time> tags. Since there was only one-time tag in each article container there was no need to define special attributes. This was also the case with description except it was in a tag instead of <time>. So, I just defined the time and description variables inside of the for loop and used the "find" method and text methods to return the data in the correct format for each article.

```
time = item.find('time').text
des = item.find('span').text
```

2.2.4 Scraping the hyperlink

Lastly, I scraped the hyperlinks to the articles which forced me to take a different approach than any of the previous data I had scraped. The reason for this being that, previously, I have only had to scrape data and store it as textual/string data, except for the stock prices. However, to scrape an actual link, I used the same find method that I used for the title except, instead of using the text method, I used the ['href'] method, which returns the value in the "href"(the hyperlink to the article.)

```
raw_link = (item.find('a', href=True)['href'])
```

2.2.5 Looping Through the Latest Sector News Items

After defining the previously mentioned variables and placing them inside the for loop, each item is carefully extracted and ready to be stored in a proper data frame or format.

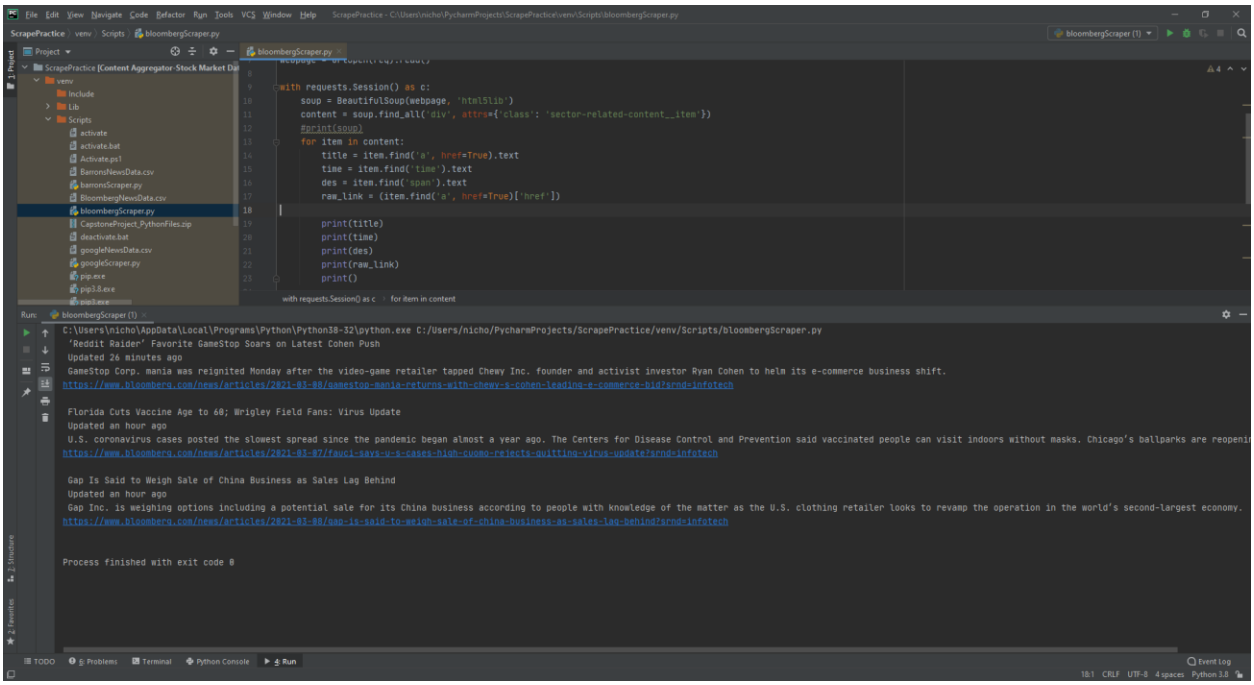


Figure1.2.5-1 Scraping and Printing Bloomberg News Article items.

2.2.6 Storing the article data

Lastly, I used the csv library and string formatting to store the data in a csv file. However, on the first attempt I realized there was an error somewhere when I opened the csv file and saw that not all data was stored in the appropriate columns. After a combination of debugging and researching, I found that this error was being caused by commas in the actual strings being used as the delimiter to create the "comma separated file". So, I used the replace() function to replace any commas in the title, description or time to be replaced with a blank space.

A	B
DBS Cuts Bonus of CEO	Senior Managers After Profit Decline
Atari Setting Up Crypto Casino to Tap Into Nostalgia and NFTs	Updated 2 hours ago
Chinese Beauty App Becomes First Major Company to Buy Ether	Updated 3 hours ago

Figure1.2.6-1 Data Stored incorrectly.

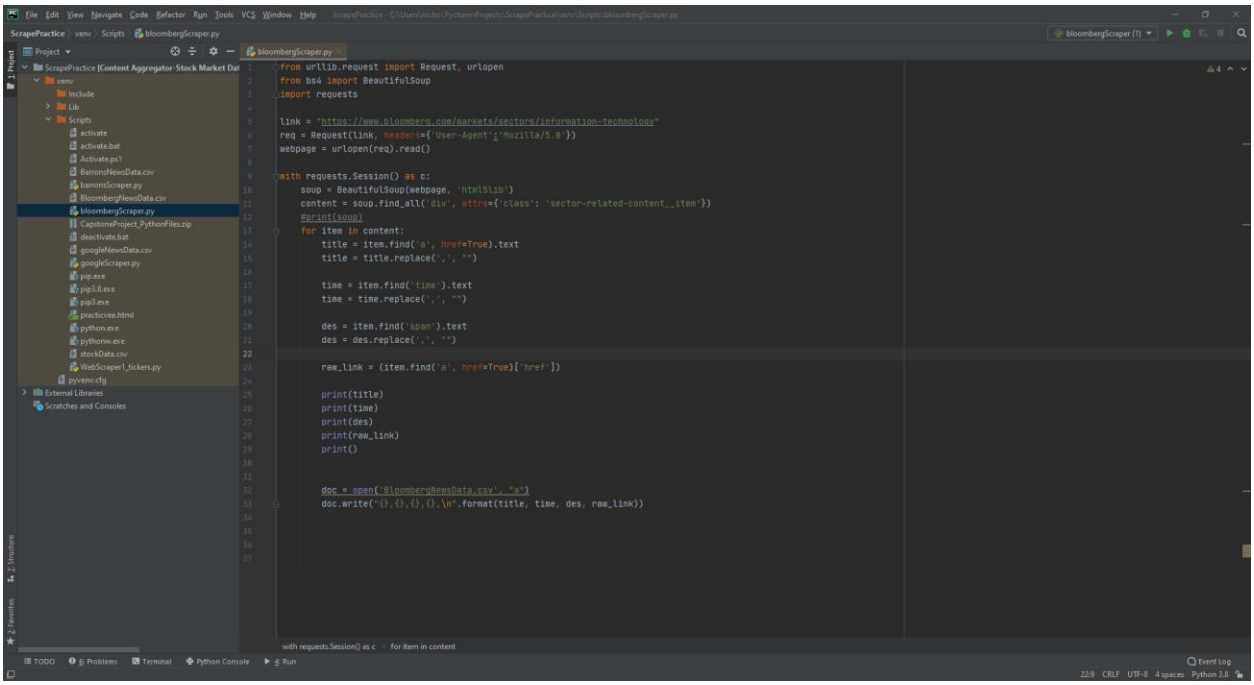


Figure1.2.6-2 Updated for loop.

	A	B
1	DBS Cuts Bonus of CEO Senior Managers After Profit Decline	Updated an hour ago
2	Atari Setting Up Crypto Casino to Tap Into Nostalgia and NFTs	Updated 2 hours ago
3	Chinese Beauty App Becomes First Major Company to Buy Ether	Updated 3 hours ago

Figure1.2.6-3 Data Stored Correctly.

2.3 Web Scraping: Barron's News Article Data

The next website I chose to scrape News article data from was the 3 most recent news articles found at https://www.barrons.com/topics/technology?mod=BOL_TOPNAV. Connecting to the Barron's data was the same as connecting to the Bloomberg data, other than changing the URL to request the appropriate address. However, the elements on this website were a bit more embedded, which required more detailed uses of the find method.

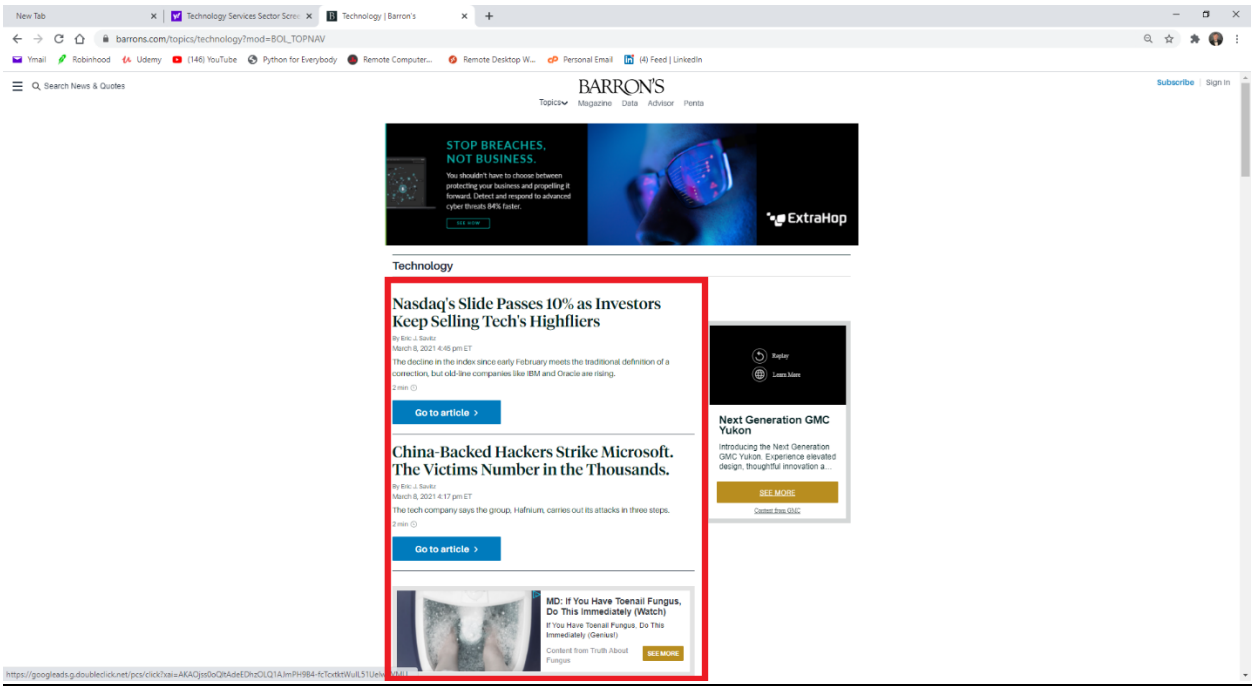


Figure1.3-1 Barron's Latest Technology Sector News

2.3.1 Scraping the Title and Hyperlink

First off, I repeated the process of visiting the website through Google Chrome and inspecting the elements that contained each of the items I wanted for individual articles. After locating the tag with the correct class name, I noticed it was pulling every single article container on the page, while I only needed the data from the top 3. Therefore, I set my "for" loop to only loop through the first 3 items. Then, the title was very simple because it was the only <a> tag inside of each of the tags, similarly to the Bloomberg data. The hyperlink also followed the same scraping technique used to scrape the Bloomberg data.

```
for item in content[0:4]:
    title = item.find('a').text
    raw_link = (item.find('a', href=True)['href'])
```

2.3.2 Scraping the description and publication date

Since the description and date were more embedded in the container, it did take some more specifications to properly extract data from the correct elements. Unlike the title, the description was not stored in the only tag of its kind, therefore, unique class attributes had to be identified. Furthermore, the time for each article was even more deeply embedded. Not only was the time stored inside of a child , but it was also stored inside of another <p> tag. So, both elements as well as their class attributes had to be defined in the find function.

```
time =
item.find('div', attrs={'class': 'BarronsTheme--wrapper-1-0--2f3XwkpR'}).find('p',{'class':'BarronsTheme--
timestamp--3V5jp2-R BarronsTheme--TopicsArticle--1XIXBXRt BarronsTheme--zero-margin-timestamp--
YIWqgHaW'}).text

description =
```

```
item.find('p', attrs={'class':'BarronsTheme--summary--3UHA7uDx'}).text
```

2.3.3 Looping Through the Latest Sector News Items

After defining the previously mentioned variables and placing them inside the for loop, each item is carefully extracted and ready to be stored in a proper data frame or format. However, since I encountered the .csv issue with storing items in the incorrect column, I went ahead and used the replace() function to be cautious.

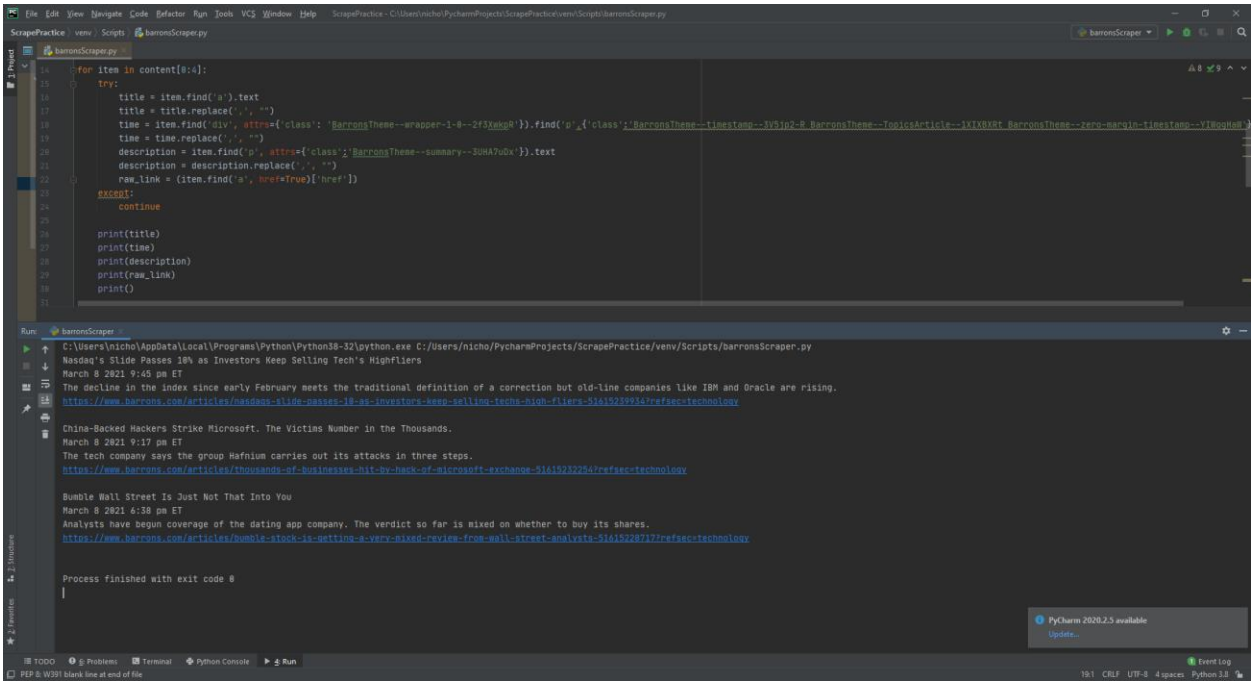


Figure1.3.3-1 Scraping and Printing Barron's News Article items.

2.3.4 Storing Barron's Article Data

Since I went ahead and updated each variable in the for loop using the replace function for any commas, the data properly stored in a csv file.

	A	B
1	Nasdaq's Slide Passes 10% as Investors Keep Selling Tech's Highfliers	March 8 2021 9:45 pm ET
2	China-Backed Hackers Strike Microsoft. The Victims Number in the Thousands.	March 8 2021 9:17 pm ET
3	Bumble Wall Street Is Just Not That Into You	March 8 2021 6:38 pm ET

Figure1.3.4-1 Barron's data stored correctly.

2.4 Web Scraping: Google Search Results

Instead of choosing a different news company website, I decided to scrape the top 3 articles posted within the last 24 hours from the Google Search engine "News" feature using the search query "technology stocks". The URL I am sending a request to is

https://www.google.com/search?q=technology+stocks&tbm=nws&source=lnt&tbs=qdr:d&sa=X&ved=0ahUKEwjs8JXwpJ_vAhURLs0KHRryCsEQpwUIKQ&biw=1920&bih=937&dpr=1

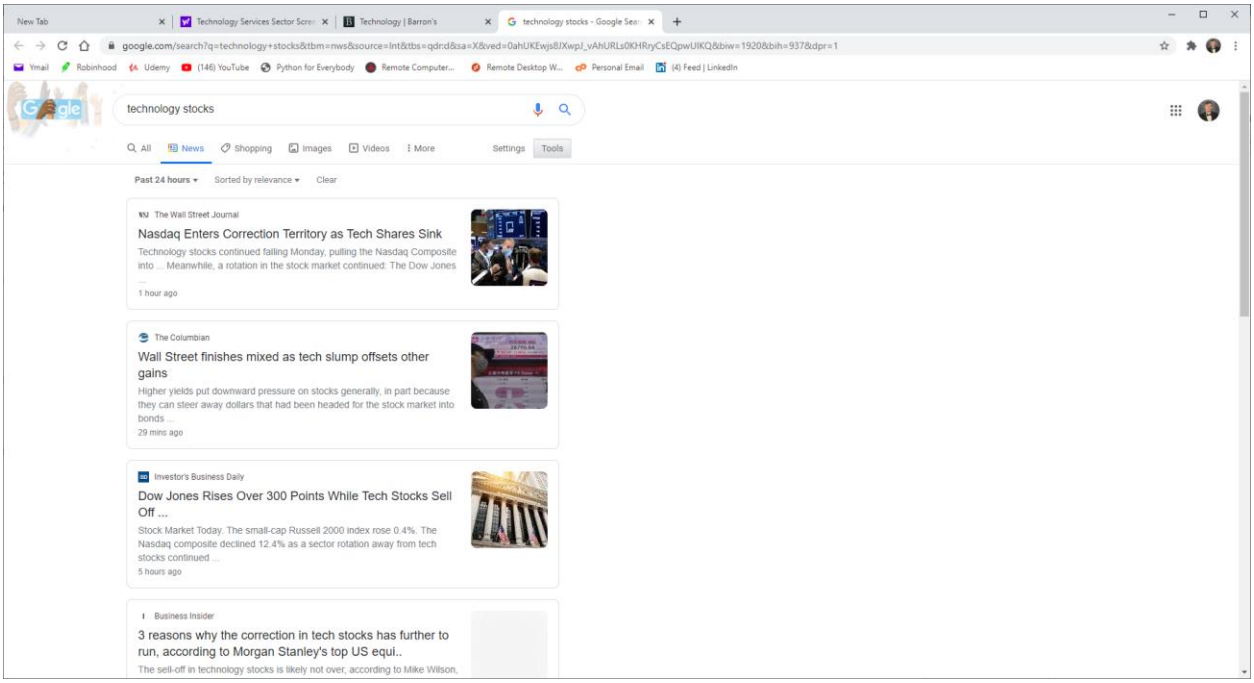


Figure1.3-1 Google news search results for: technology stocks.

2.4.1 Scraping the Data

I took the exact same approach to locating the correct container to iterate through as I did the previous two times. Moreover, scraping the items was also very similar to the other websites, with a few additional steps. The additional steps involved separating the time and description to two separate values by using the split() function to separate the data on the bullet point.

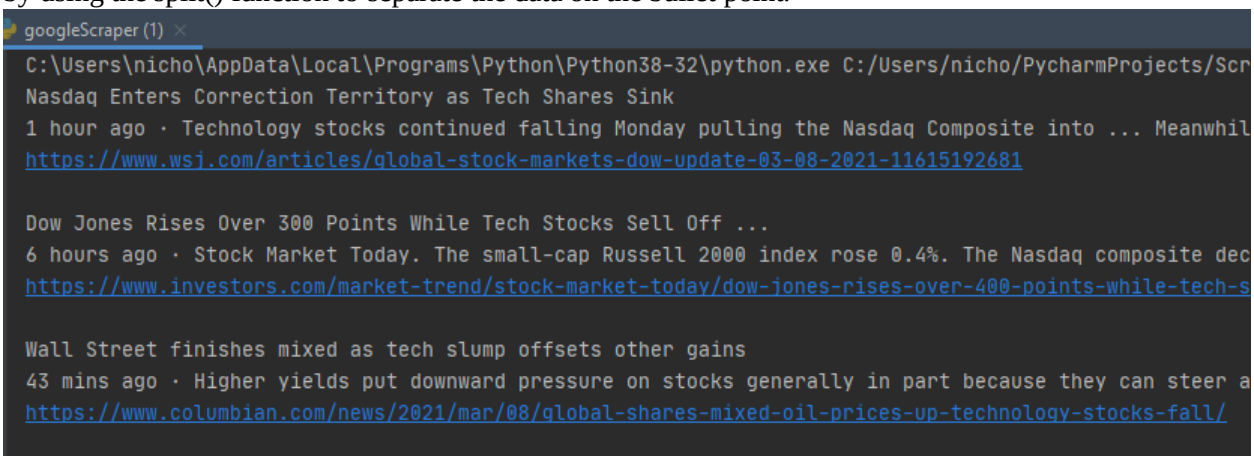


Figure1.4.1-1 Incorrectly extracting time and description.

```
    for item in content[0:3]:
        title = item.find('div', attrs={'class': 'BNeawe vvjwJb AP7Wnd'}).text
        title = title.replace(',',"")

        raw_link = (item.find('a', href=True)['href'])
        link = (raw_link.split("/url?q=")[1]).split('&sa=U&')[0]

        description = item.find('div', attrs={'class': 'BNeawe s3v9rd AP7Wnd'}).text
        description = description.replace(',',"")
        time = description.split(" · ")[0]
        intro = description.split(" · ")[1]

        print(title)
        print(time)
        print(intro)

        print(link)
        print()
```

Figure1.4.1-2 Using the split function to correct extraction.

```
googleScraper (1)
C:\Users\nicho\AppData\Local\Programs\Python\Python38-32\python.exe C:/Users/nicho/PycharmProjects/ScrapePractice/venv/Scripts/googleScraper.py
Nasdaq Enters Correction Territory as Tech Shares Sink
1 hour ago
Technology stocks continued falling Monday pulling the Nasdaq Composite into ... Meanwhile a rotation in the stock market continued: The Dow Jones...
https://www.wsj.com/articles/global-stock-markets-dow-update-03-08-2021-11615192681

Dow Jones Rises Over 300 Points While Tech Stocks Sell Off ...
6 hours ago
Stock Market Today. The small-cap Russell 2000 index rose 0.4%. The Nasdaq composite declined 12.4% as a sector rotation away from tech stocks continued...
https://www.investors.com/market-trend/stock-market-today/dow-jones-rises-over-400-points-while-tech-stocks-sell-off-treasury-yields-near-recent-highs/

Stocks rise broadly on Wall Street even as technology lags
14 hours ago
Tech stocks were mostly lower. Apple fell 2.7% and Facebook was down 1.5%. Trading has been choppy in recent weeks as investors fret over a sudden spike in...
https://wqnradio.com/news/business-news/global-shares-mixed-oil-prices-up-technology-stocks-fall/
```

Figure1.4.1-3 Correctly extracting time and description

2.4.2 Storing the data

I used the exact same method for storing the data in a csv file as in the previous two examples. Since the replace function was already included, the data stored correctly.

	A	B
1	Nasdaq Enters Correction Territory as Tech Shares Sink	1 hour ago
2	Dow Jones Rises Over 300 Points While Tech Stocks Sell Off ...	6 hours ago
3	Stocks rise broadly on Wall Street even as technology lags	14 hours ago

Figure 1.4.2-1 Data stored Correctly.

3 Midterm Summary

Now that I have explored the art of web scraping and have successfully found a way to collect the data in the correct format whenever I need, it is time to introduce Django. Django is an extremely powerful front-end framework for Python. I will use this framework to store, manage, and display my content in an organized website format.

4 The Django Framework

4.1 Hosting

Before I began using the Django framework, I switched from the “Pycharm” IDE to the “Atom” text editor. I found that my project would be more compatible, portable, and dependable by creating a virtual environment and testing my project files in the command prompt. However, I did still test the project/run specific files in PyCharm periodically.

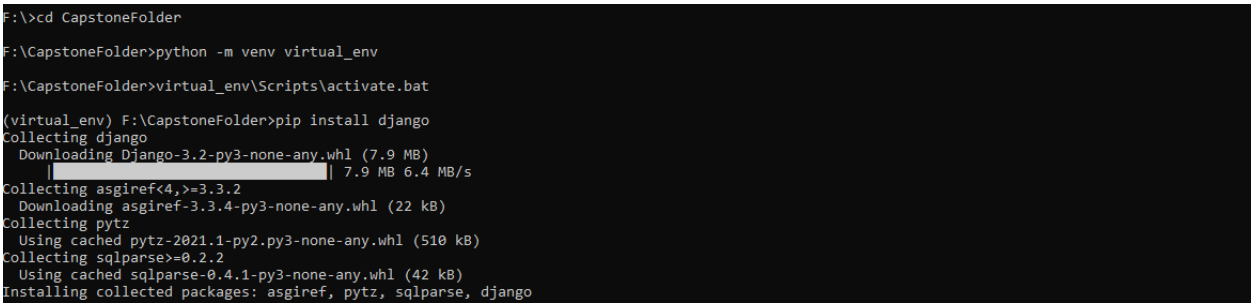


Figure 3.1-1- Creating Virtual Environment

4.2 Creating a Django Project

After installing Django and other necessary dependencies, and an extensive period of trial and error exploring the functionality of Django, I found that the framework operates on a very simple three “MVT” architecture which consist of, the Model, View, and Template files. A Django project, like any website, can contain multiple applications that are used for different functionalities, and each Django application contains the “MVT” architecture. The Model is a Python script used to create, modify, or delete database tables in the SQLite3 database that comes with a Django project by default. The View is Python file that enables you to specify the logic in which you want to display/render data to the webpage. The template is a folder that is created conventionally to store all the HTML/CSS files that make up the front-end of the website. The following screenshots contain the simple commands I ran to create my Django project and successfully start the Django server.

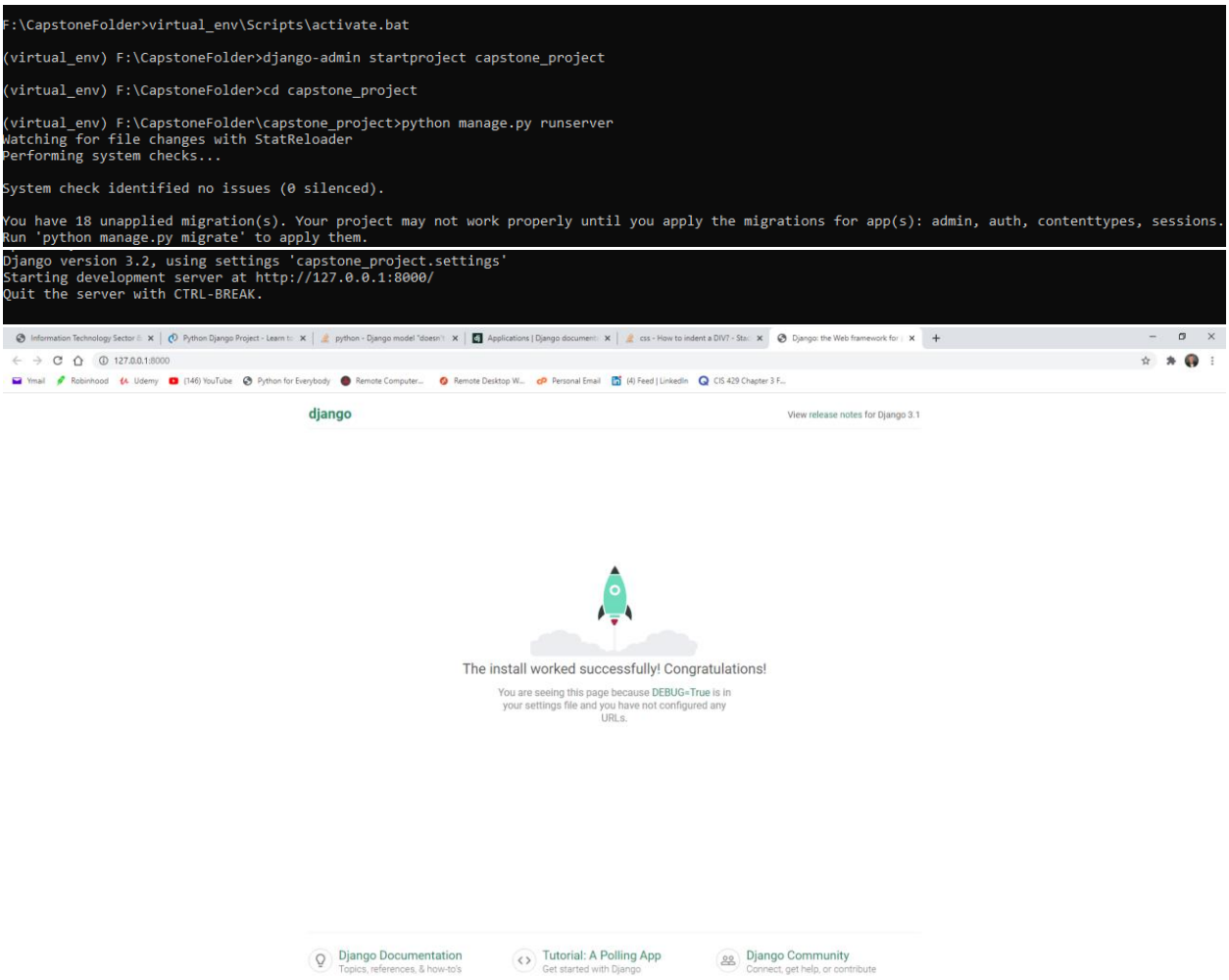


Figure 3.2-1 Django server running.

Figure 3.2-2- Django project directory created as 'capstone project'.

4.3 Creating the News App

The next process involved is creating an application (websites can contain multiple applications). The application was also created in the command prompt as demonstrated below.

```
(virtual_env) F:\CapstoneFolder\capstone_project>python manage.py startapp news_app

(virtual_env) F:\CapstoneFolder\capstone_project>
```

Figure 3.3-1 creating the news application.

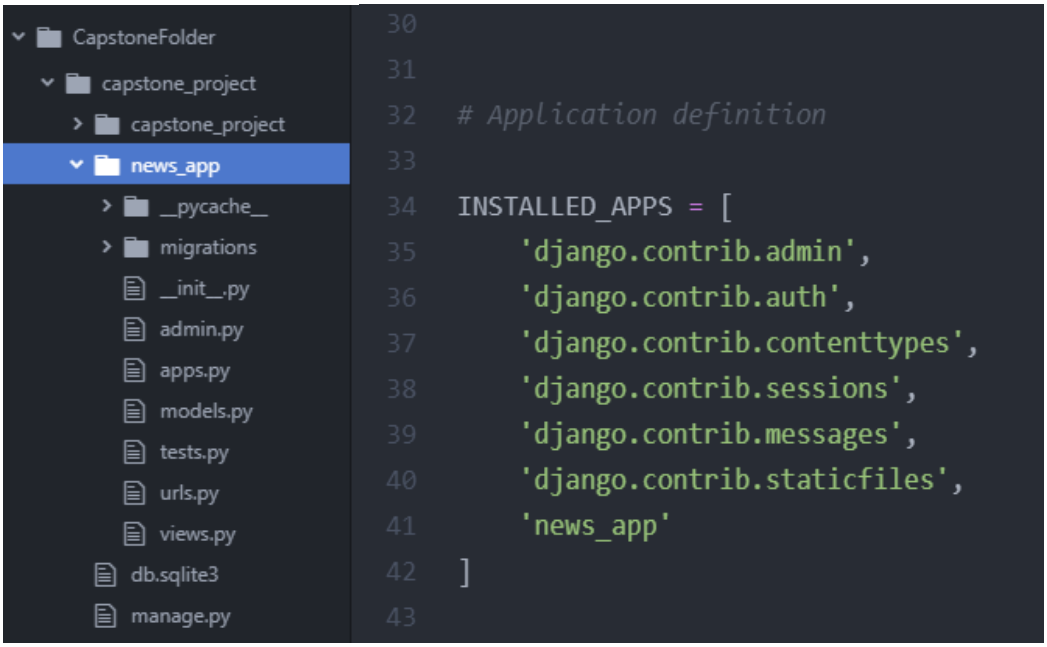


Figure 3.3-2- news app file directory created. Figure 3.3-3 – configuring project settings with new application.

4.3.1 Developing the first View

In Django, the views script works as a function or class to send a request and return a response. However, before anything could be requested or returned, I had to configure my project "urls.py" file to point to my function in the "views.py" file inside my new application. By using the HttpResponse function, the views file can return a response displaying the data in the browser straight from the python script rather than querying from elsewhere. The following screenshots show my initial experience/success with displaying data.

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse('<h1>My first HTTP Response</h1>')

# Create your views here.
```

Figure 3.3.1-1 creating first view containing the index function.

```
from django.conf.urls import url
from news_app import views
urlpatterns = [
    url(r'', views.index, name='index'),

]
```

Figure.3.3.1-2 - creating the news_app/urls.py file.

```
from django.contrib import admin
from django.conf.urls import url, include
from news_app import views

urlpatterns = [
    url(r'', include('news_app.urls')),
    url(r'admin/', admin.site.urls),
]
```

Figure 3.3.1-3 Configuring project urls.py file with app urls.py.

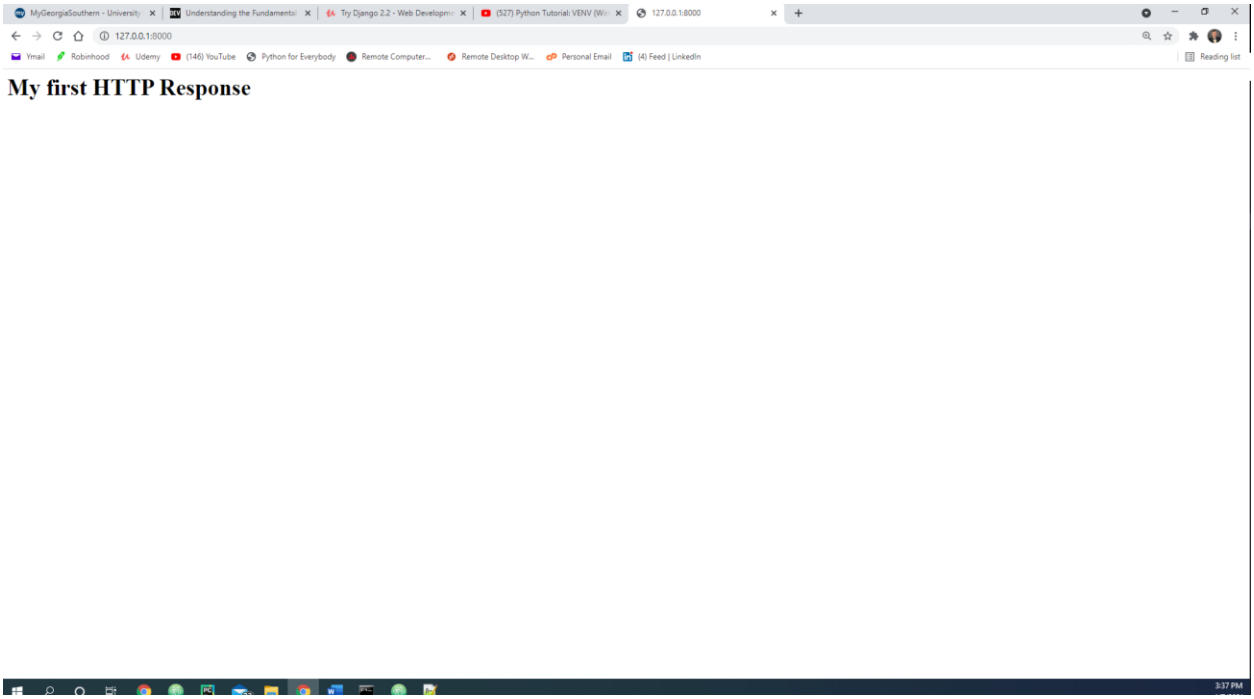


Figure 3.3.1-4 -My first HTTP response my first view

4.3.2 Implementing the web scraper function into the view

My next goal is to use the HttpResponse function to list items in the browser that my web scraper function generates. The first step is to replace the “index” function in the views file to the new function “news_scrape”, then reconfigure my news_app/urls.py file from pointing to “views.index” to “views.news_scrape”.

```
(virtual_env) F:\CapstoneFolder\capstone_project>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

Exception in thread django-main-thread:
Traceback (most recent call last):
  File "C:\Users\nicho\AppData\Local\Programs\Python\Python38-32\lib\threading.py", line 932, in _bootstrap_inner
    self.run()
  File "C:\Users\nicho\AppData\Local\Programs\Python\Python38-32\lib\threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\utils\autoreload.py", line 64, in wrapper
    fn(*args, **kwargs)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\core\management\commands\runserver.py", line 118, in inner_run
    self.check(display_num_errors=True)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\core\management\base.py", line 419, in check
    all_issues = checks.run_checks(
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\core\checks\registry.py", line 76, in run_checks
    new_errors = check(app_configs=app_configs, databases=databases)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\core\checks\urls.py", line 13, in check_url_config
    return check_resolver(resolver)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\core\checks\urls.py", line 23, in check_resolver
    return check_method()
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\urls\resolvers.py", line 412, in check
    for pattern in self.url_patterns:
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\utils\functional.py", line 48, in __get__
    res = instance.__dict__[self.name] = self.func(instance)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\urls\resolvers.py", line 598, in url_patterns
    patterns = getattr(self.urlconf_module, "urlpatterns", self.urlconf_module)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\utils\functional.py", line 48, in __get__
    res = instance.__dict__[self.name] = self.func(instance)
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\urls\resolvers.py", line 591, in urlconf_module
    return import_module(self.urlconf_name)
  File "C:\Users\nicho\AppData\Local\Programs\Python\Python38-32\lib\importlib\__init__.py", line 127, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 1014, in _gcd_import
  File "<frozen importlib._bootstrap>", line 991, in _find_and_load
  File "<frozen importlib._bootstrap>", line 975, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 671, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 783, in exec_module
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "F:\CapstoneFolder\capstone_project\capstone_project\urls.py", line 21, in <module>
    url(r'', include('news_app.urls')),
  File "F:\CapstoneFolder\virtual_env\lib\site-packages\django\urls\conf.py", line 34, in include
    urlconf_module = import_module(urlconf_module)
  File "C:\Users\nicho\AppData\Local\Programs\Python\Python38-32\lib\importlib\__init__.py", line 127, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 1014, in _gcd_import
  File "<frozen importlib._bootstrap>", line 991, in _find_and_load
  File "<frozen importlib._bootstrap>", line 975, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 671, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 783, in exec_module
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "F:\CapstoneFolder\capstone_project\news_app\urls.py", line 4, in <module>
    url(r'', views.index, name='index'),
AttributeError: module 'news_app.views' has no attribute 'index'
```

Figure 3.3.2-1 ERROR did not reconfigure urls.py.

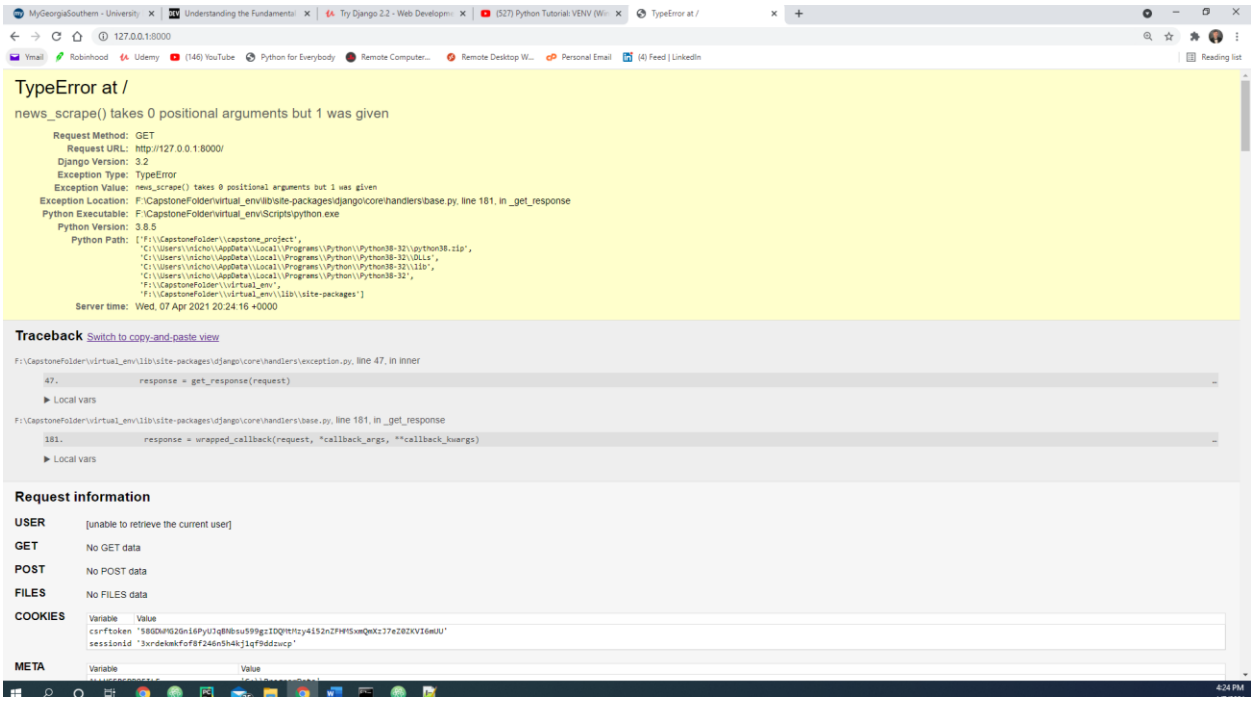


Figure 3.3.2-2-ERROR- forgetting to use "requests" parameter in new function

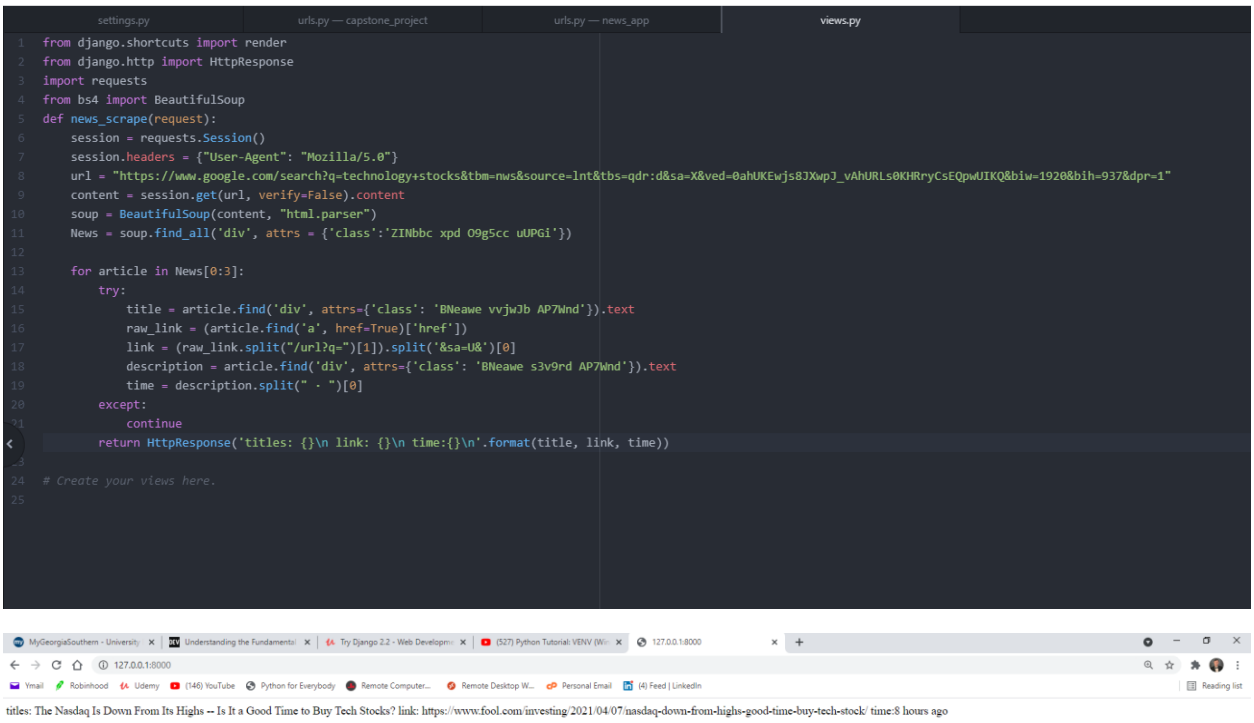


Figure 3.3.2-3 successfully listed titles in browser using HTTPResponse.

Next, I repeated the process for the other two web scraping scripts by placing them inside the news_scrape function and successfully rendered data to the browser with both. I have been extensively researching the proper way to get/request data in Django, and found that “render”, “redirect”, and “HttpResponse” are the most commonly used methods. However, unless your application is extremely small, using the “HttpResponse” method can get quite repetitive, as well as extensive. Since I was aware that my website would contain multiple different pages (HTML templates), I chose to use the render method for requesting specific data for HTML templates.

4.3.3 Introducing Templates with Bootstrap

As previously mentioned, since I intended on creating multiple webpages inside my website, it is conventional to create a “templates” folder inside my Django project directory (capstone_project), which will contain all of my static HTML/CSS files. However, after the new folder has been created, I had to configure the Django project “settings.py” file to look for my html files in the “templates” directory. Next, I created my first template “index.html”, and will be using the “Bootstrap” for my front-end framework.

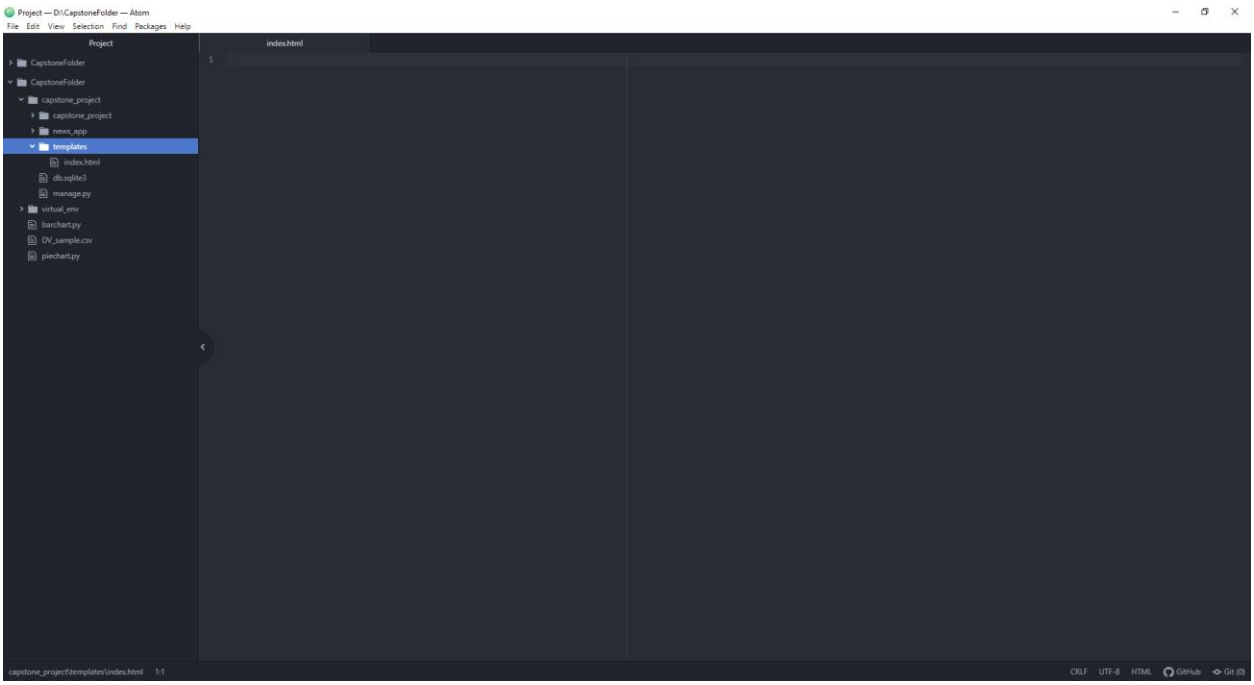


Figure 3.3.3-1 creating first template inside the new directory for templates.

Bootstrap is the most popular CSS Framework for developing responsive and mobile-first websites. It's an open-source framework that contains numerous pre-created templates for users to copy and customize to their liking. For starters, I simply went their website, and copied/pasted their start template to my index template.

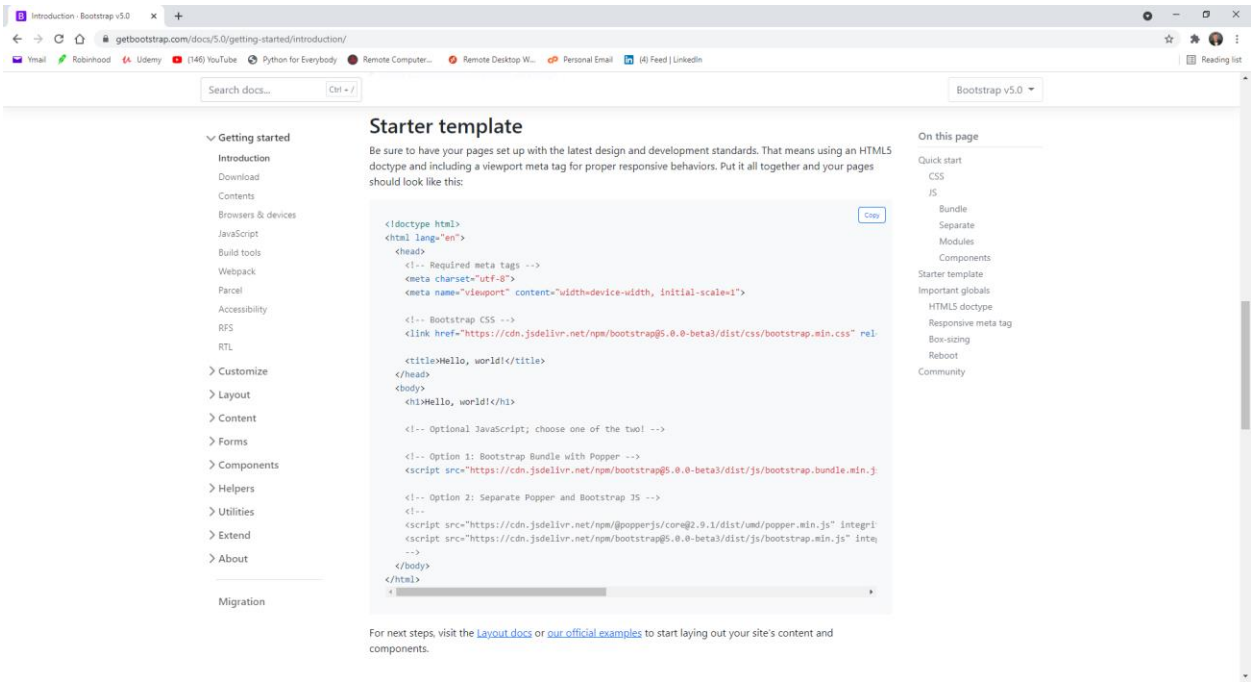


Figure 3.3.3-2- Copying the Bootstrap starter template

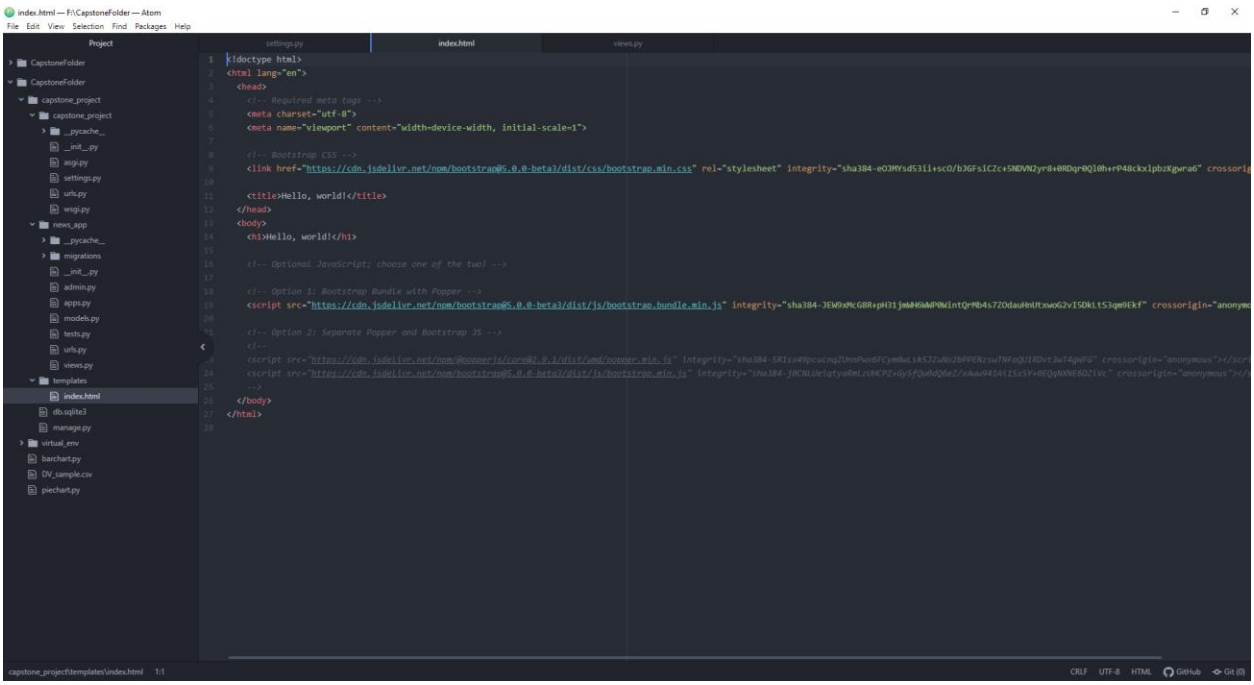


Figure 3.3.3-2 index.html with title "hello world"

The next step in the process is to implement the “render” function inside my views.py file in order to return the new “index.html” file in the browser.

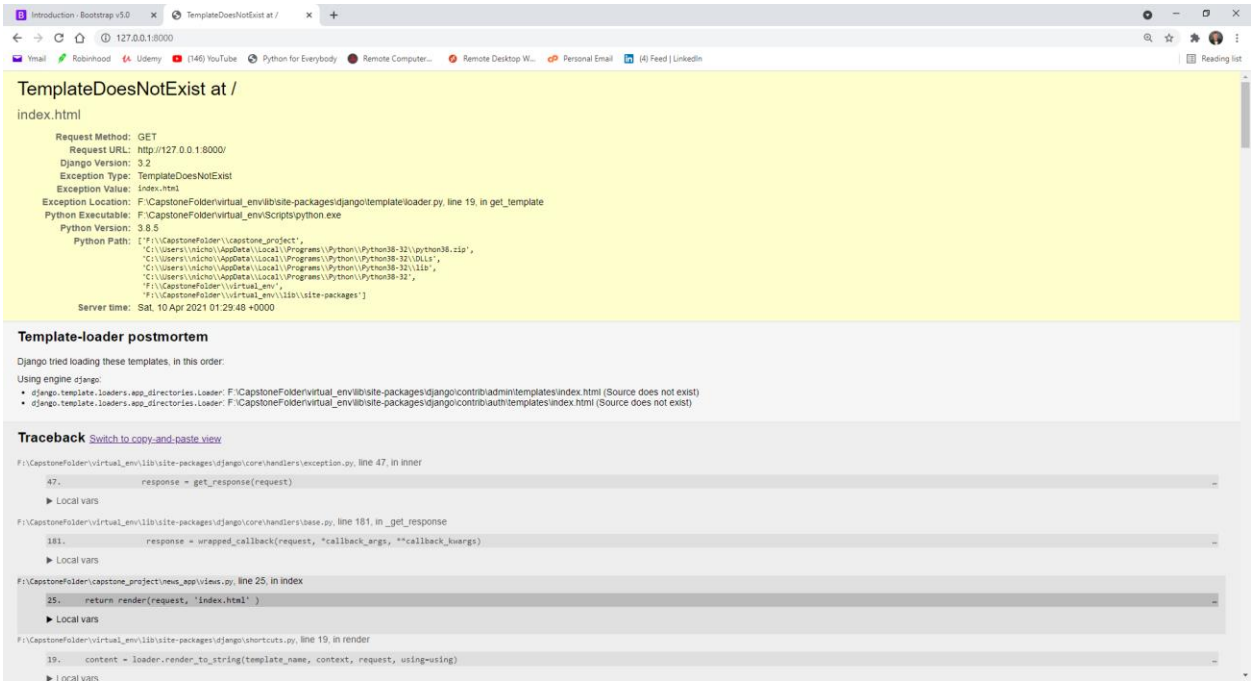


Figure 3.3.3-3 ERROR- "index.html" template does not exist.

The error message above is being displayed because the server cannot find my templates. This is because I did not properly configure the settings.py file to point to the “templates” directory. By creating the variable shown on line 17 below, and then adding to the variable “TEMPLATES” list. Django then

knows to look inside the project directory for a folder names “templates”, for rendering HTML files. This is all credited to Django’s unique “MVT” paradigm.

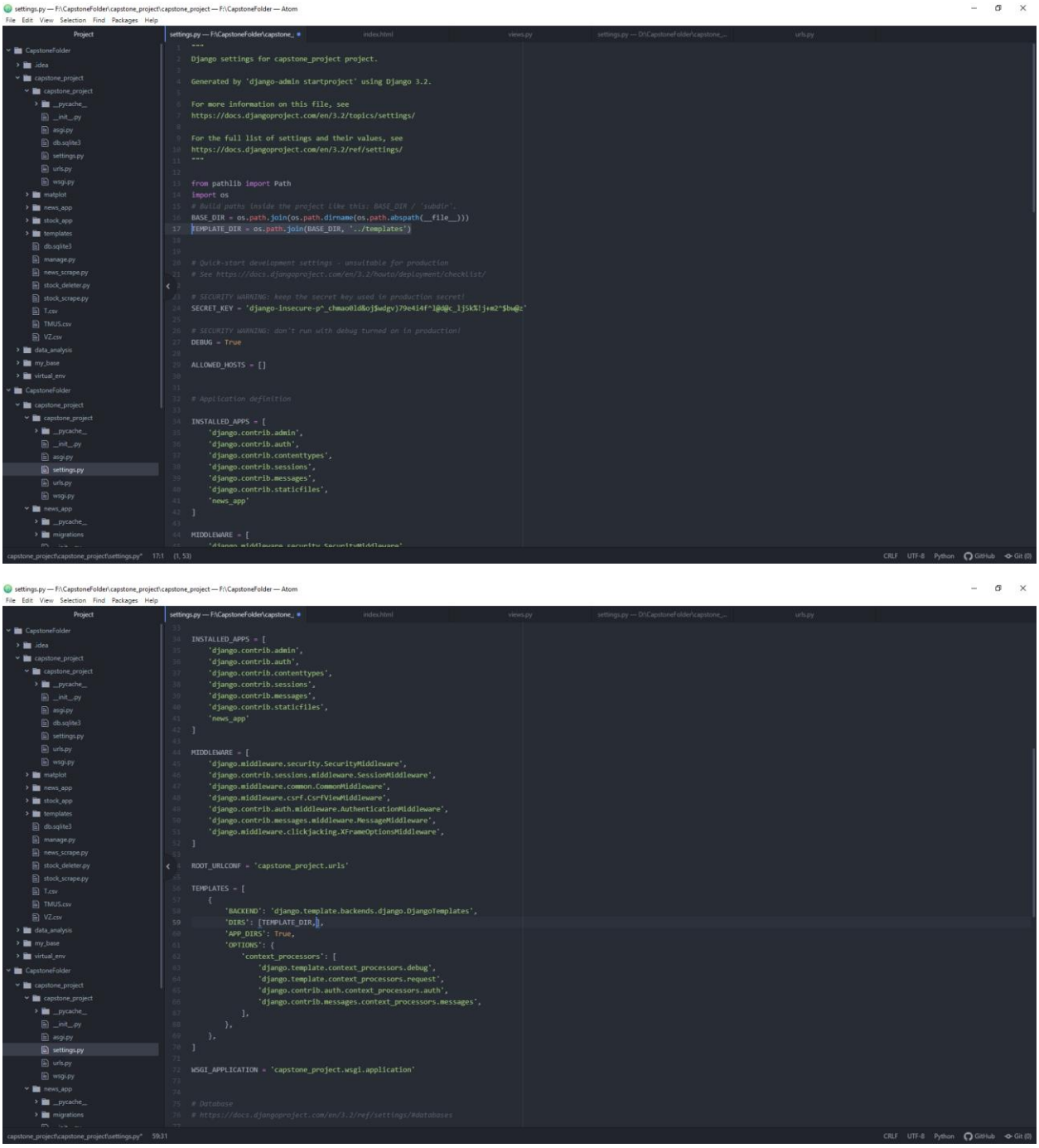


Figure 3.3.3-4 adding new variable to project templates

After making the previous changes, and reloading the webpage, the “index.html” file containing the bootstrap stater template was successfully rendered.

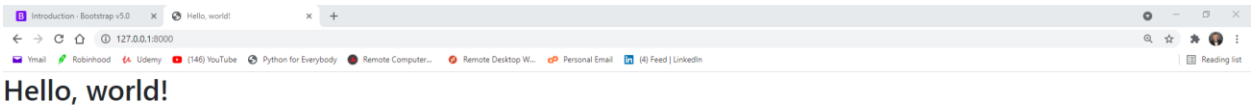


Figure 3.3.3-4 Rendering my first HTML template.

4.3.4 Using Django Context Processers

One of the most powerful, useful features of Django are their context processors. These processors are written in a language of their own and enable users to uniquely populate web pages with data generated by the views when a template is rendered with a request. Essentially, connecting python's backend capabilities with front-end language capabilities. This unique language consists of two main operators within its syntax: {% block content %} for using python logic inside HTML files, and {{content}} for printing data generated from the python function in the views.py file. For experimental purposes I will place the titles from one of my scraping functions inside the views file, append them to a list, place proper context processors in the index.html webpage and attempt displaying the titles in the browser.

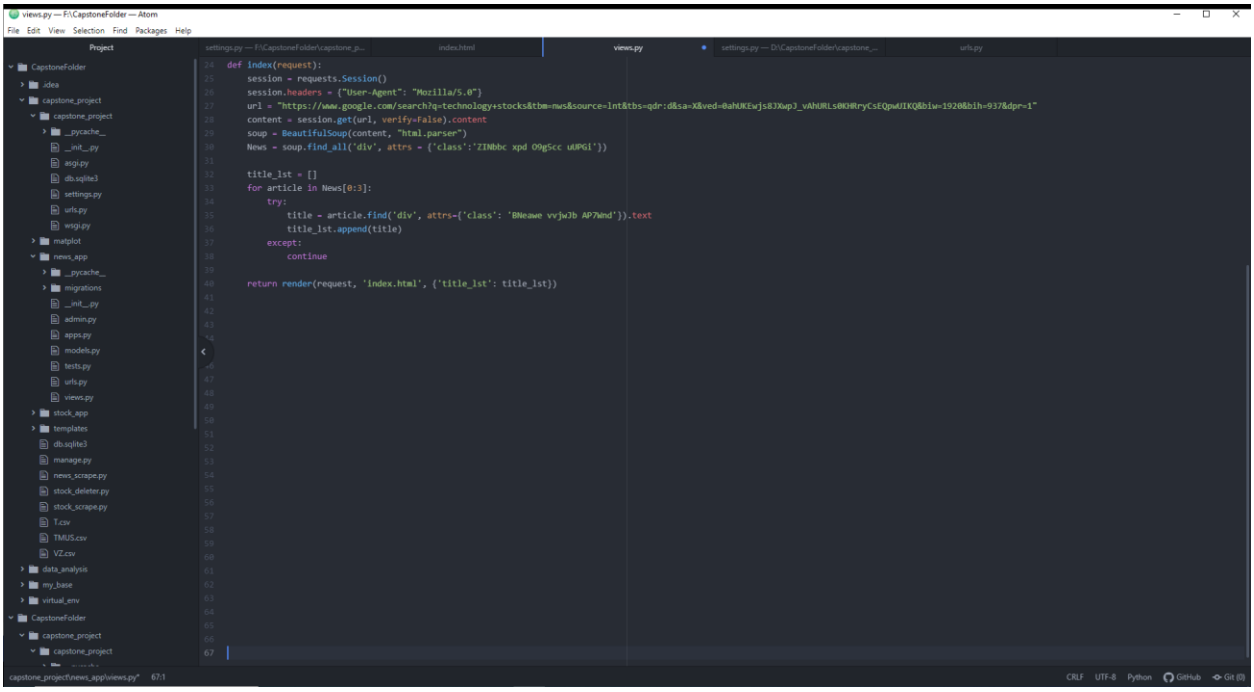


Figure 3.3.4-1 placing titles in a list in views.py.

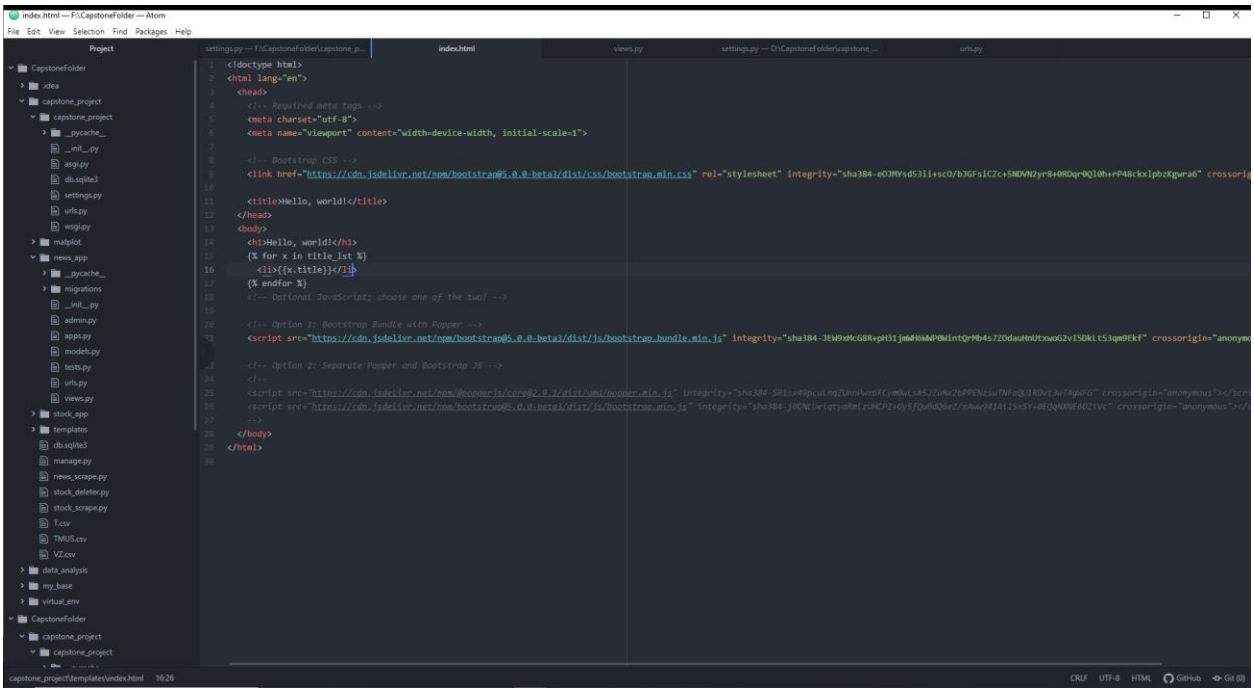


Figure 3.3.4-2 Using context Processors in index.html.

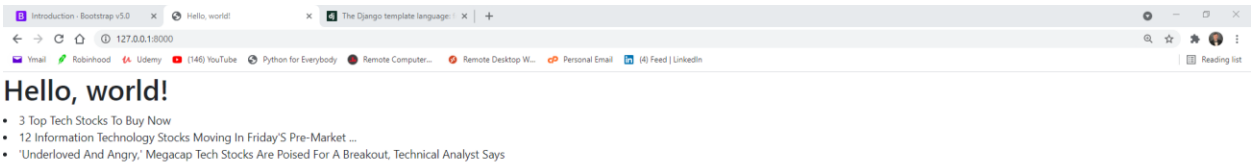


Figure 3.3.4-3 displaying the list of titles

After testing all scrape functions as demonstrated above, I ran into issues with displaying the most recent data as well as having to constantly reboot the server to update the content. Also, I anticipated that I would need to find a different technique in order to separate the article's content by *website*. Eventually, I realized this was because I had not yet utilized the "Model" phase of the Django architecture which involves storing/managing the data in relational database tables (SQLite3).

4.3.5 Creating a Django Model

On the instance of creating a Django application, inside the new application that is automatically created, there is a "models.py" file that is used for creating tables in the "SQLite3.db" database that comes with the Django project by default. In the following procedure I will create 3 database tables (one for each website) inside the "news_app/models.py" file, and then run the necessary commands to physically create the tables in the database.

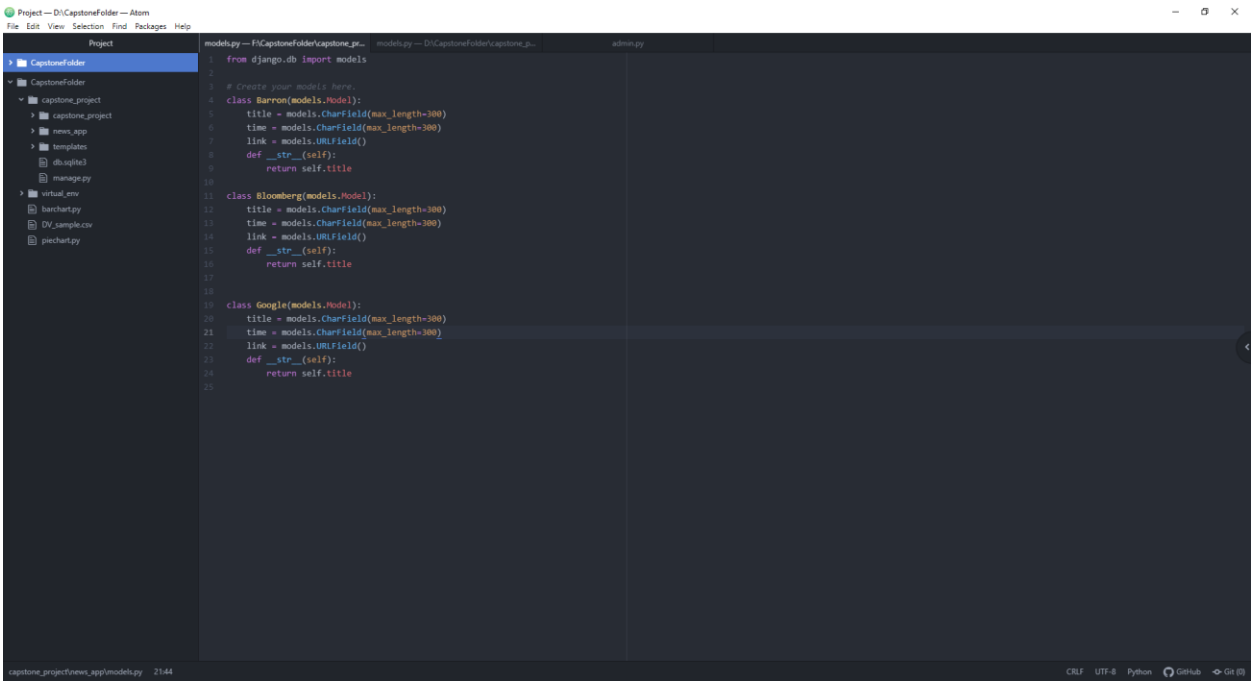


Figure 3.3.5-1 Creating Database Tables (models) for each website.

The code shown above contains 3 classes, each one represents a table for a different website, and each table contains 3 fields. The 1st field is created to store the article's title, the 2nd is for the time the article was posted, and the last field is a unique "URL field" that will be used to store the hyperlink to the actual article. Next, I will make my migrations in the command prompt to physically initialize and create the tables in the database and create a "super user" account. Then, by adding "/admin" to the end of my local IP address and logging in with the superuser credentials I previously created, I will be able to manage and view the tables through the administrator panel (GUI).

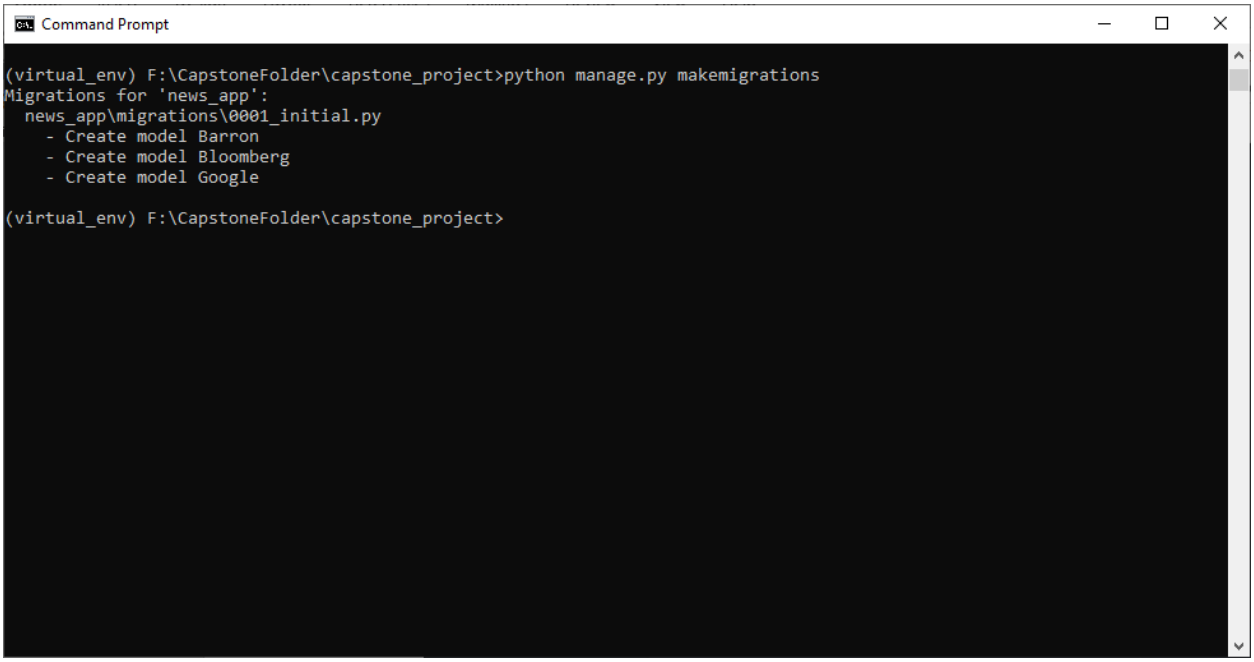


Figure 3.3.5-2 Making Migrations

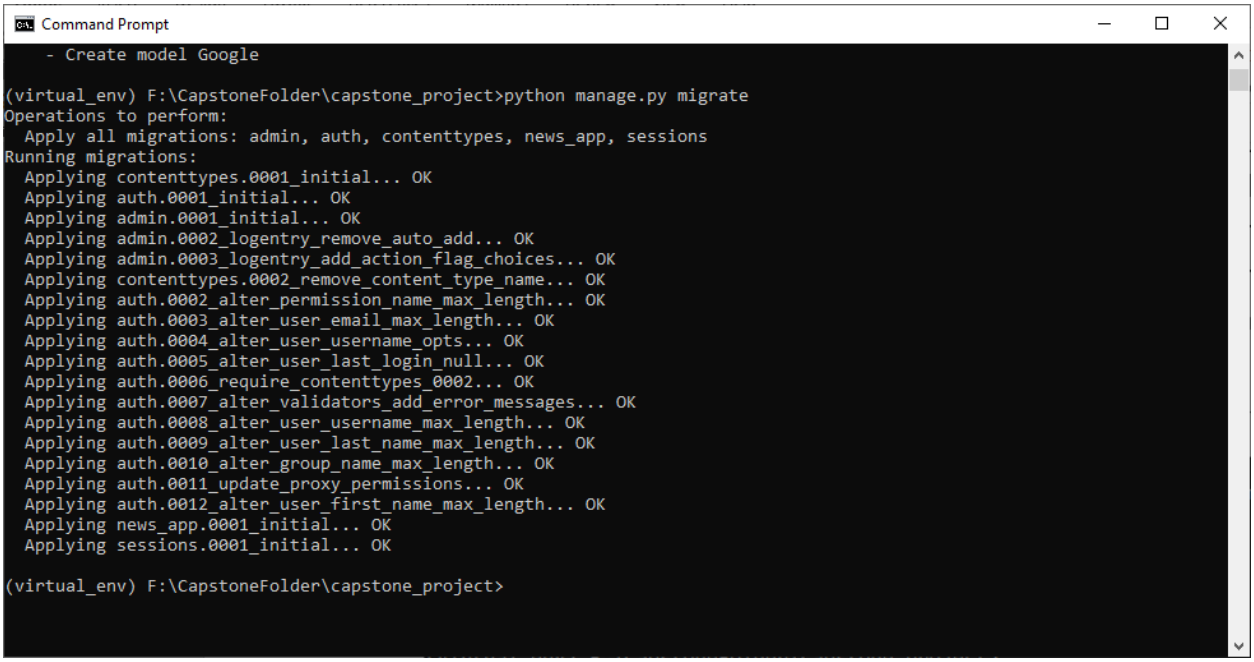


Figure 3.3.5-3New tables successfully created

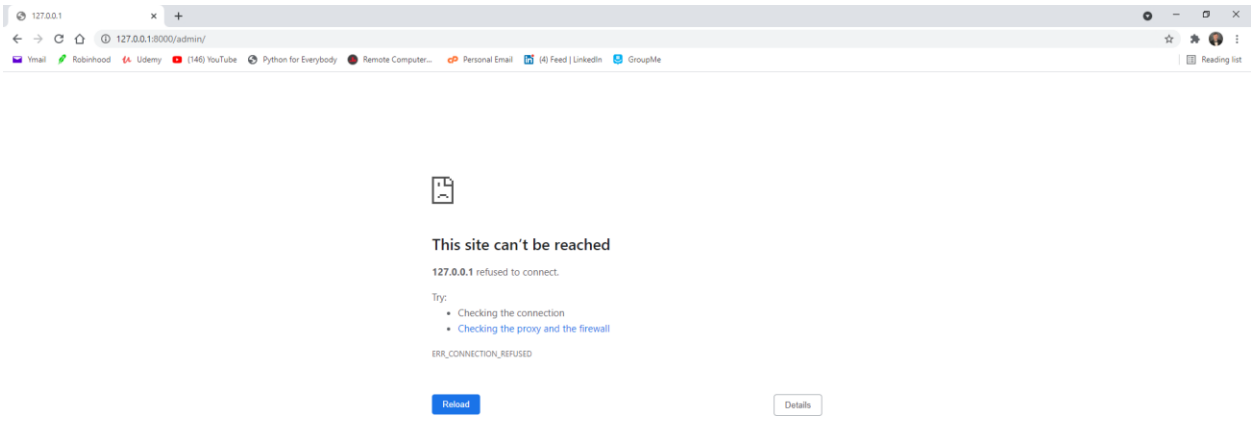


Figure 3.3.5-4 ERROR – can not log into admin panel

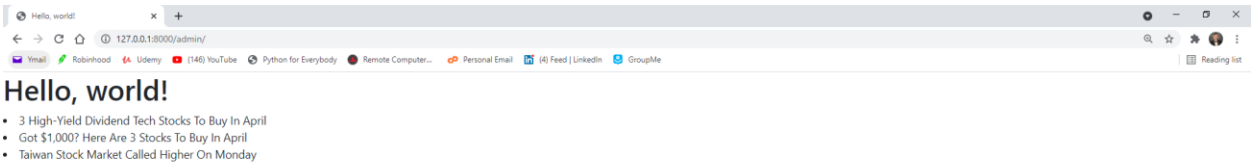


Figure 3.3.5-5 ERROR - admin URL returns same has webpage "index.html" URL returns.

I concluded that the previous two errors were raised due to not correctly defining URL paths and also because I did not create a super user account which is needed to log into the panel.

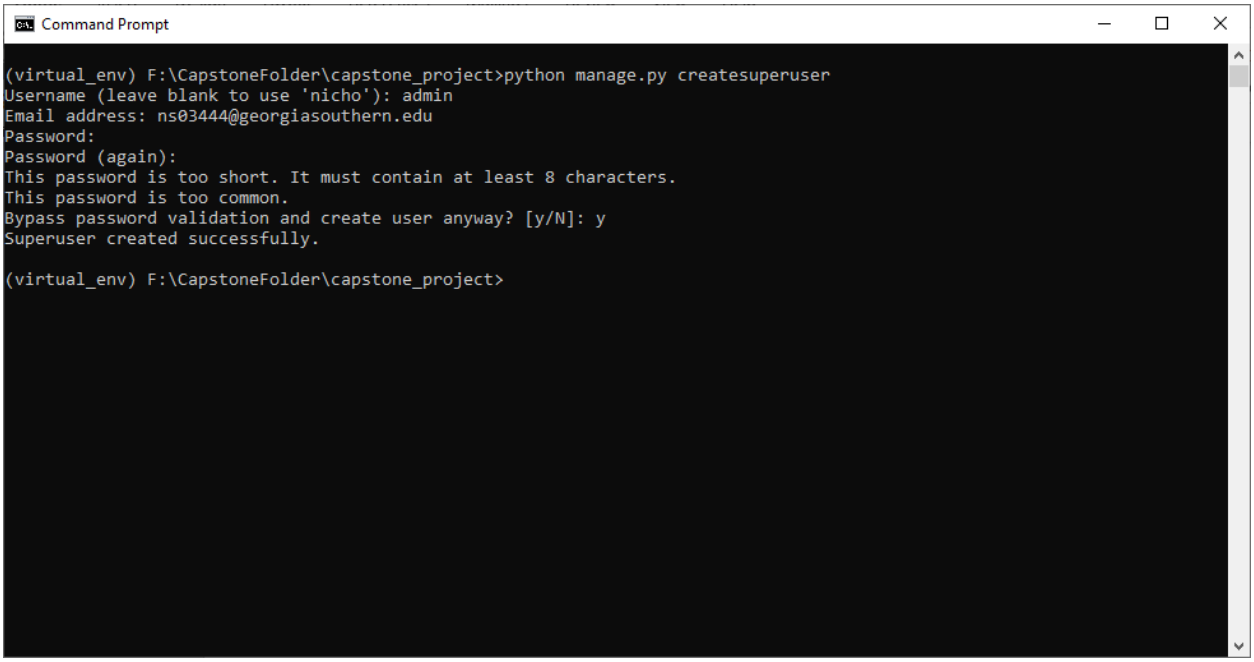


Figure 3.3.5-6 Creating super user for admin log in.

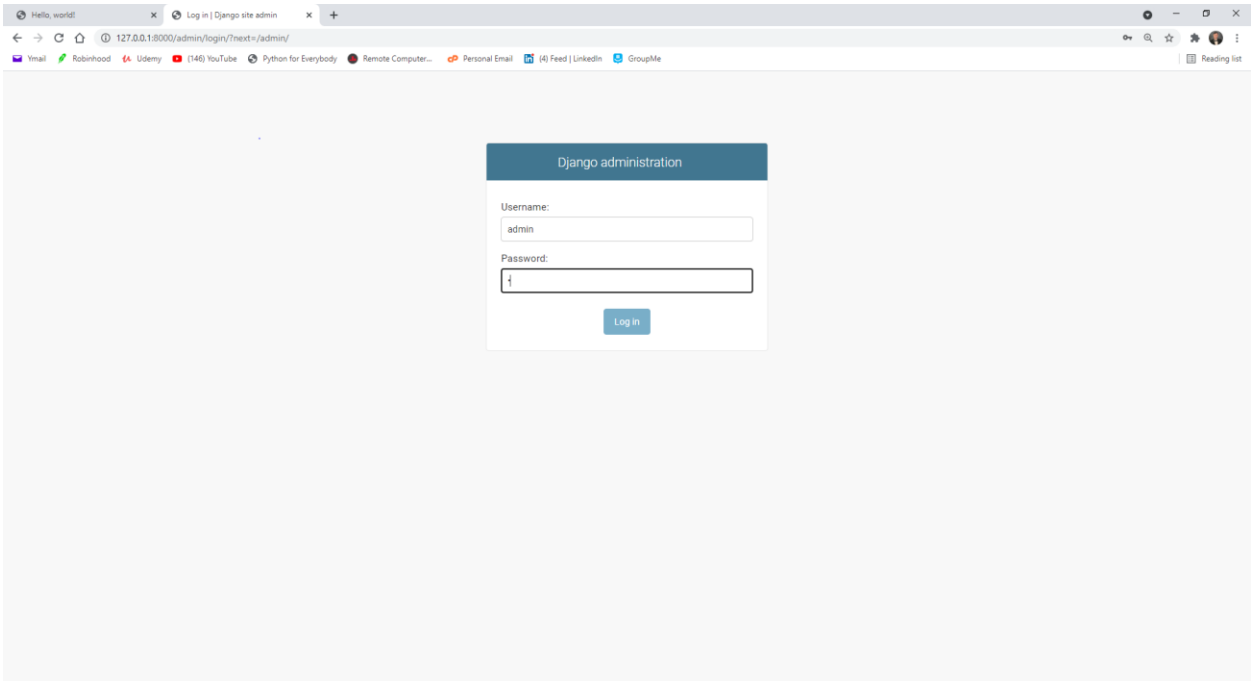


Figure 3.3.5-7 Logging into admin panel.

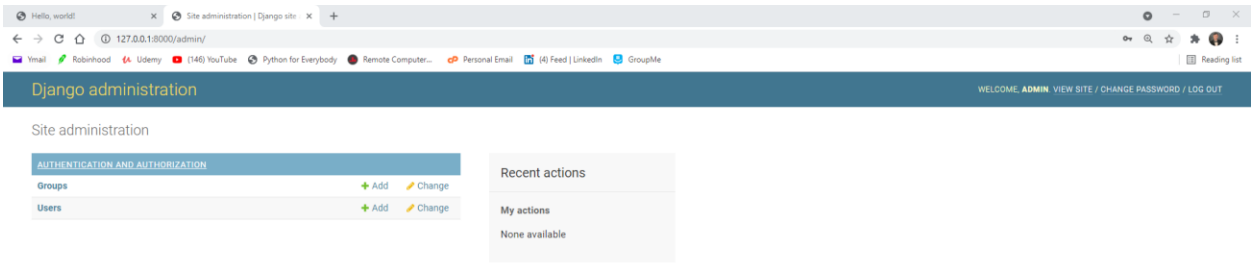


Figure 3.3.5-7 ERROR - no tables in admin panel.

After I finally managed to successfully log into the admin panel, the tables previously created were not busy. This was not expected, however, I realized I forgot to register the tables/models in the "news_app/admin.py" file.

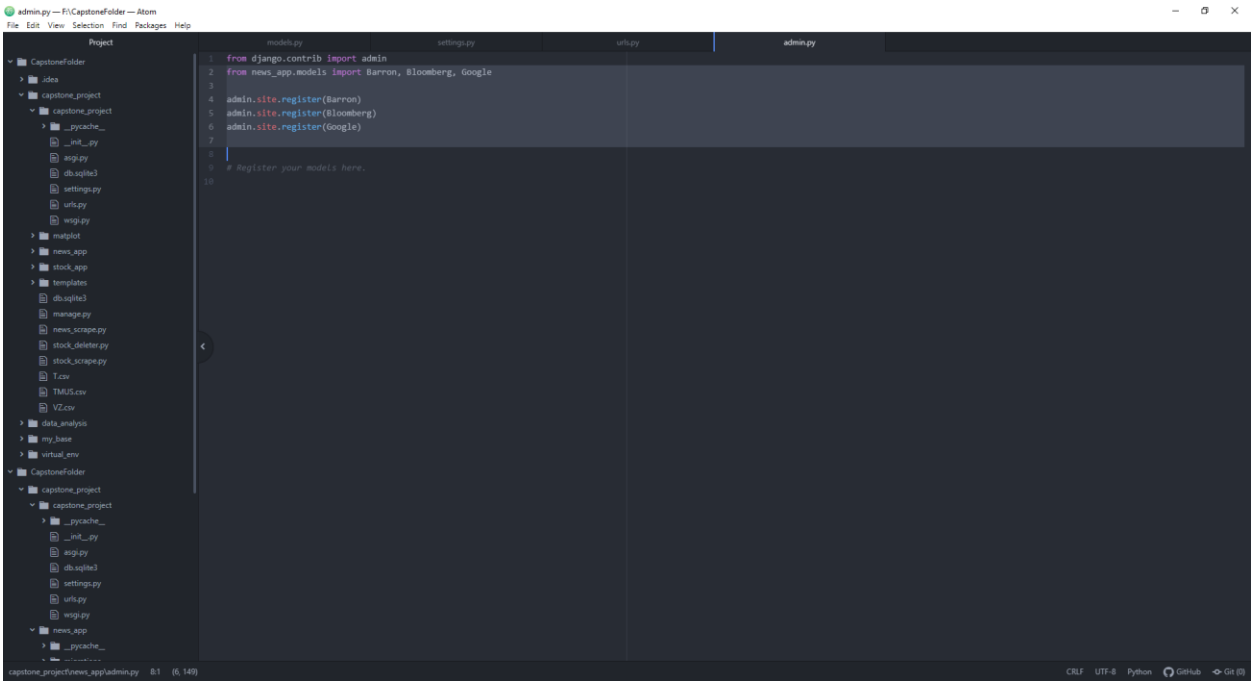


Figure 3.3.5-9 Registering models in admin.py file.

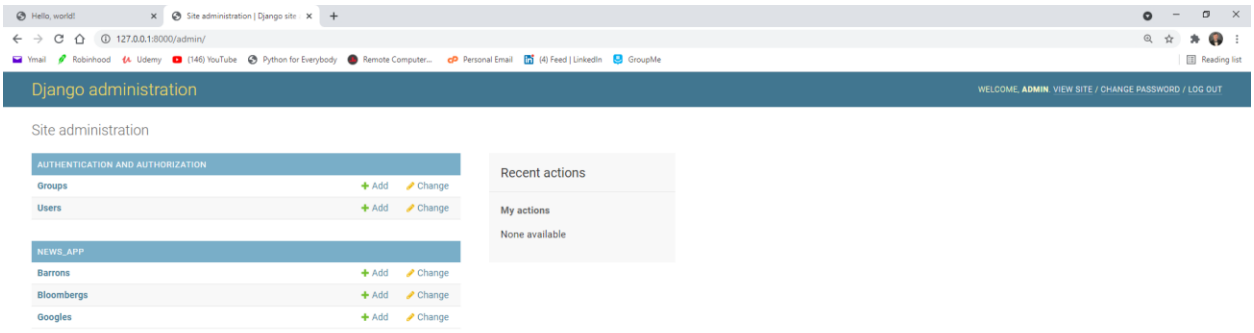


Figure 3.3.5-10 refreshed page, tables visible.

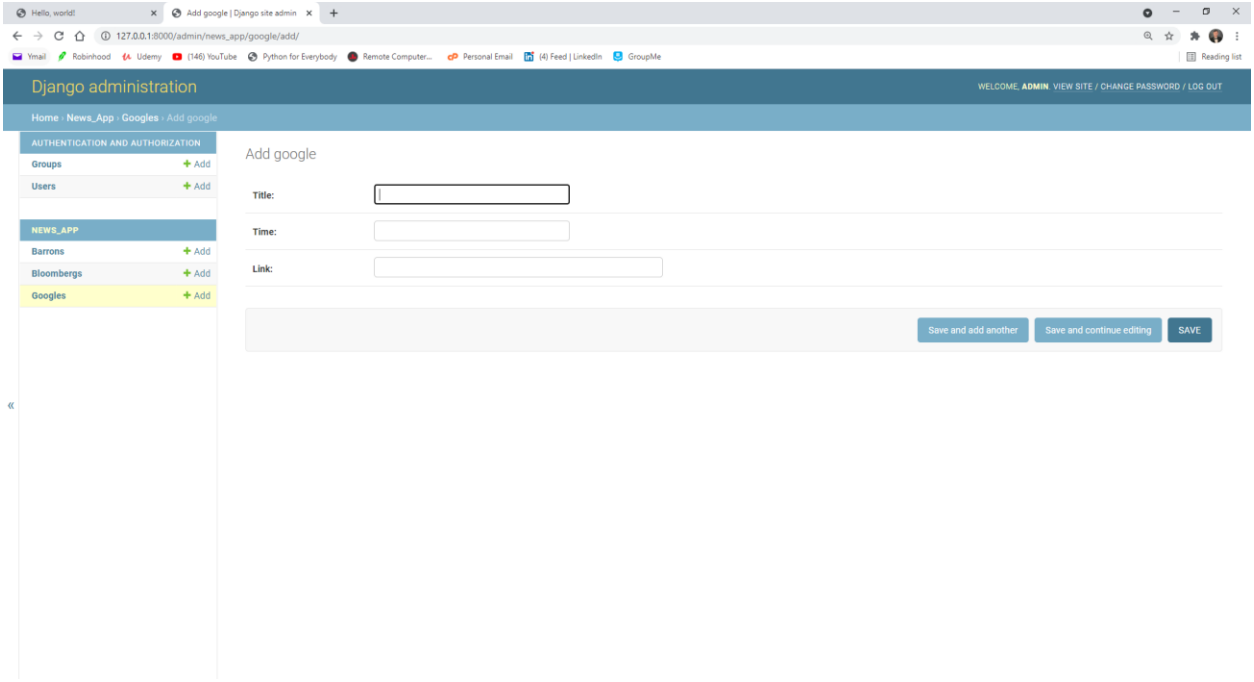
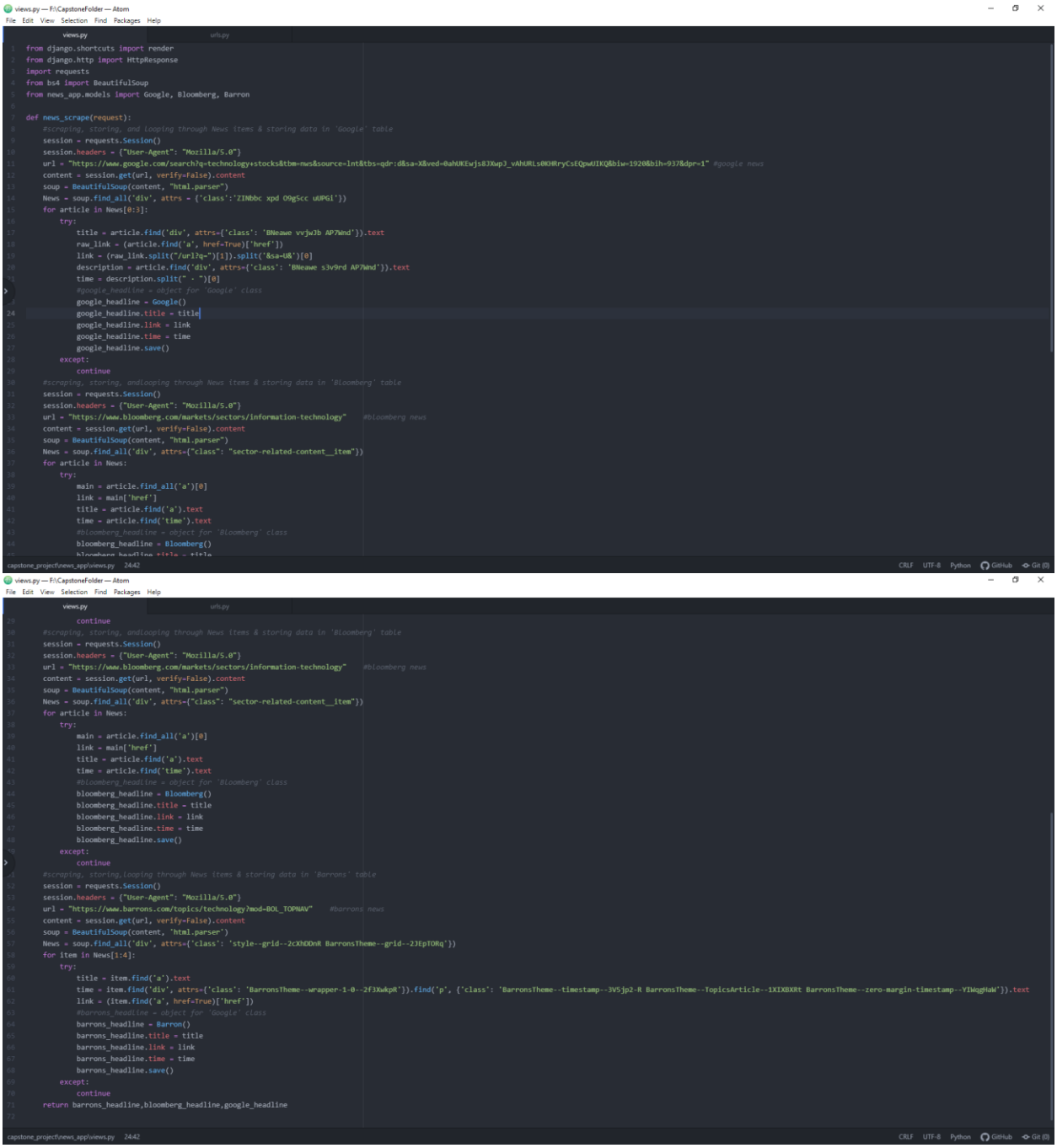


Figure 3.3.5-11 Exploring tables and insert feature through admin panel.

4.3.6 Populating the Database

Now that the tables are created, I will combine my scrape functions and store them inside a single function in the “views.py” file. Also, there will be additional code added to the end of each for loop in the function, to store the data in the appropriate tables. The next function in the “views.py” file called, “news_lst” will contain the request parameter and the context variables for which the items will be passed

to the template (index.html). However, before creating the next function, I will run the server to test my news_scrape function. If there are items in the database, then it has worked successfully.



```
from django.shortcuts import render
from django.http import HttpResponse
import requests
from bs4 import BeautifulSoup
from news_app.models import Google, Bloomberg, Barron

def news_scrape(request):
    #scraping, storing, and looping through News items & storing data in `Google` table
    session = requests.Session()
    session.headers = {"User-Agent": "Mozilla/5.0"}
    url = "https://www.google.com/search?q=technology+stocks&tbm=nws&source=lnt&tbs=qdr:d&sa=X&ved=0ahUKEwjs8JXwpJ_vAhURLs0KHRryCsEQpwUIKQ&biw=1920&bih=937&dpr=1" #google news
    content = session.get(url, verify=False).content
    soup = BeautifulSoup(content, "html.parser")
    News = soup.find_all('div', attrs = {'class':'ZINbbc xpd O9g5cc uUPGi'})
    for article in News[0:3]:
        try:
            title = article.find('div', attrs={'class': 'BNeawe vvjwJb AP7Wnd'}).text
            raw_link = (article.find('a', href=True)['href'])
            link = (raw_link.split("/url?q=")[1]).split('&sa=U&')[0]
            description = article.find('div', attrs={'class': 'BNeawe s3v9rd AP7Wnd'}).text
            time = description.split(" · ")[0]
            #google_headline = object for `Google` class
            google_headline = Google()
            google_headline.title = title
            google_headline.link = link
            google_headline.time = time
            google_headline.save()
        except:
            continue
    #scraping, storing, andLooping through News items & storing data in `Bloomberg` table
    session = requests.Session()
    session.headers = {"User-Agent": "Mozilla/5.0"}
    url = "https://www.bloomberg.com/markets/sectors/information-technology"    #bloomberg news
    content = session.get(url, verify=False).content
    soup = BeautifulSoup(content, "html.parser")
    News = soup.find_all('div', attrs={"class": "sector-related-content__item"})
    for article in News:
        try:
            main = article.find_all('a')[0]
            link = main['href']
            title = article.find('a').text
            time = article.find('time').text
            #bloomberg_headline = object for `Bloomberg` class
            bloomberg_headline = Bloomberg()
            bloomberg_headline.title = title
            bloomberg_headline.link = link
            bloomberg_headline.time = time
            bloomberg_headline.save()
        except:
            continue
    #scraping,storing,looping through News items & storing data in `Barrons` table
    session = requests.Session()
    session.headers = {"User-Agent": "Mozilla/5.0"}
    url = "https://www.barrons.com/topics/technology?mod=BOL_TOPNAV"    #barrons news
    content = session.get(url, verify=False).content
    soup = BeautifulSoup(content, 'html.parser')
    News = soup.find_all('div', attrs={'class': 'style--grid--2cXhDDnR BarronsTheme--grid--2JEpTORq'})
    for item in News[1:4]:
        try:
            title = item.find('a').text
            time = item.find('div', attrs={'class': 'BarronsTheme--wrapper-1-0--2f3XwkpR'}).find('p', {'class': 'BarronsTheme--timestamp--3V5jp2-R BarronsTheme--TopicsArticle--1XIXBXRt BarronsTheme--zero-margin-timestamp--YIWqgHaW'}).text
            link = (item.find('a', href=True)['href'])
            #barrons_headline = object for `Google` class
            barrons_headline = Barron()
            barrons_headline.title = title
            barrons_headline.link = link
            barrons_headline.time = time
            barrons_headline.save()
        except:
            continue
    return barrons_headline,bloomberg_headline,google_headline
```

Figure 3.3.6-1 - storing scrape functions inside news_scrape().

After running the server, I received the following server errors, as well as some unusual functionality with the database tables. After logging into the admin panel, I realized each table had data inserted an uneven number of times, as well, as storing identical items. I believe something is not working properly with the call of my “news_scrape” function. Furthermore, just moments later I noticed that number of items in each table is increasing as the server continues running.

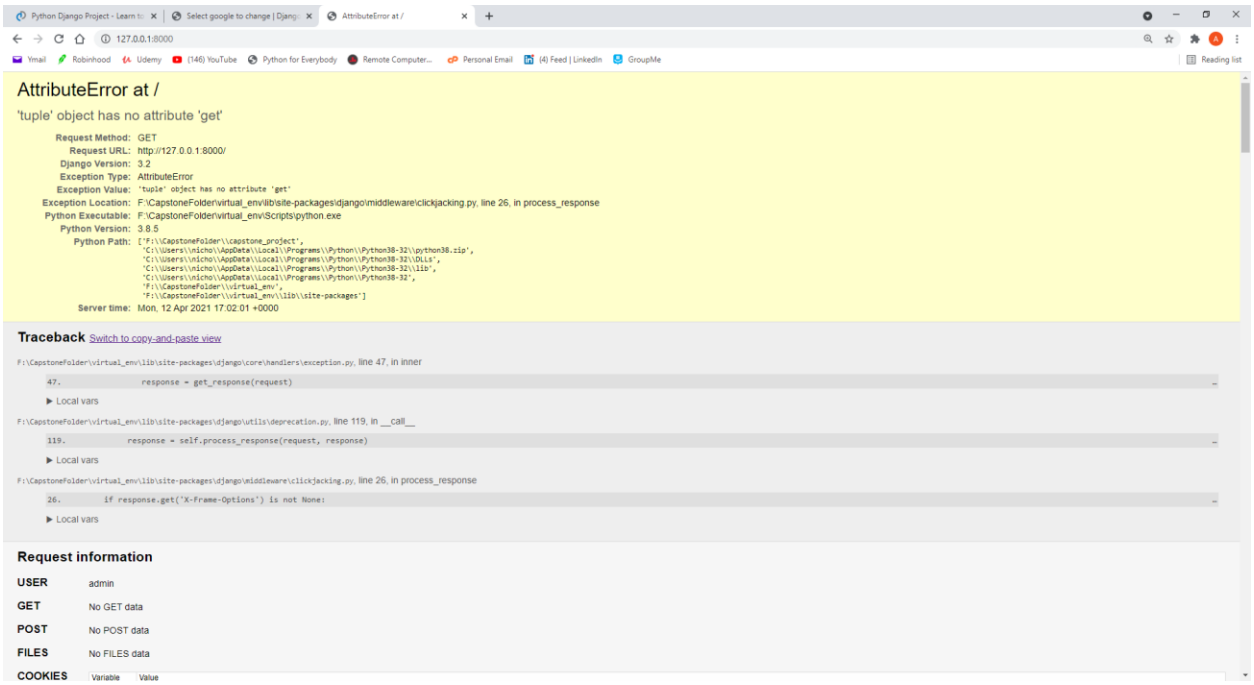


Figure 3.3.6-2 ERROR - 'tuple object has no get attribute'.

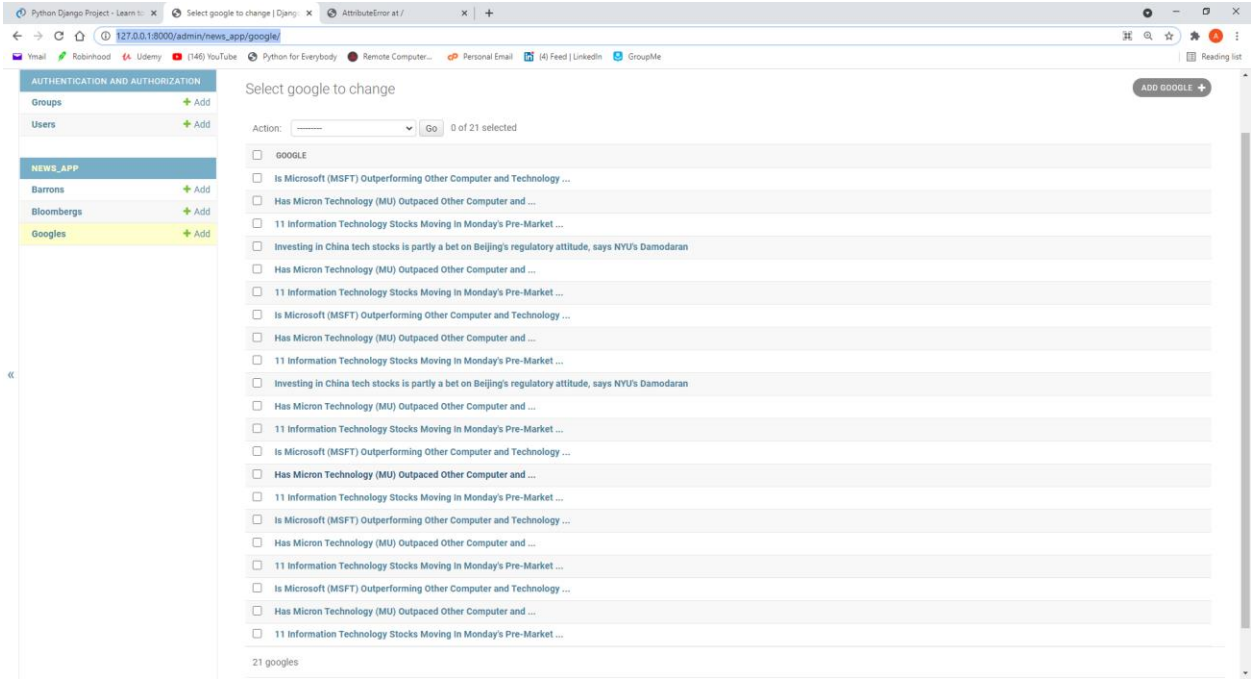


Figure 3.3.6-3 google table with 21 items.

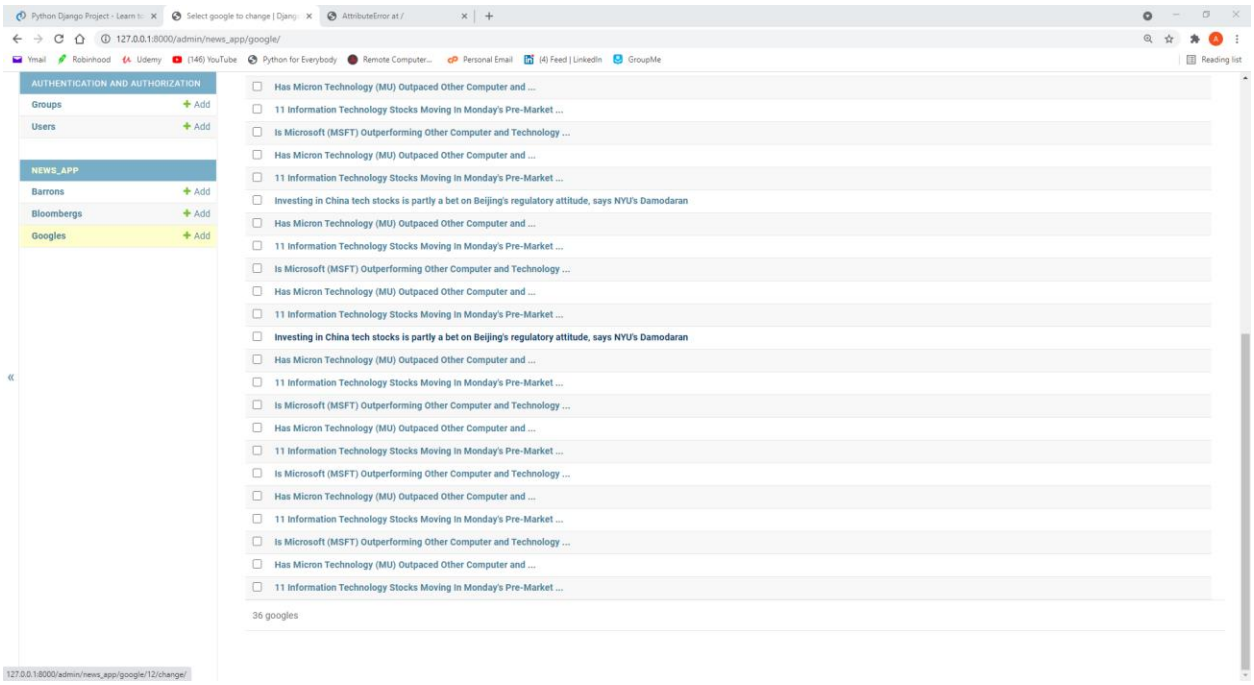


Figure 3.3.6-4 Google table seconds later with 36 items

Luckily, I can use the Django's admin feature to clear all the items from the database.

Next, I will create the "news_list" function in the "views.py" file and try passing some of the items to the template using context processors and hopefully it clears up some issues with the first function.

```
74
75  def news_list(request):
76      barrons_headline = Barron.objects.all()
77      barrons_headline = Bloomberg.objects.all()
78      google_headline = Google.objects.all()
79      context = {
80          'barrons_list': barrons_headline,
81          'bloomberg_list': bloomberg_headline,
82          'google_list': google_headline,
83      }
84      return render(request, "news/home.html", context)
85
```

Figure 3.3.6-5 Creating the "news_list()" function in the view.

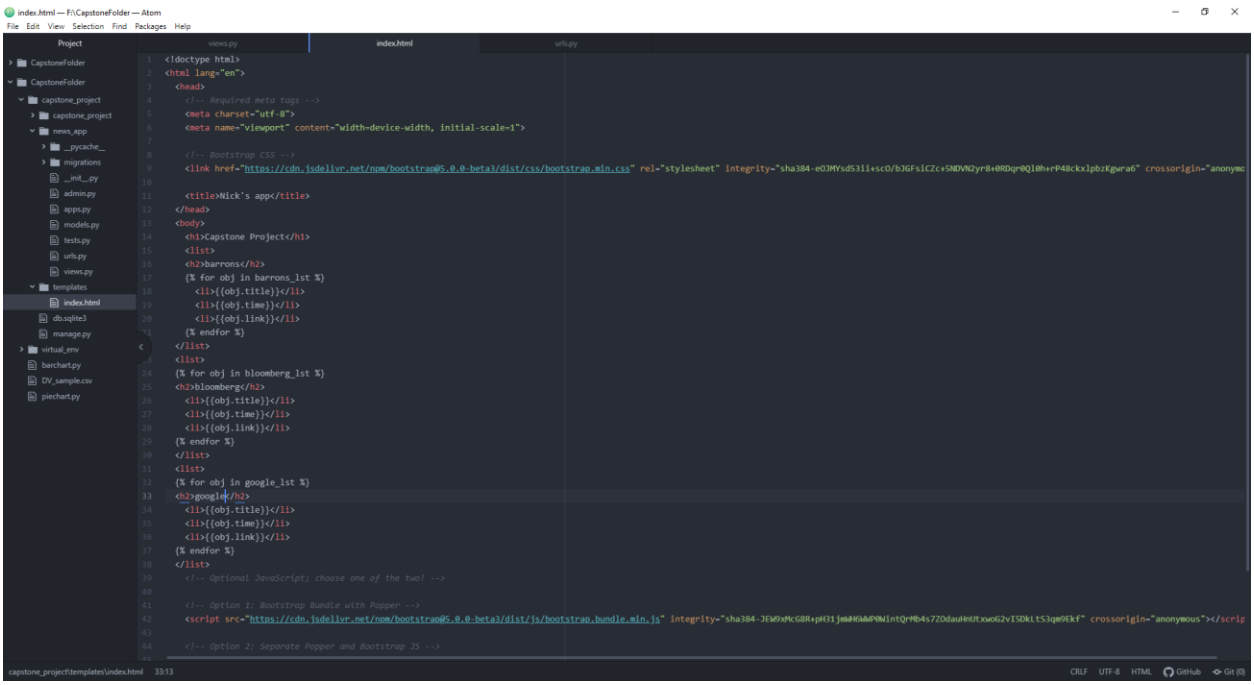


Figure 3.3.6-6 Injecting article items in template using context processors

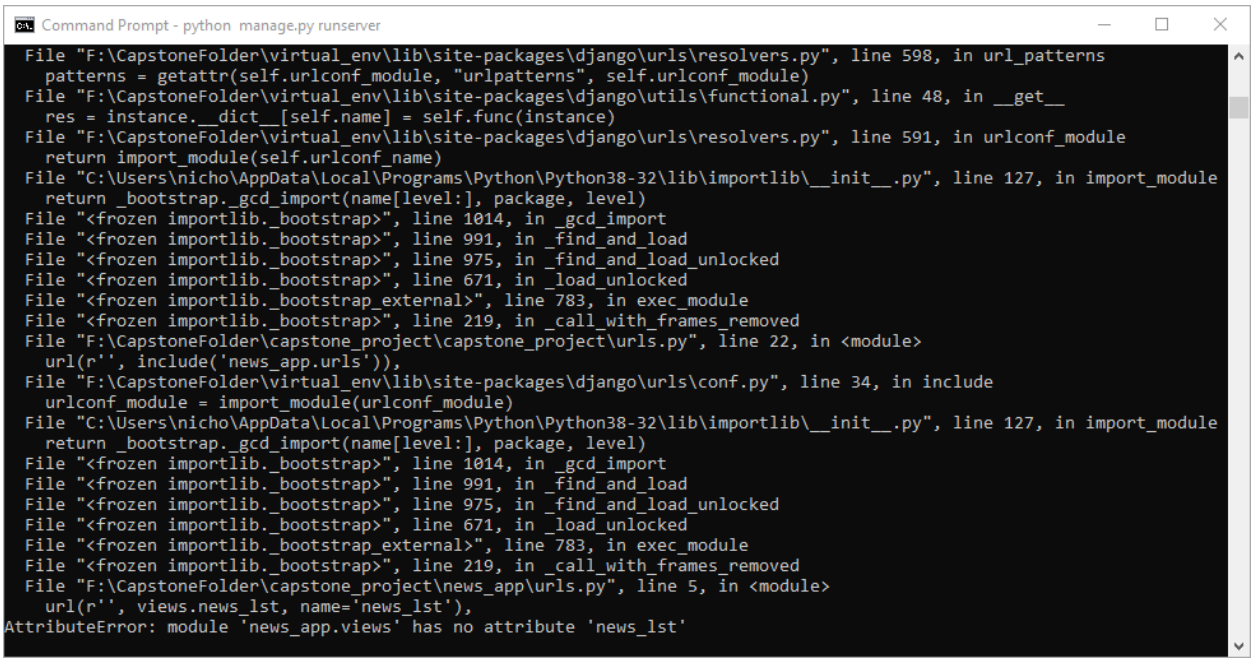


Figure 3.3.6-7 ERROR – typo: misspelled function name in urls.py file

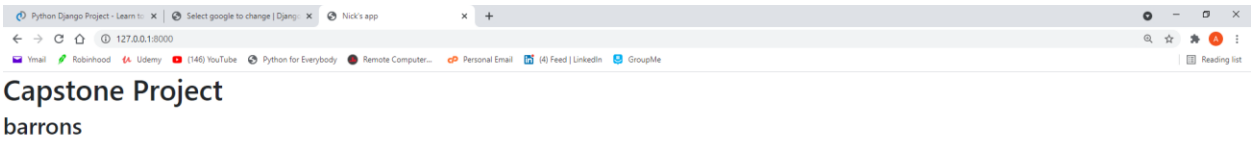


Figure 3.3.6-8 Template rendered in browser, no items displayed.

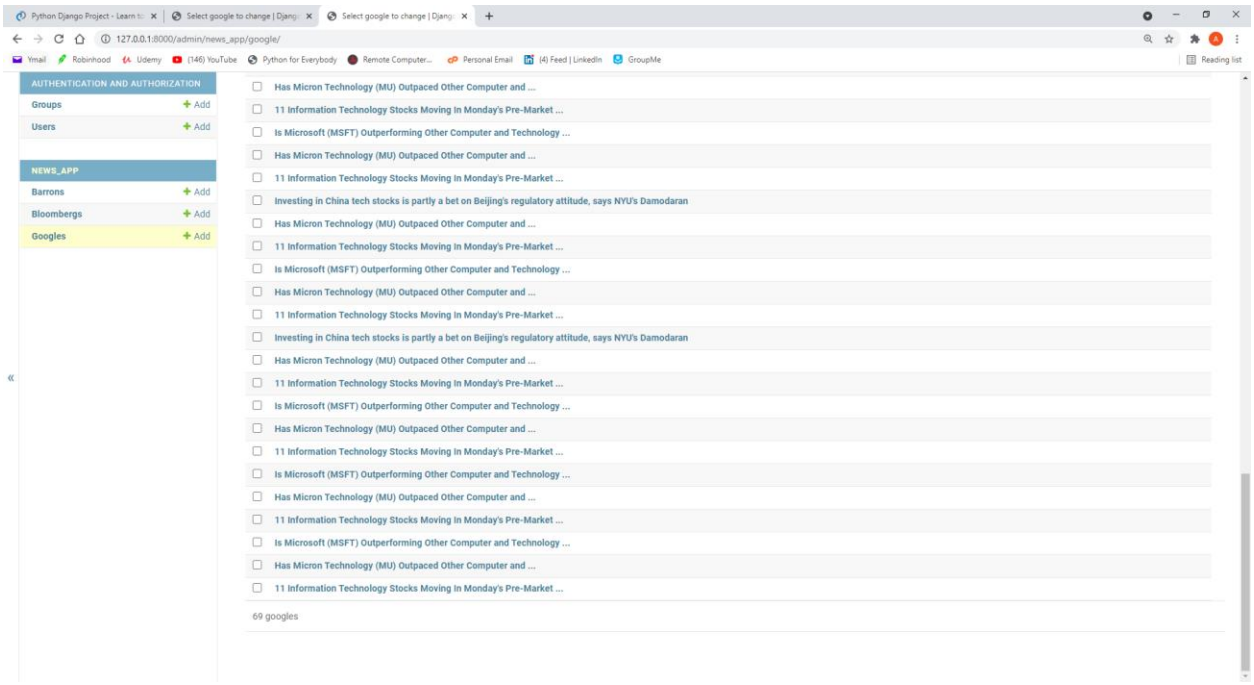


Figure 3.3.6-9 database continuing to populate while server is running.

I concluded that this must be happening because of Django's file directory structure and that my "views.py" file is calling the "news_scrape" function every time there is an http request in the back end. Therefore, I must find a different process for populating my database.

4.3.7 A different approach to populating the database.

After conducting some research to solve the issue I had been faced with, I realized the issue may be starting in my "views.py" file. Since this file is being ran (requested) while the server is running, I found that you can create a new python file inside the Django project directory (outside of application directory) to contain the “news_scrape” function along with some additional dependencies. This file runs as a normal python file inside the command prompt (inside the virtual environment) and is used to populate the database. The if statement at the end of the file is used to tell the python interpreter to check the module’s source code to ensure that this is the main program to run.

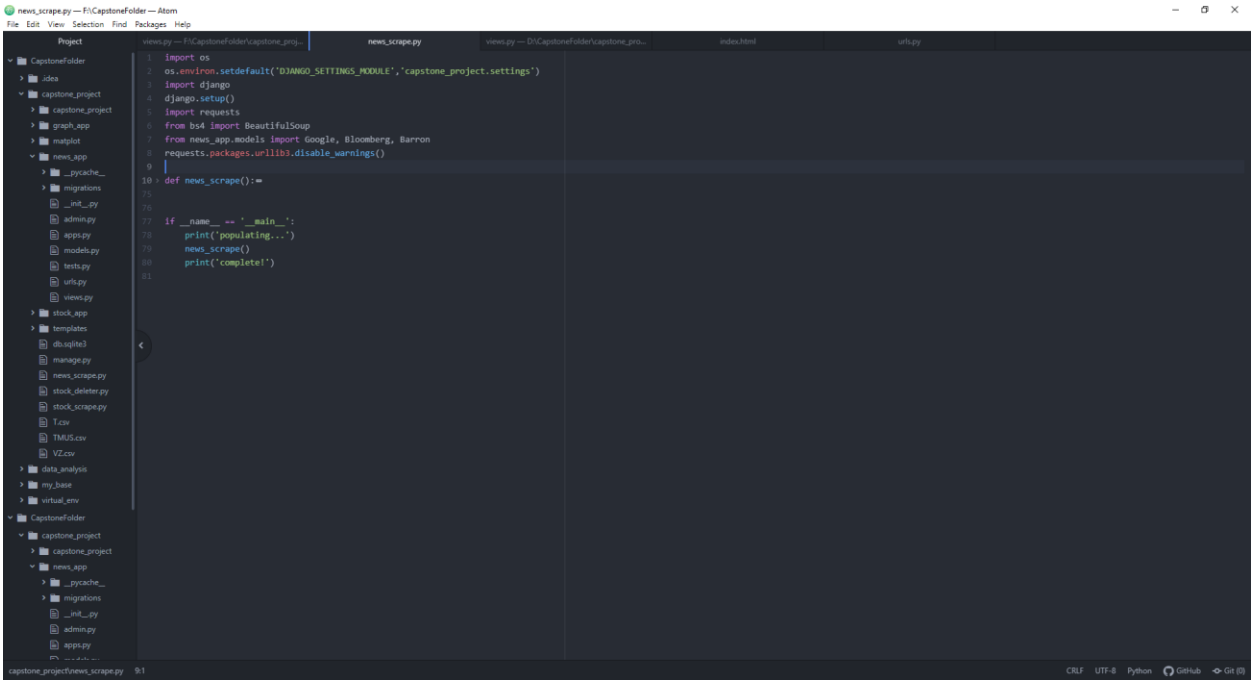


Figure 3.3.7-1 Creating the new file named “news_scrape.py” to populate database.

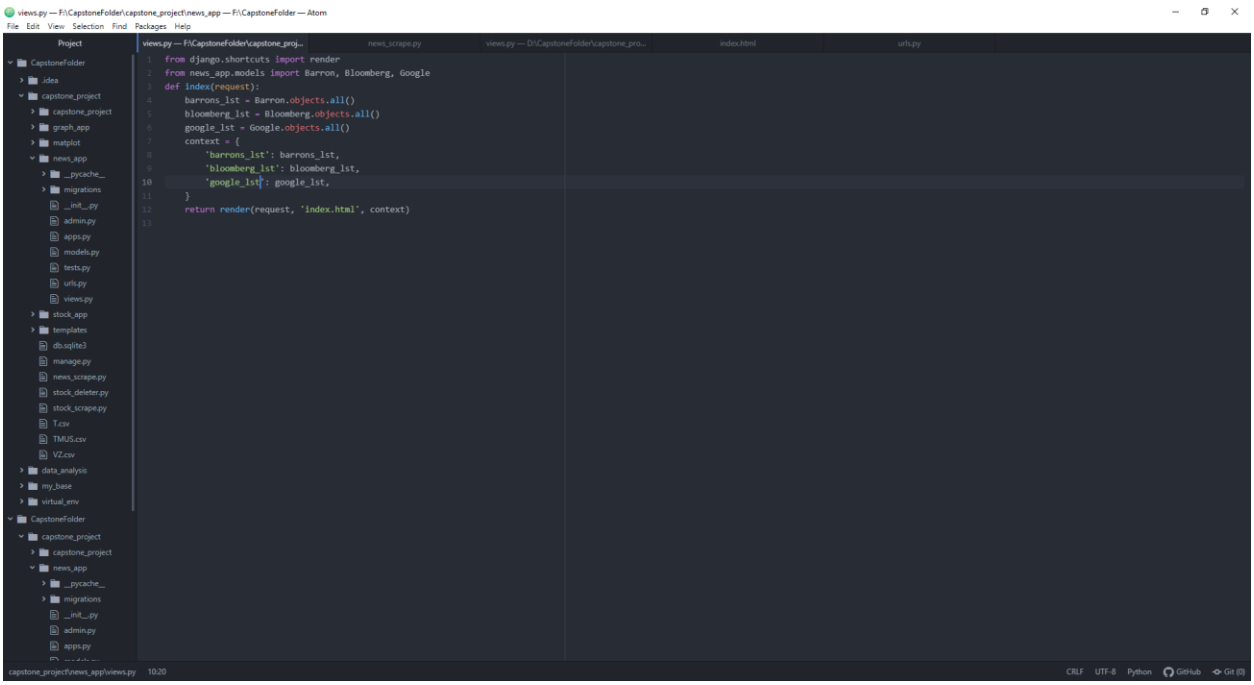


Figure 3.3.7-2 Editing and renaming function in views.py as "index()".

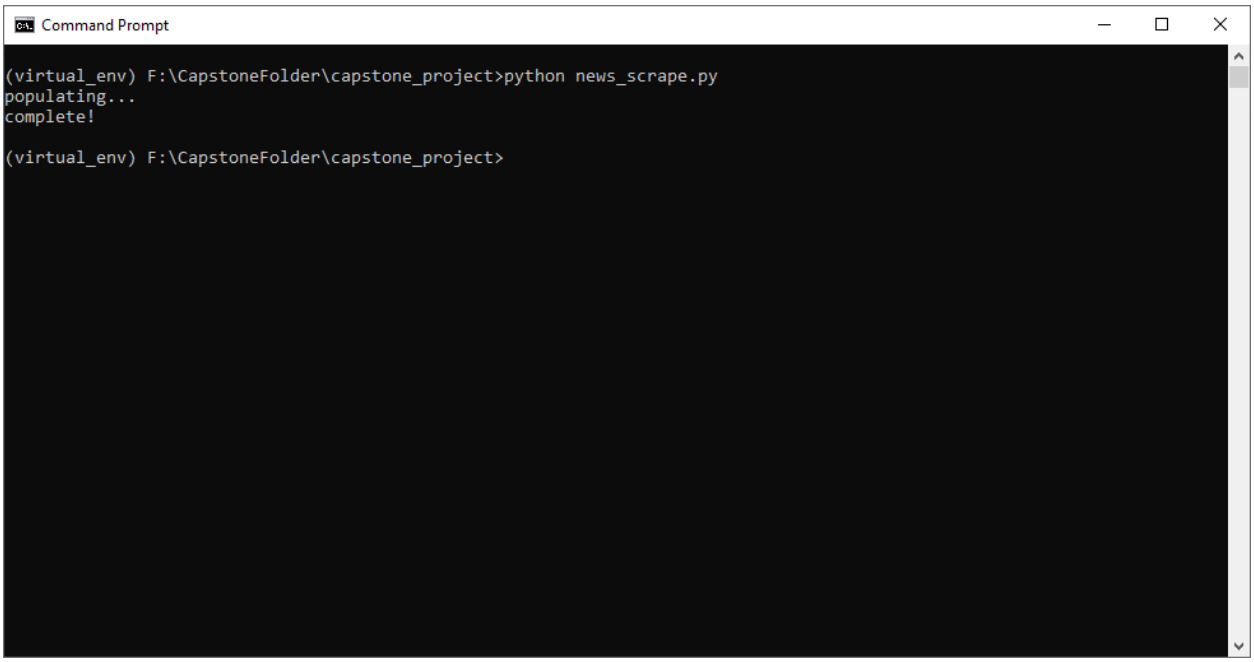


Figure 3.3.7-3 Successfully running "news_scrape.py".

Now that my new script has ran successfully without any errors, it is time to log back into the admin panel and check the tables, then check the browser for "index.html" containing the new items.

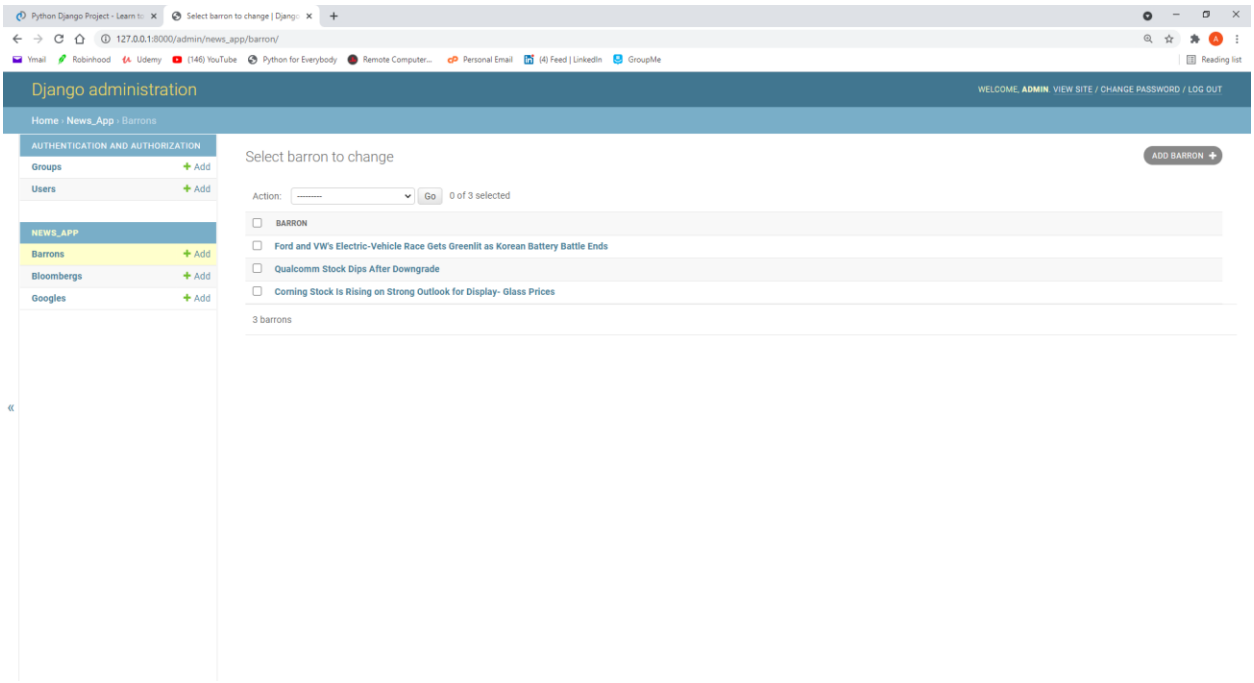


Figure 3.3.7-4 Populating Barron table.

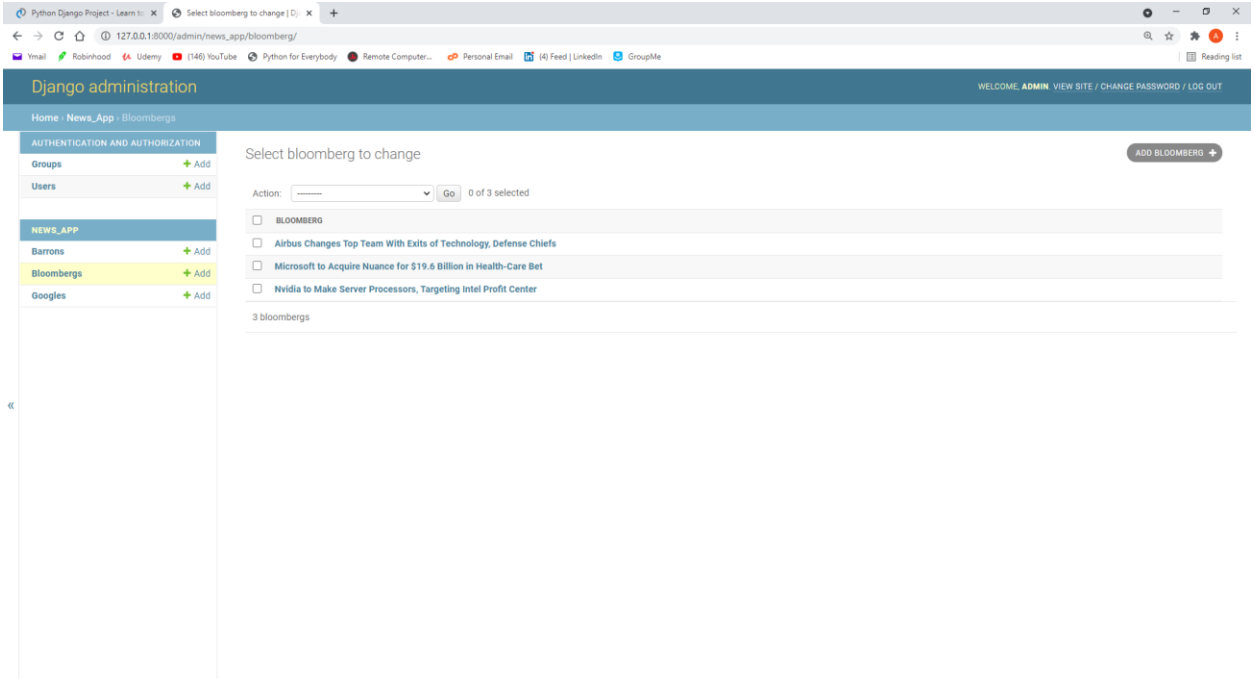


Figure 3.3.7-5 Populating Bloomberg table

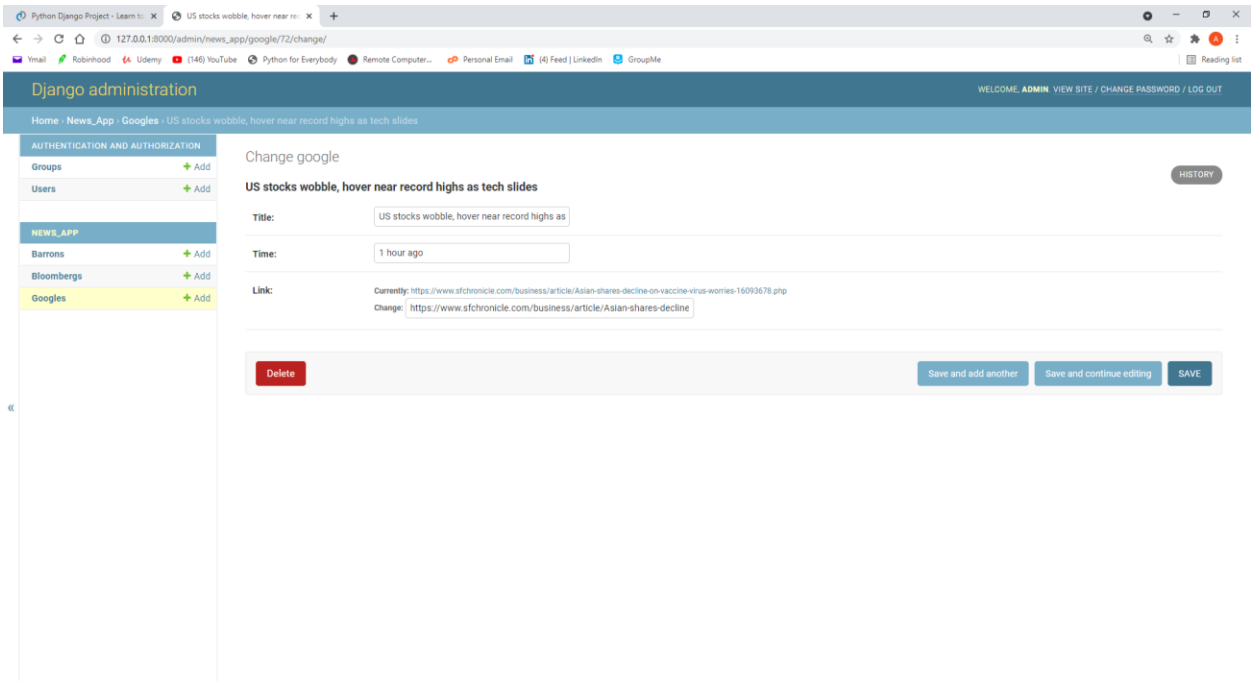


Figure 3.3.7-6 Populating Google table

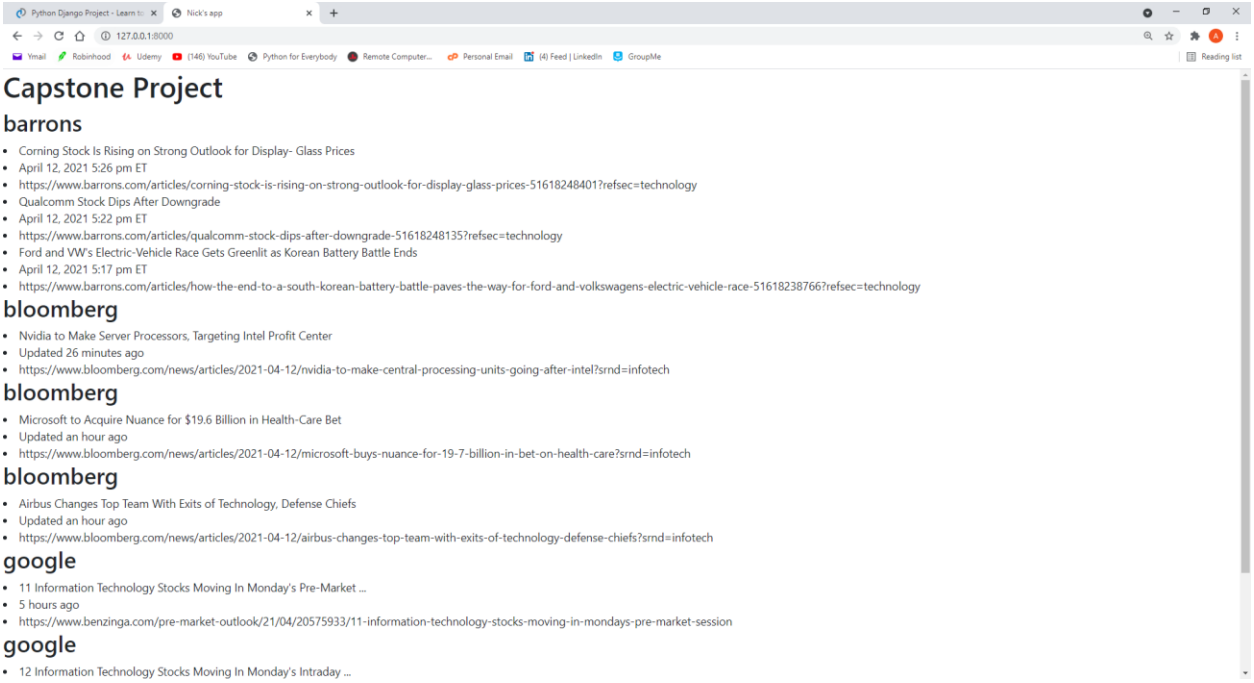


Figure 3.3.7-7 News items displayed in browser, unformatted.

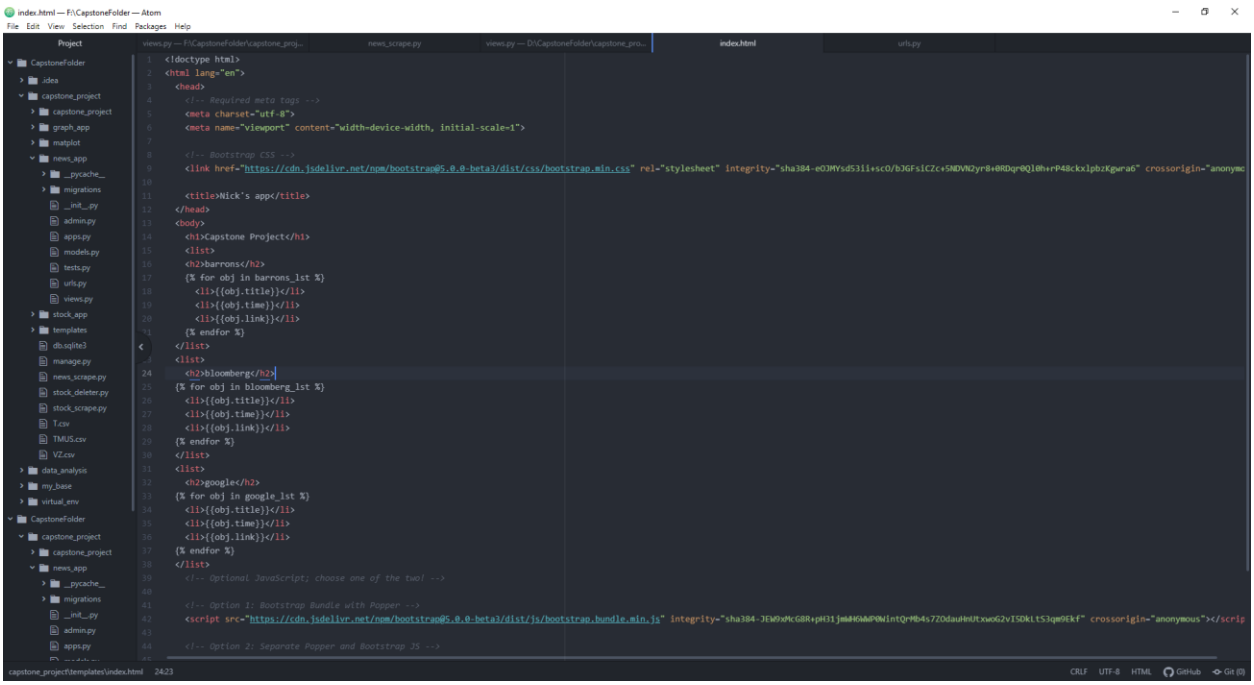


Figure 3.3.7-8 Moved titles outside of for loop to enhance formatting.

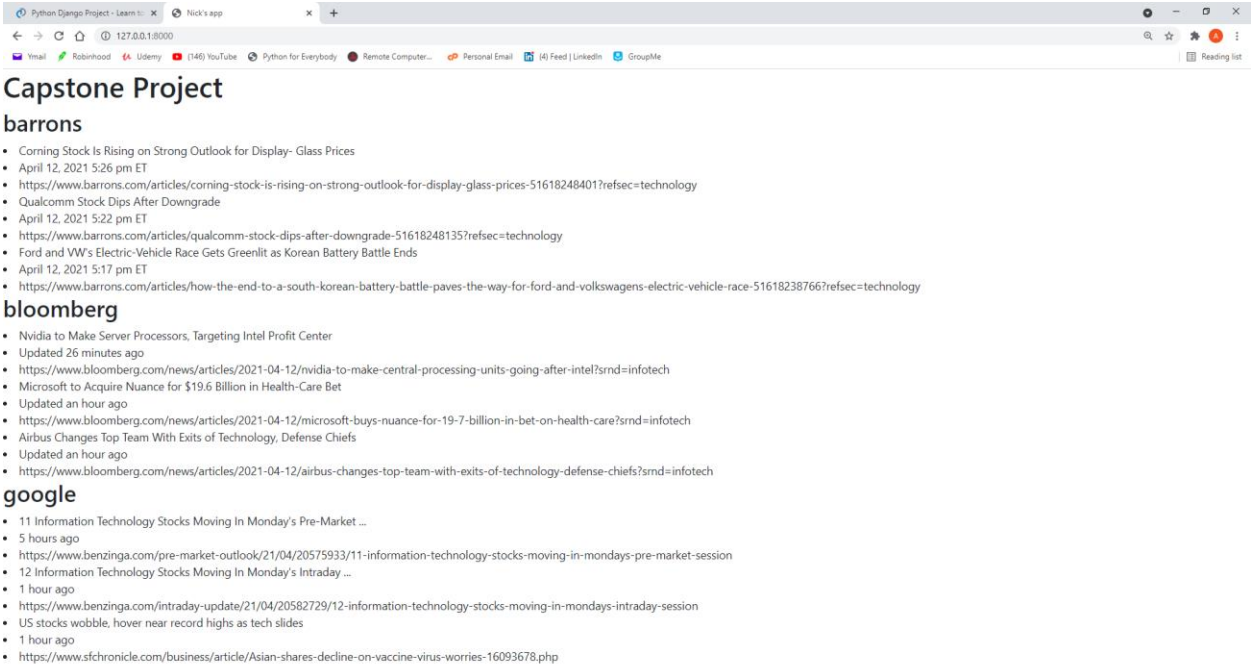


*Figure 3.3.7-9 **MILESTONE ACHIEVED – RENDERED ALL ITEMS TO TEMPLATE IN SEPARATE LISTS FROM DATABASE.*

4.4 Designing the News page

Now that the Django project has been created and the model, view, template system is properly functioning, I still need to make some front-end adjustments to make the webpage more presentable. Although I was fortunate enough to find an example template very similar to my need, I will have to do some digging in the bootstrap documentation to make all the adjustments needed.

4.4.1 Formatting the home page

Now that I have properly rendered the database objects to the template in the correct logic, it is time to explore bootstrap's excellent documentation to add some design features to the homepage. Luckily, without having to look very long, I found a bootstrap template specifically used for news articles being web scraped from a Django tutorial on the Data Flair website (Link).

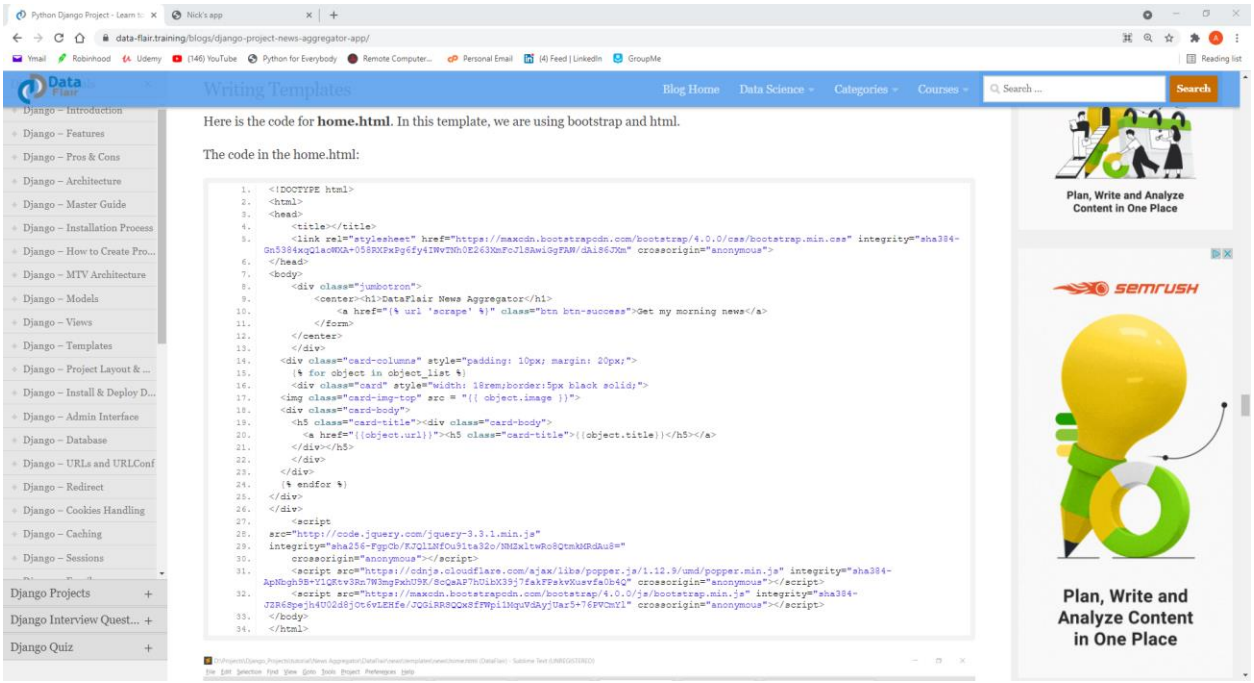


Figure 3.4.1-1 Data flair tutorial template.

After coping and pasting the html code shown above, I will edit the context processors to match the variables I created previously.

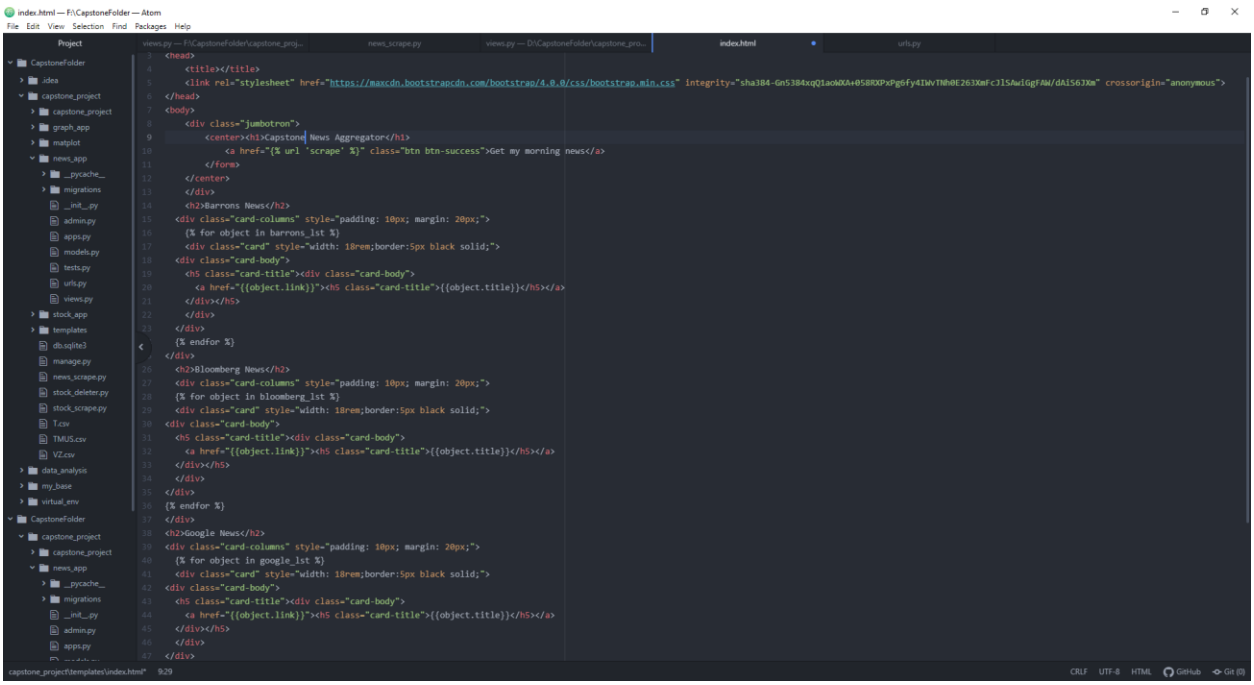


Figure 3.4.1-2 Editing the data flair template for my context processors

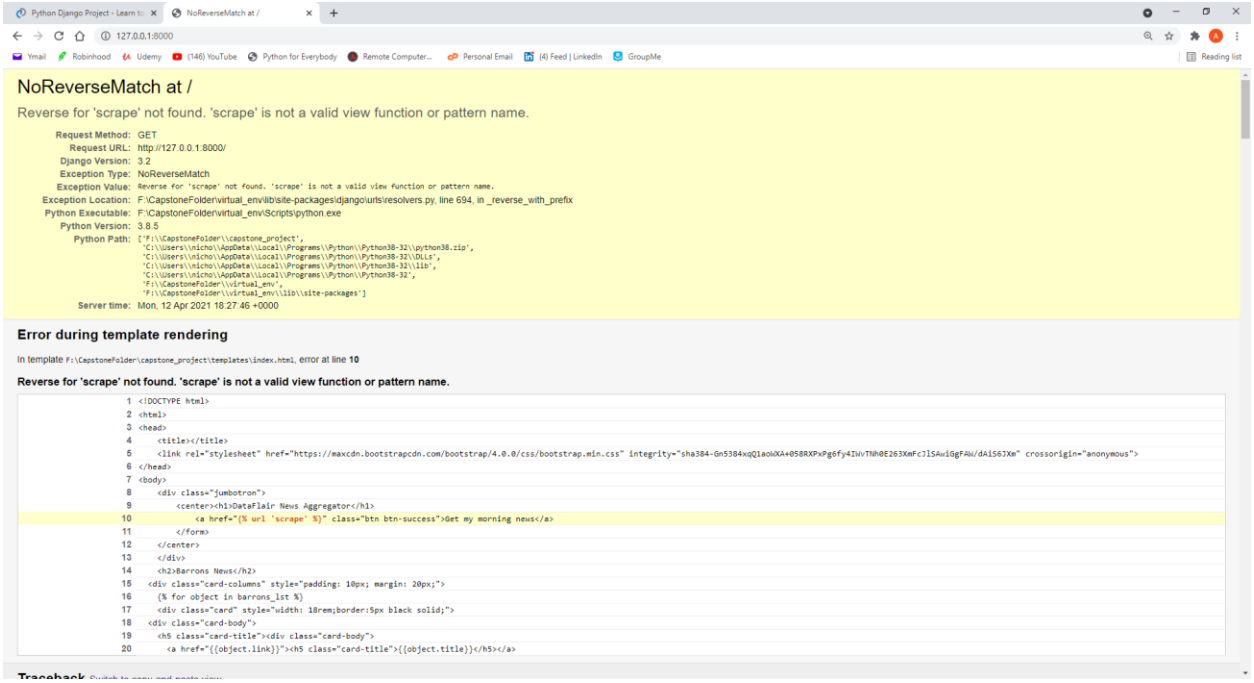


Figure 3.4.1-3 ERROR – did not make all necessary changes to data flair template

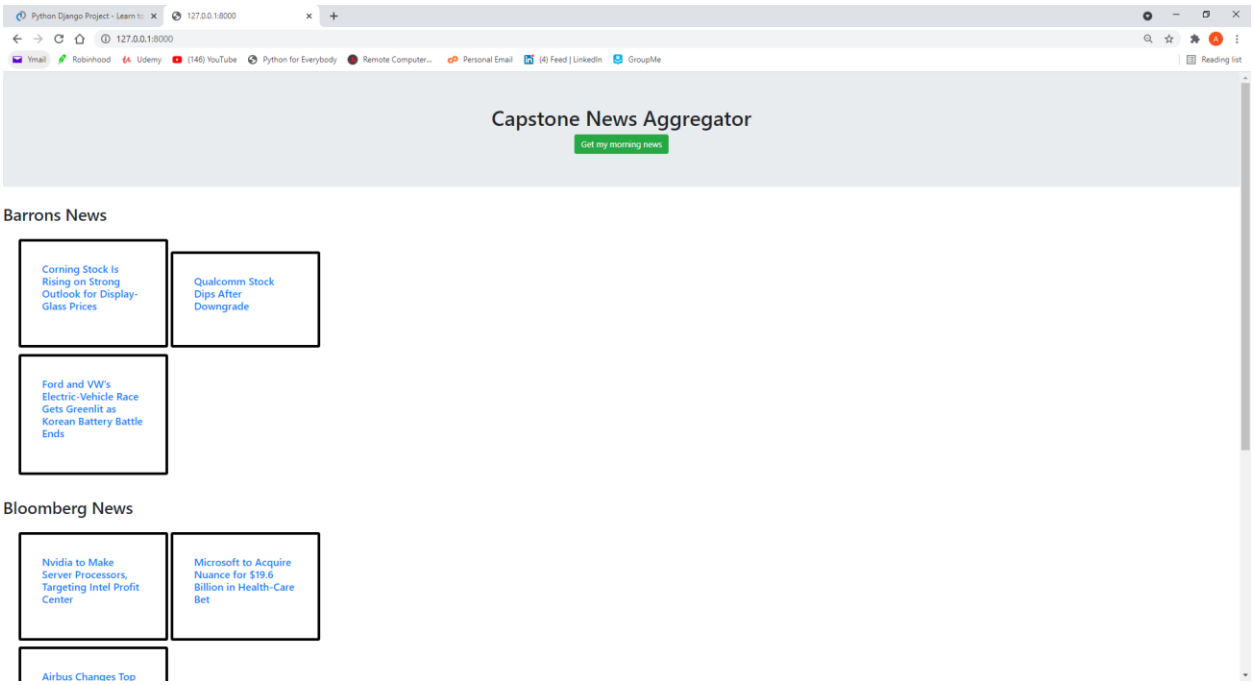
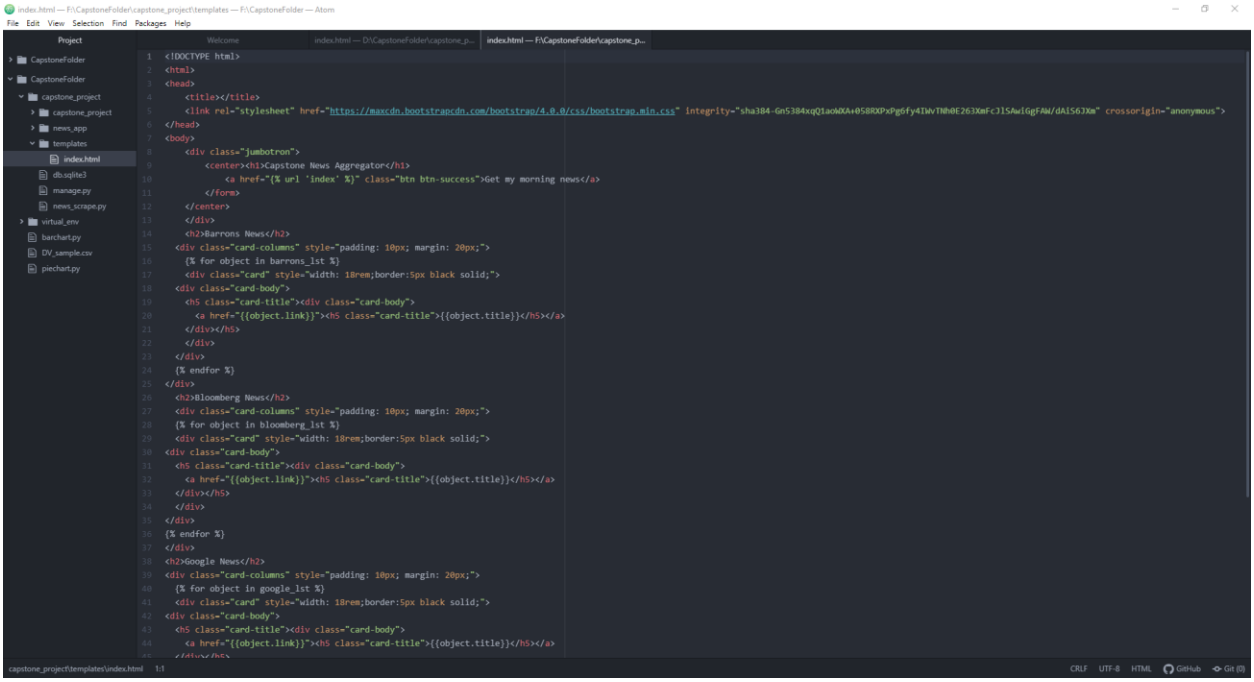


Figure3.4.1-4 items returned with new features (cards, push button, header, etc.)

The next process just consists of exploring the bootstrap documentation to make small front-end adjustments and customizations to display the content to my personal pleasing.

4.4.2 Aligning page content with Bootstrap

To begin the process, I will focus on the main elements that I want to manipulate and search Bootstrap's documentation related to those elements. The first element or group of elements that I want to focus on are the "card" elements containing the article titles. Furthermore, since the template I used from Data Flair used "card-columns" I will first attempt to display the boxes horizontally in the middle of the webpage.



```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
</head>
<body>
    <div class="jumbotron">
        <center><h1>Capstone News Aggregator</h1>
            <a href="{% url 'index' %}" class="btn btn-success">Get my morning news</a>
        </form>
    </center>
    </div>
    <h2>Barrons News</h2>
  <div class="card-columns" style="padding: 10px; margin: 20px;">
    {% for object in barrons_lst %}
    <div class="card" style="width: 18rem;border:5px black solid;">
  <div class="card-body">
    <h5 class="card-title"><div class="card-body">
      <a href="{{object.link}}"><h5 class="card-title">{{object.title}}</h5></a>
    </div></h5>
    </div>
  </div>
  {% endfor %}
</div>
  <h2>Bloomberg News</h2>
  <div class="card-columns" style="padding: 10px; margin: 20px;">
  {% for object in bloomberg_lst %}
  <div class="card" style="width: 18rem;border:5px black solid;">
<div class="card-body">
  <h5 class="card-title"><div class="card-body">
    <a href="{{object.link}}"><h5 class="card-title">{{object.title}}</h5></a>
  </div></h5>
  </div>
</div>
{% endfor %}
</div>
<h2>Google News</h2>
<div class="card-columns" style="padding: 10px; margin: 20px;">
  {% for object in google_lst %}
  <div class="card" style="width: 18rem;border:5px black solid;">
<div class="card-body">
  <h5 class="card-title"><div class="card-body">
    <a href="{{object.link}}"><h5 class="card-title">{{object.title}}</h5></a>
```

Figure 3.4.2-1 "index.html" using card columns.

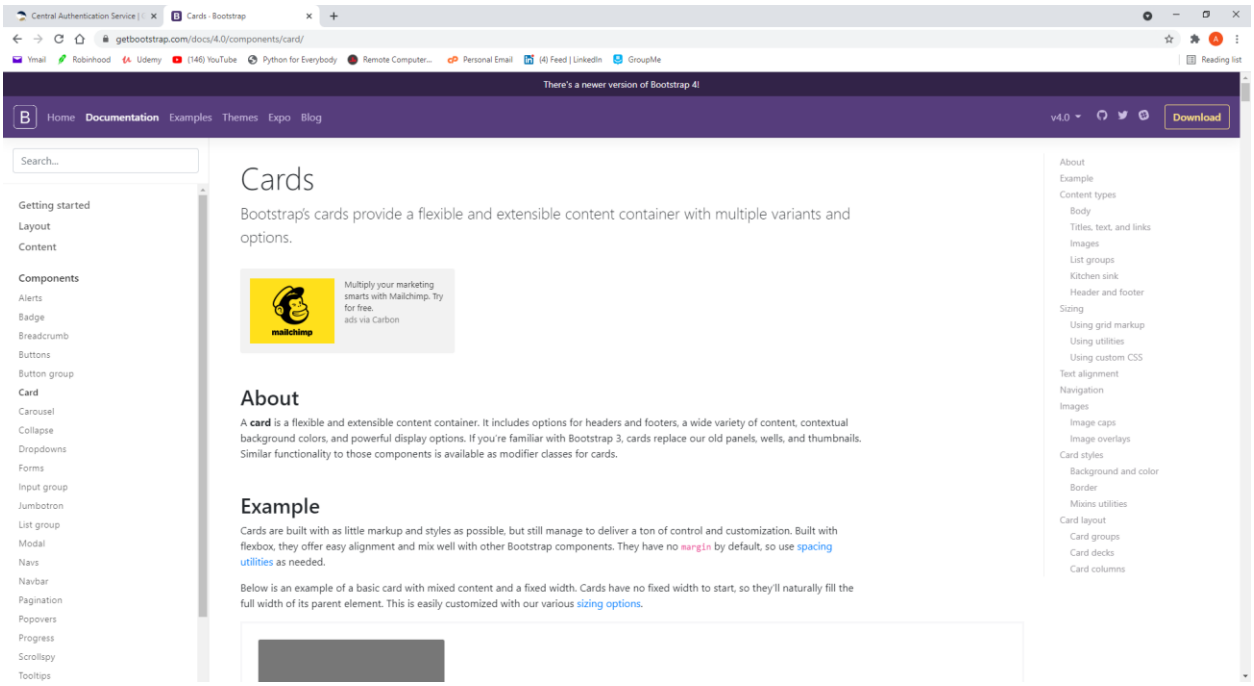


Figure 3.4.2-2 Bootstrap card documentation.

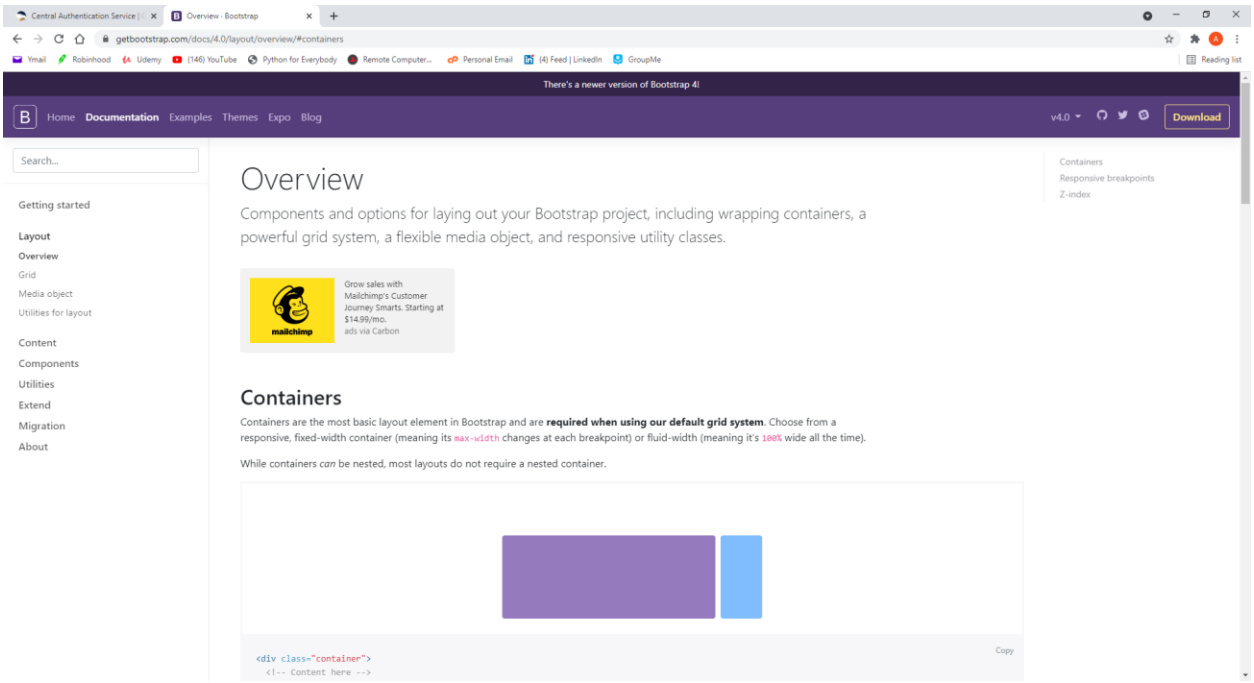


Figure 3.4.2-4 Bootstrap containers documentation.

By changing the card-column "div" elements to a row and container div elements, the cards are moved towards the center of the page in a horizontal manner, shown below.

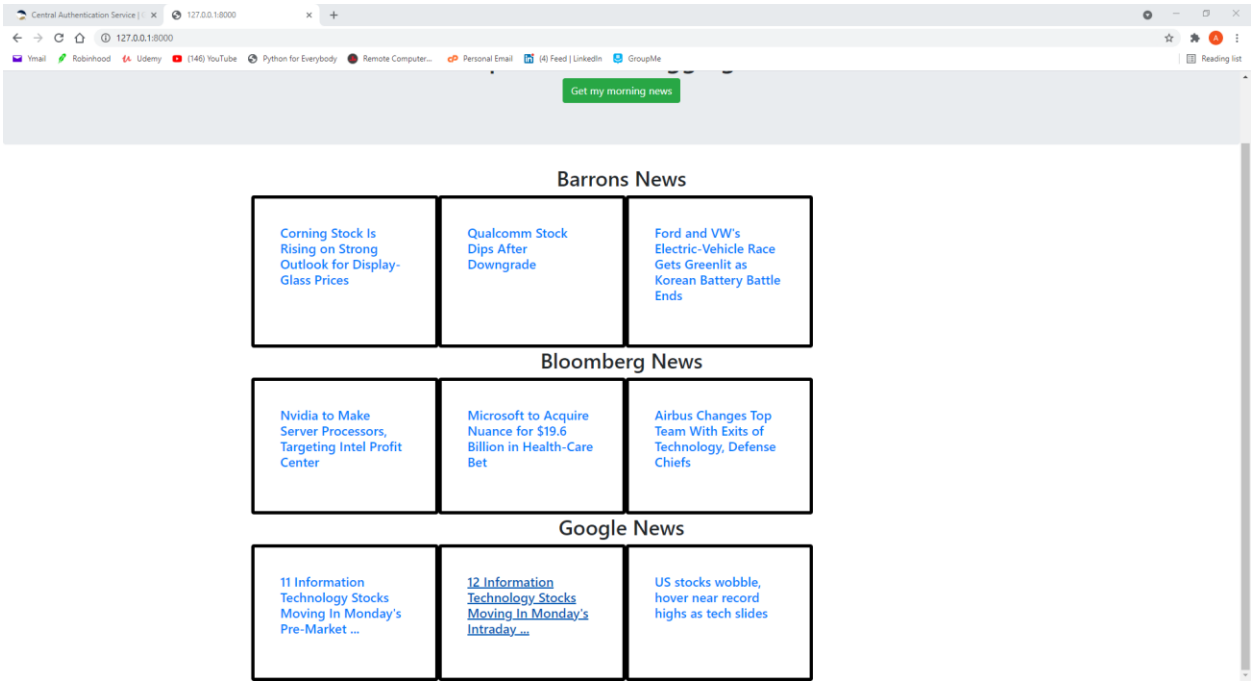


Figure 3.4.2-5 Removing card-columns to display rows.

4.4.3 Bootstrap Colors

The next objective is to add some color and spacing to the elements. In bootstrap, elements can be altered by adding css attributes in the "style" or "class" attributes of the element.

```
<div class="card bg-dark" style="width: 25rem;border:2px blue solid; margin: 0 auto;">
```

Figure 3.4.3-1 Style attribute to customize elements.

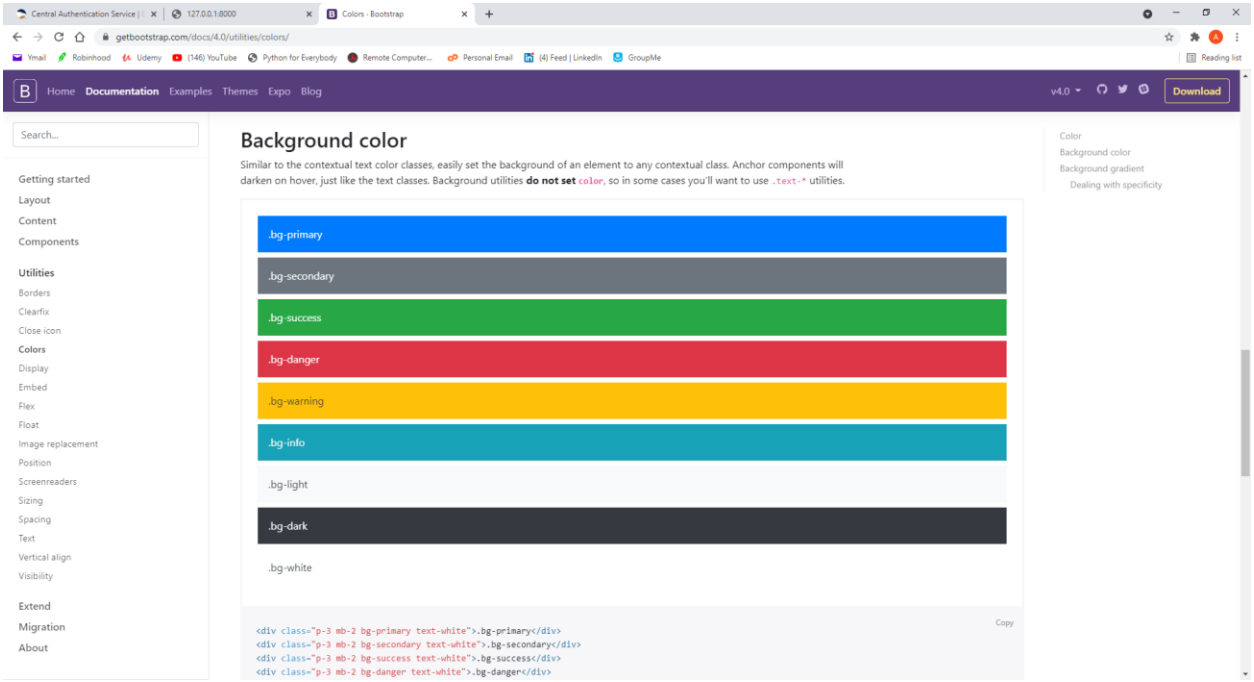


Figure 3.4.3-2 Bootstrap color documentation.

```
<body class='bg-secondary'>
```

Figure 3.4.3-3 Altering HTML body by changing class to Bootstrap background color.

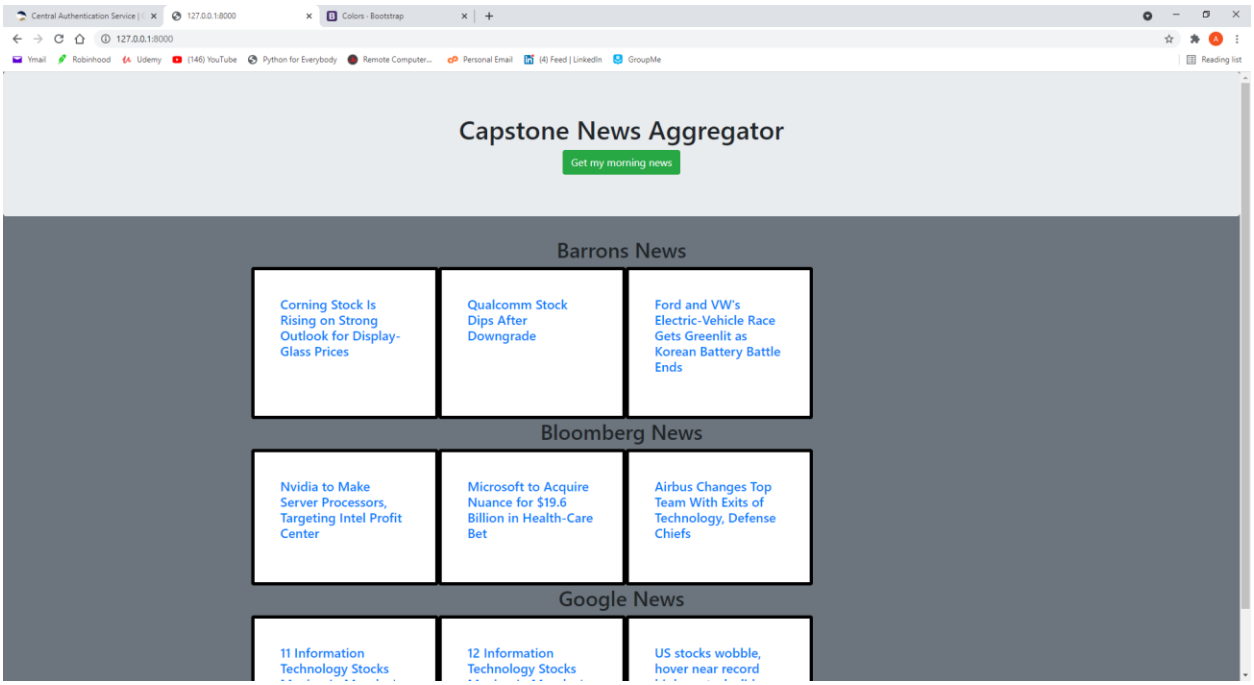


Figure 3.4.3-4 Color and layout formatting continued.

Next, I will make additional size and formatting (margin, width, etc.) changes to further align the content.

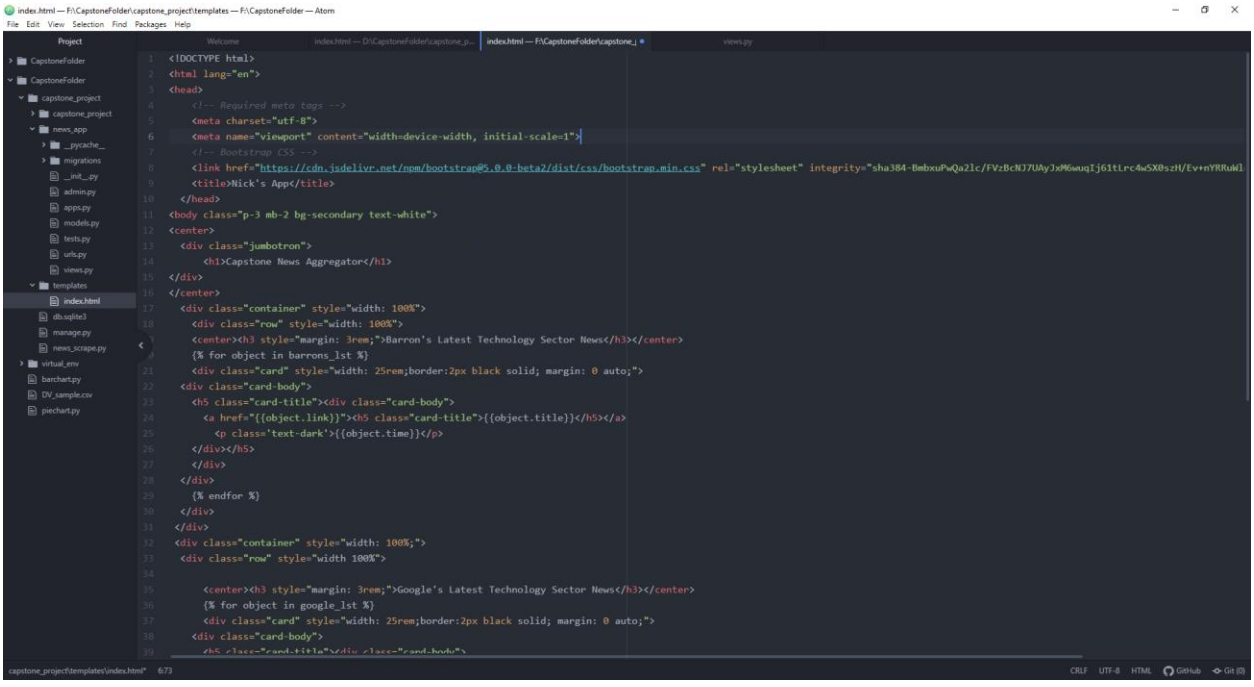


Figure 3.4.3-5 Adding additional customizations to "index.html"

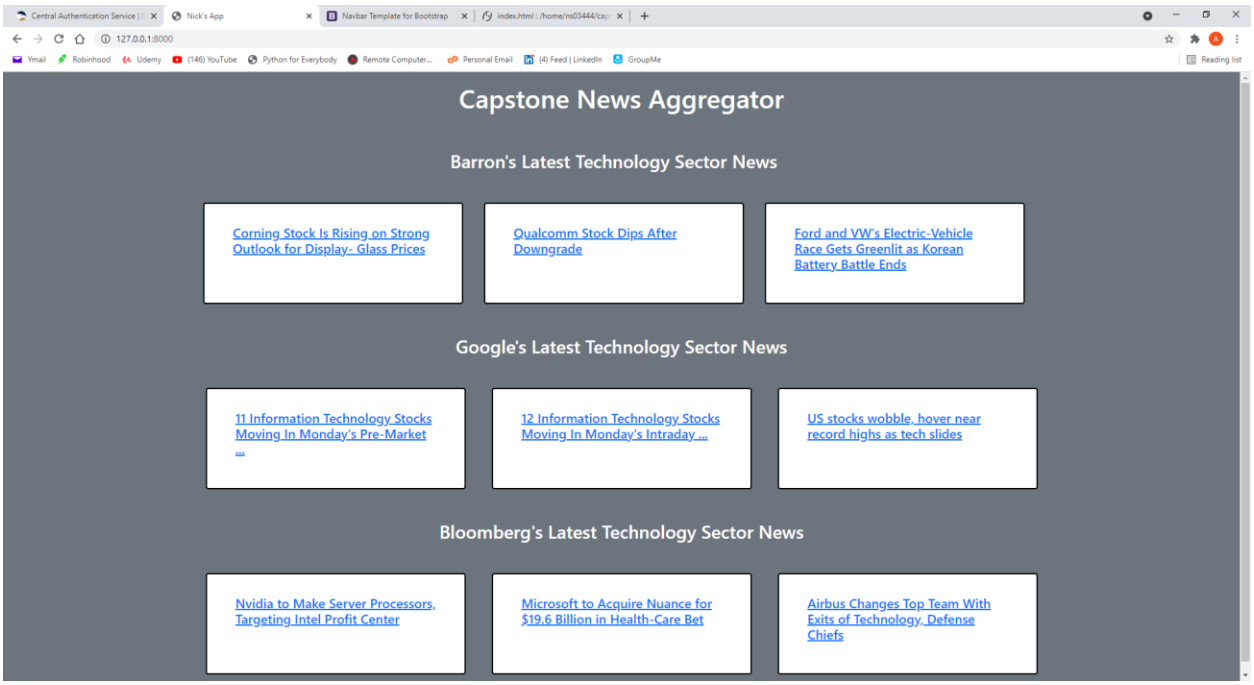


Figure 3.4.3-6 formatting "index.html" output.

***NOTE: the 'Get my Morning News' button was accidently deleted but will be added back in the next section.*

4.4.4 Bootstrap Navigation Bars

Although my webpage only consists of one template right now, I will need a navbar in the future so users can navigate between pages easily. Luckily, Bootstrap has a variety of sample header templates to choose from and customize however you please.

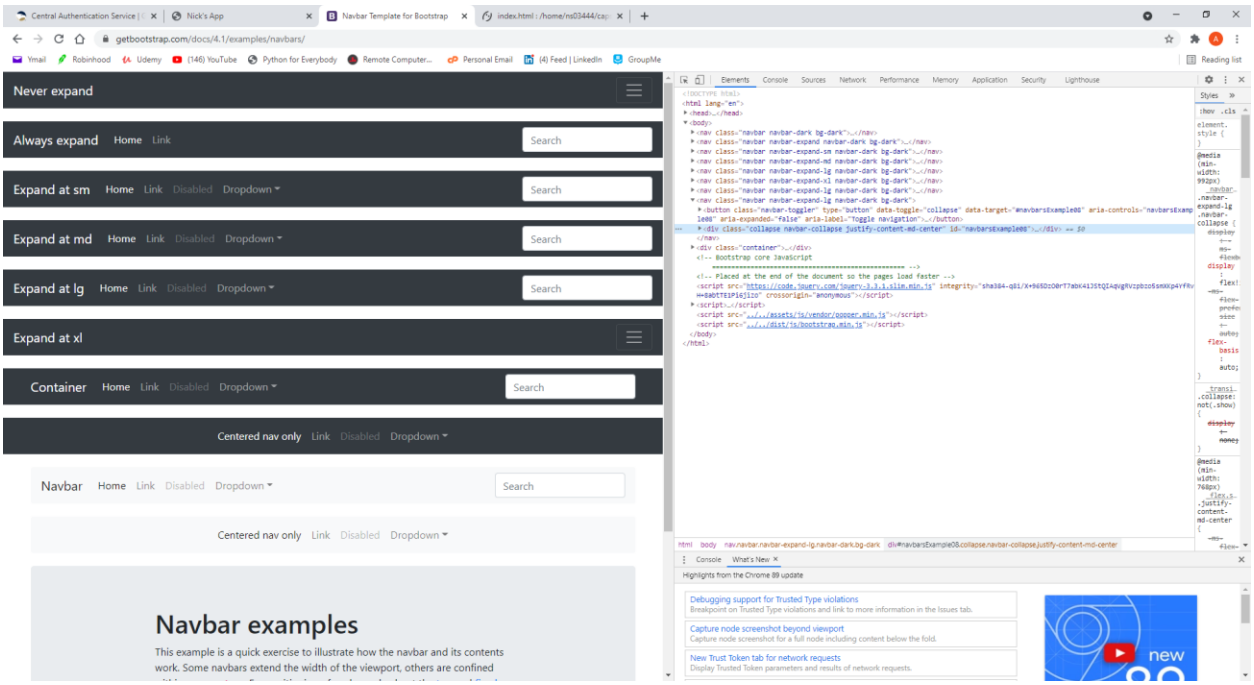


Figure 3.4.4-1

After finding a navbar that I favored, I inspected the html code of the webpage and copied/pasted the elements inside my header. Next, I will change the names to something more closely related to website. At this point, the navbar links will be static.

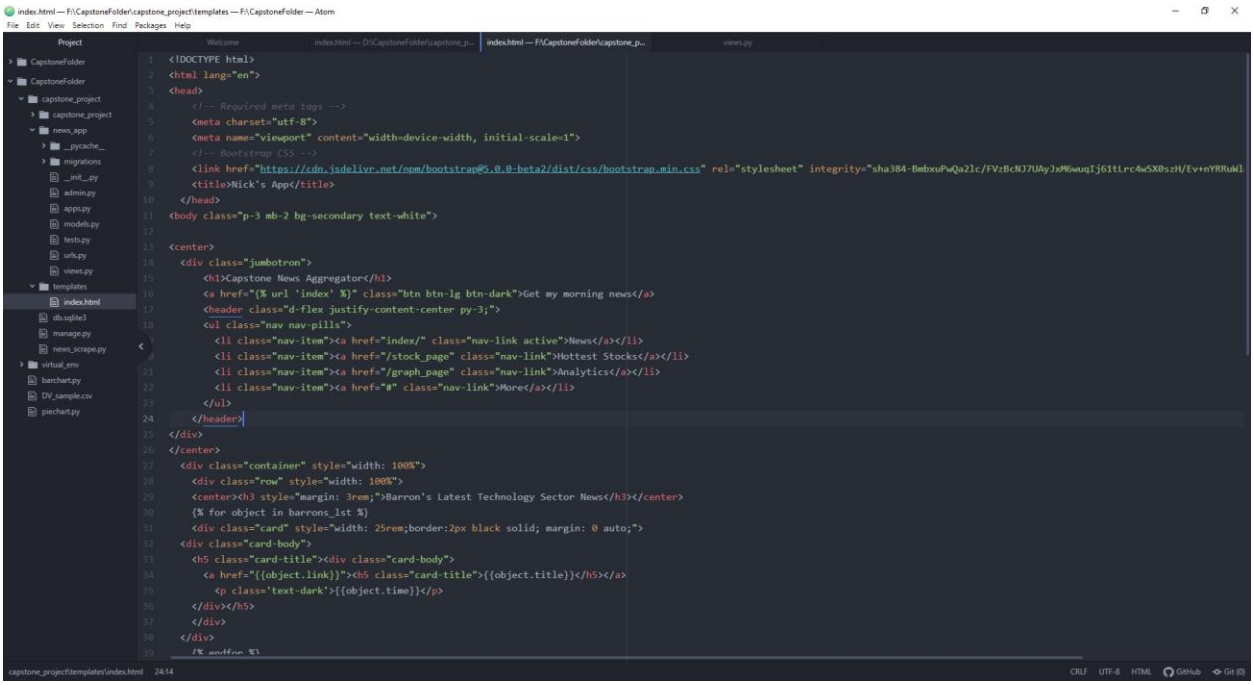
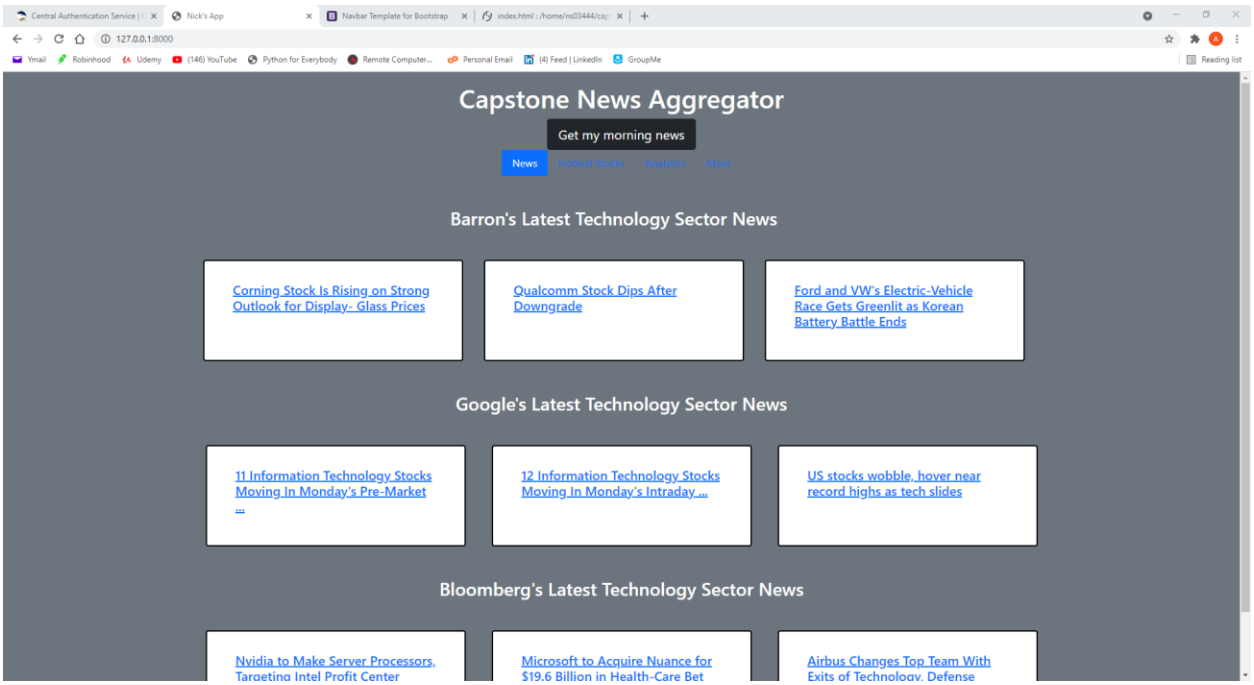


Figure 3.4.4-2 Adding a navigation bar to "index.html".



***NOTE: the article time was not being displayed in the previous sections due to a typo in my 'news_scrape.py' file. The bug was fixed, and the time was added.*

4.5 Final Output

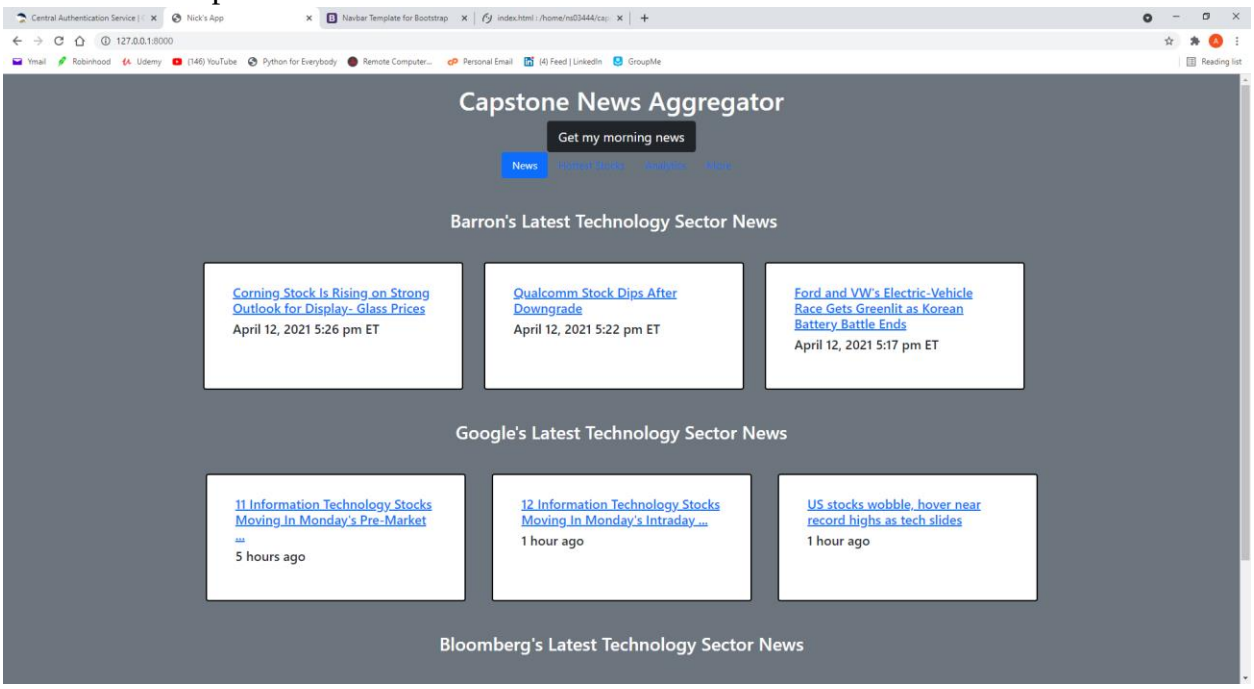


Figure 3.5-1 Final output for "index.html".

4.6 Creating the Stock App

Now that I have a much better understanding of how content can be injected into html templates and, the process of rendering templates in the browser, querying database objects, etc. I will use the same process I used for the “news_app” to make the “stock_app”, relatively. The stock app will render a web page with a table of the scraped data referenced in section 1.1 (Web Scraping Performance Data). So, displaying the data in tabular form and using the correct data types for the fields will take a different approach opposed to creating the news app. I will start by creating the app via command prompt.

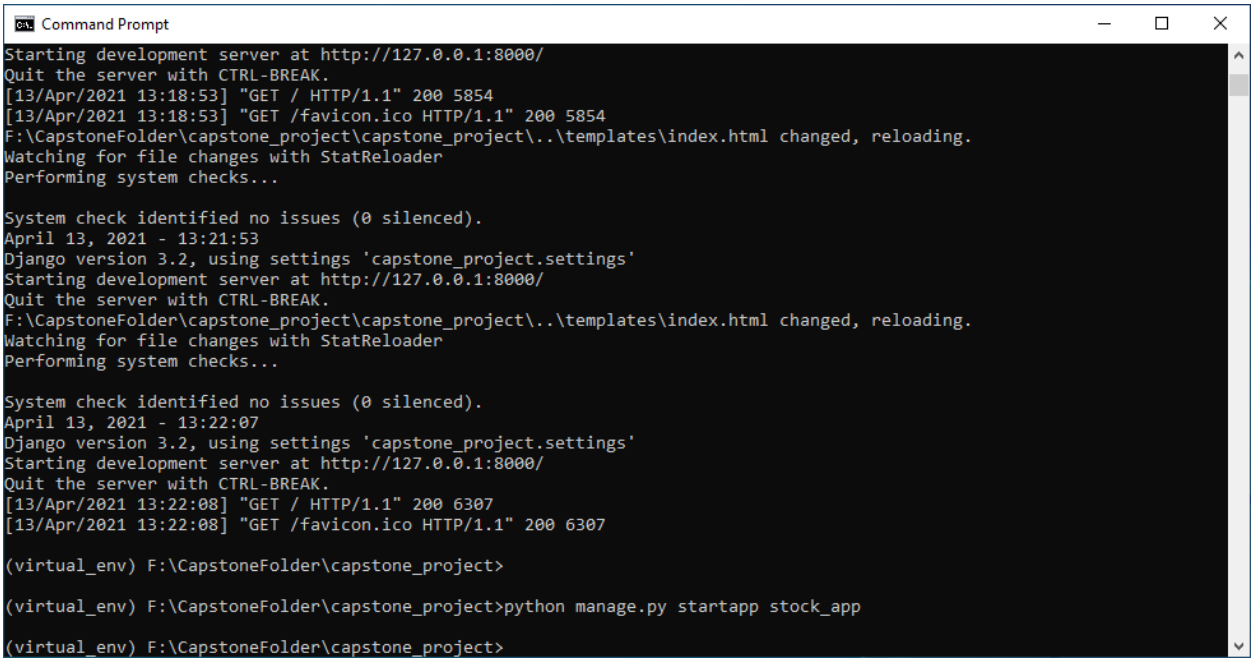


Figure 3.6-1 Creating the stock application.

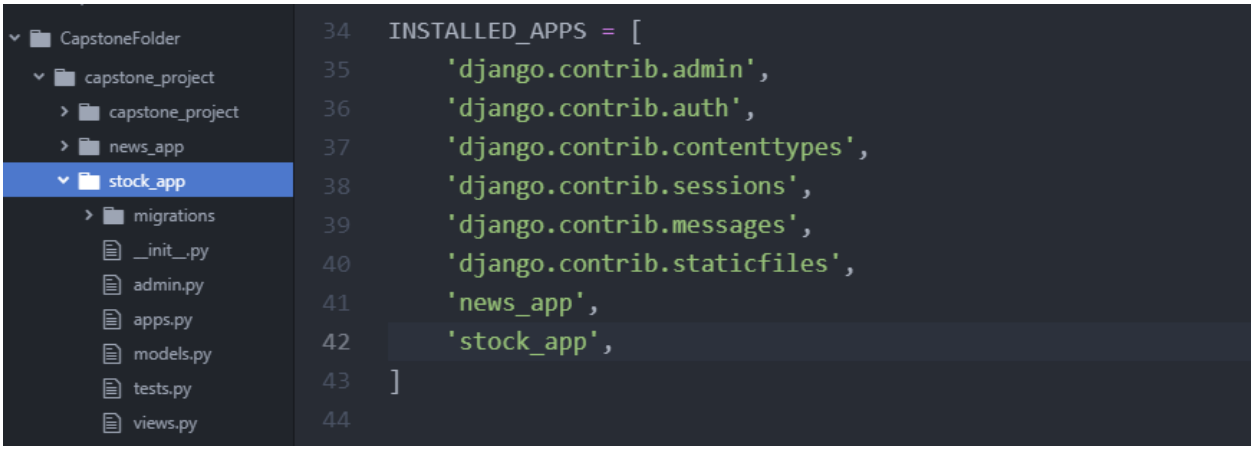


Figure 3.6-2 Configuring Django project settings for new application.

4.6.1 Creating a Django Model

Although creating this application will endure many of the procedures used previously, I can develop the app in a more efficient manner now since I am no longer experimenting with Django functionalities. Therefore, I will begin by creating a Django model “Stock” with fields to store the ticker, name, current price, and the link to that specific stock on finance.yahoo.com.

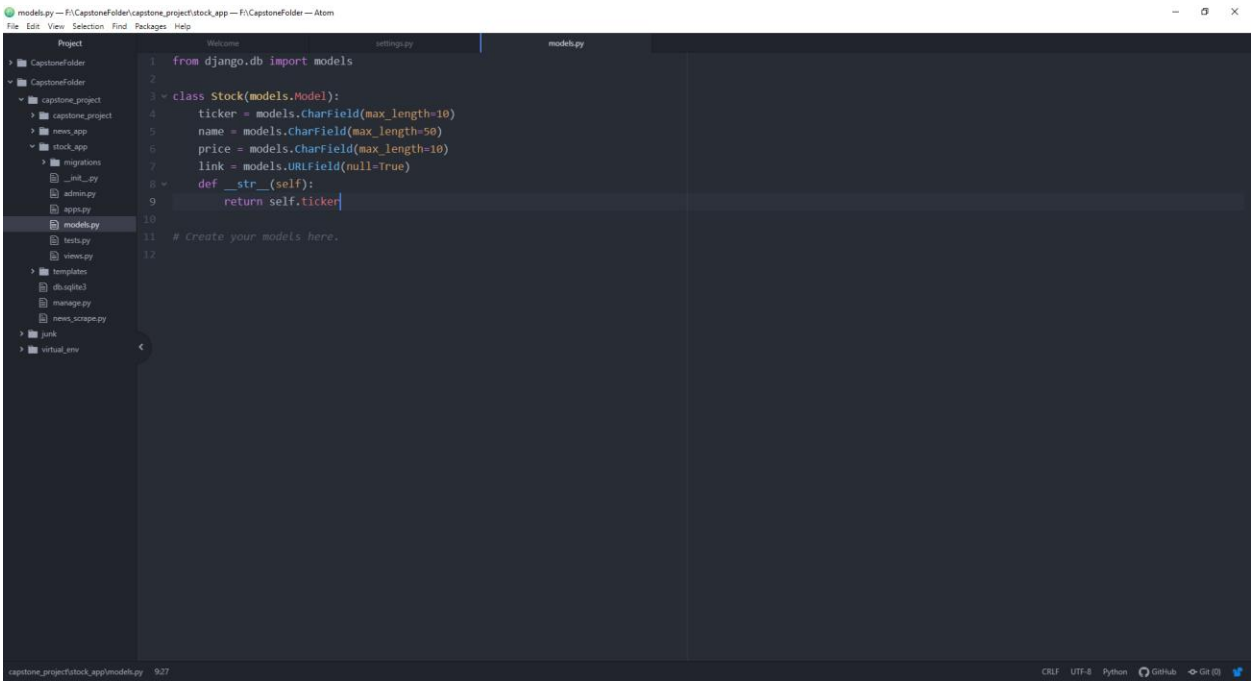


Figure 3.6.1-1 Creating the database table for stock performance data.

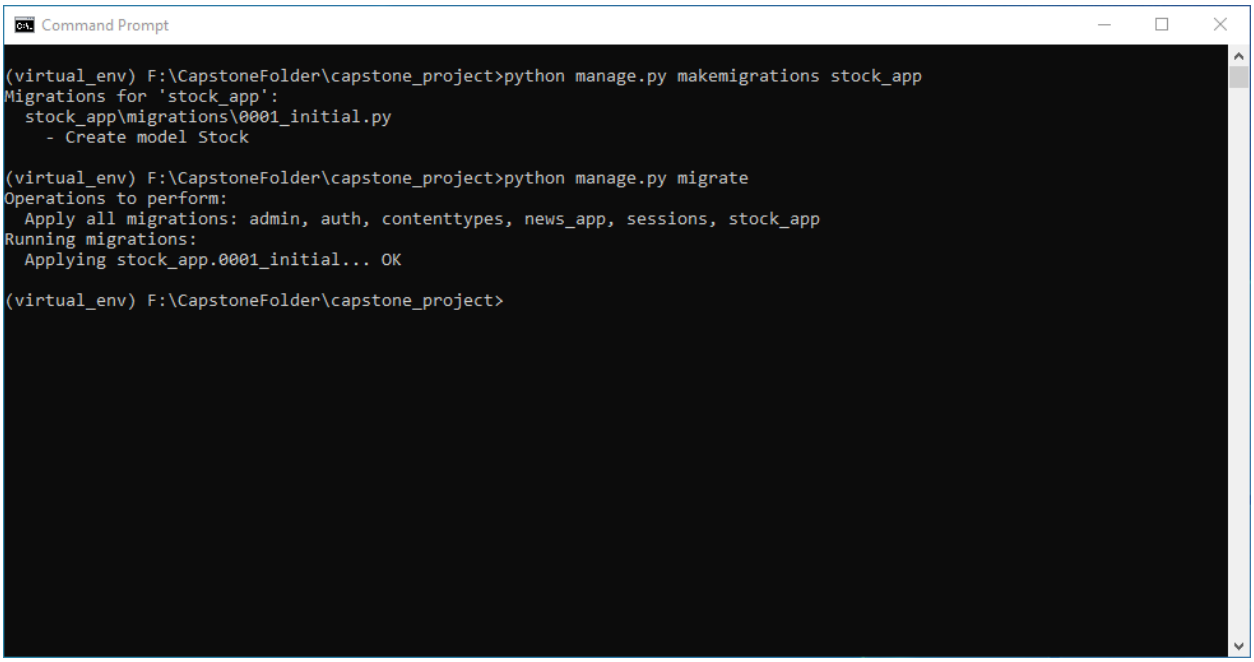


Figure 3.6.1-2 Applying migrations for new database table, "Stock".

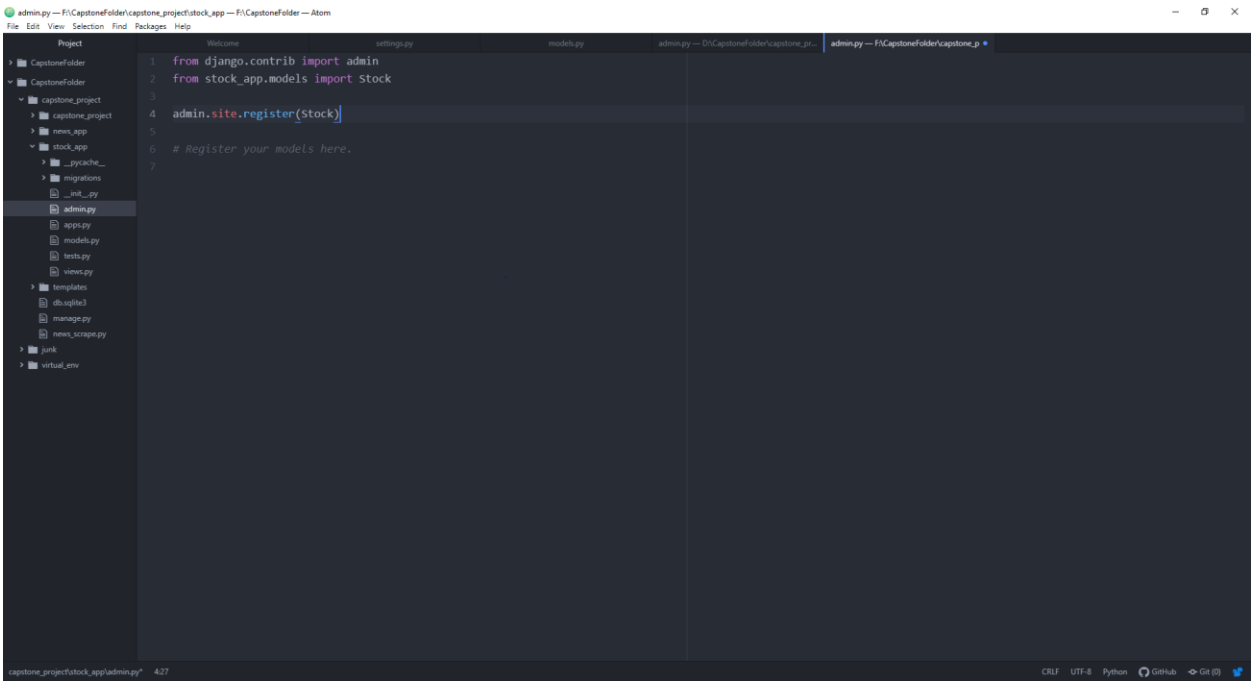


Figure 3.6.1-3 Registering "Stock" model.

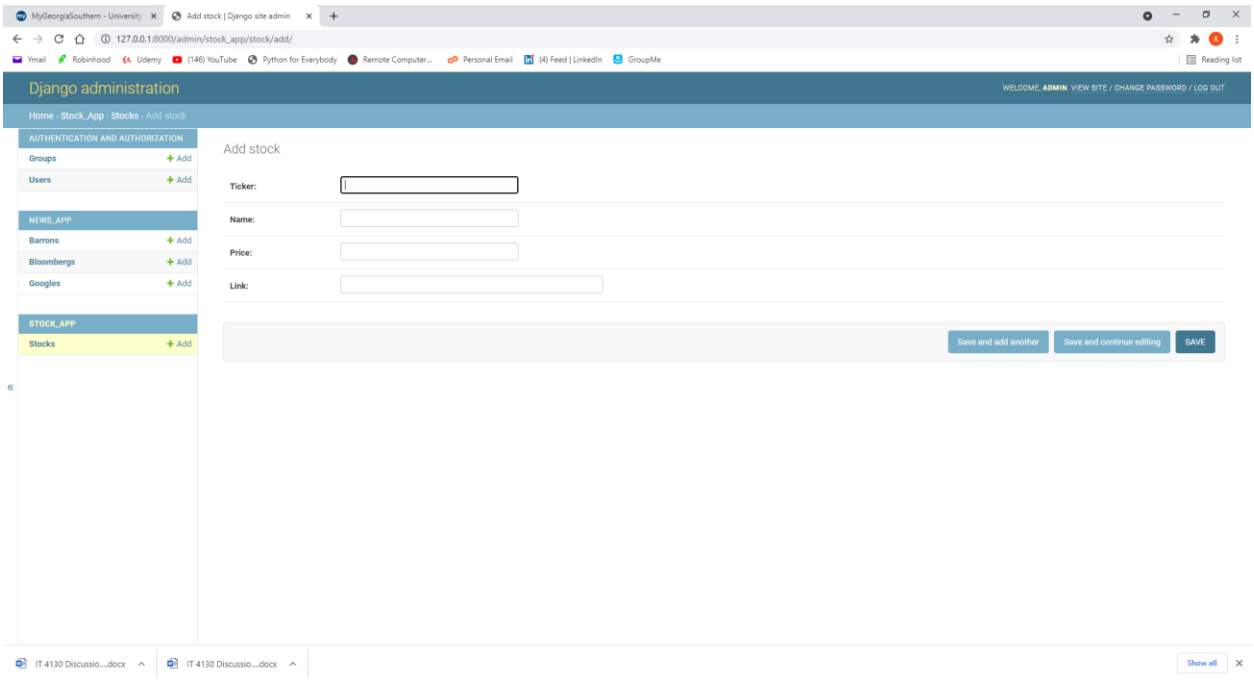
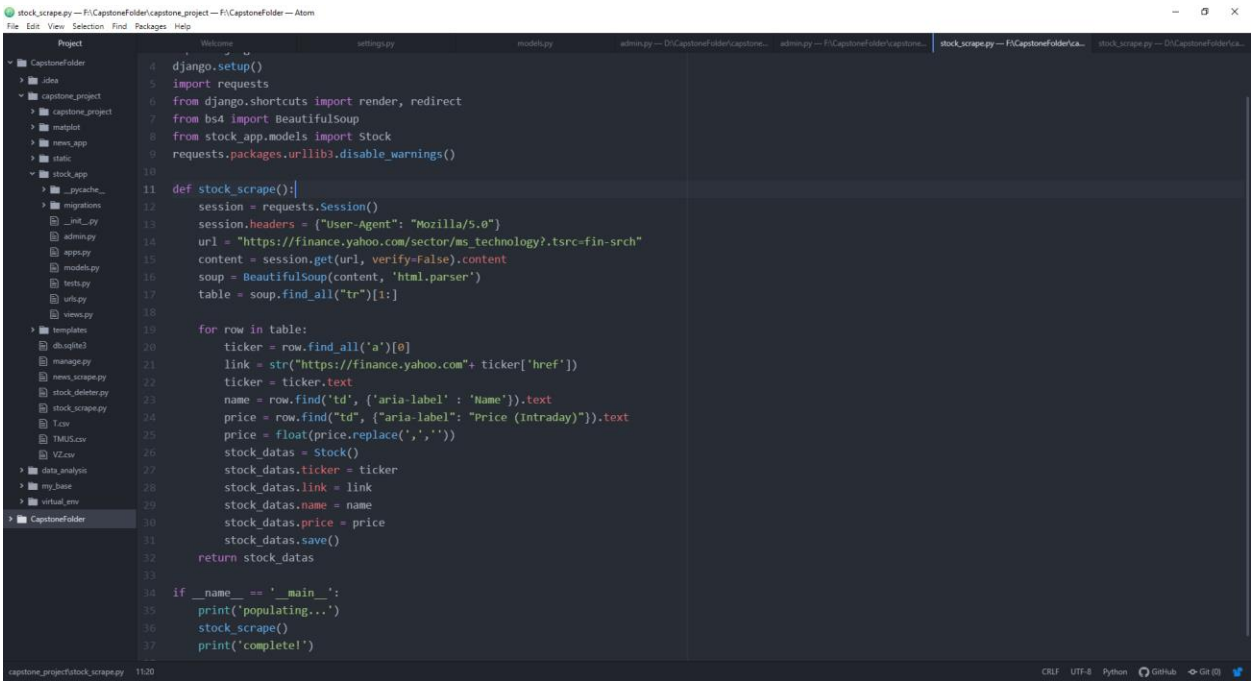


Figure 3.6.1-4 New table created successfully and accessed through admin panel.

4.6.2 Populating the Database

Since finding a successful method for properly inserting scraped data into the news app models was not intuitive, I will use the same process to populate the "Stock" table. So, I will start by creating a file "stock_scrape.py" and a function "stock_scrape" that will contain the code for scraping the data as well as storing in the model.



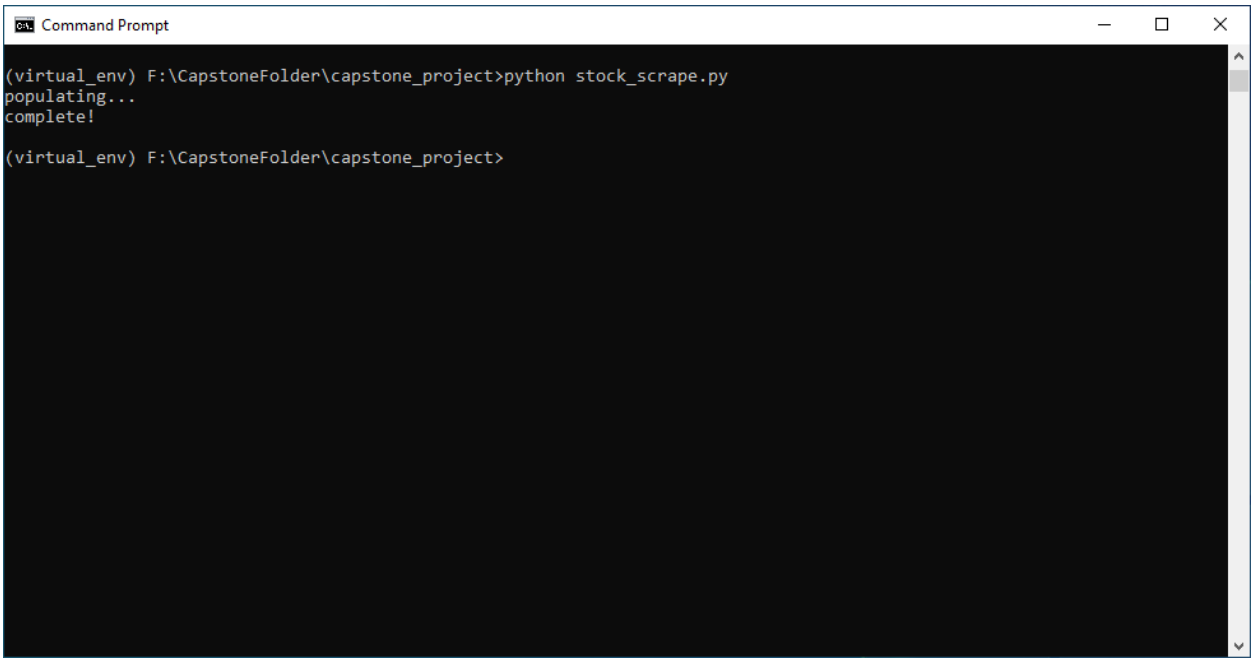
```
django.setup()
import requests
from django.shortcuts import render, redirect
from bs4 import BeautifulSoup
from stock_app.models import Stock
requests.packages.urllib3.disable_warnings()

def stock_scrape():
    session = requests.Session()
    session.headers = {"User-Agent": "Mozilla/5.0"}
    url = "https://finance.yahoo.com/sector/ms_technology?.tsrc=fin-srch"
    content = session.get(url, verify=False).content
    soup = BeautifulSoup(content, 'html.parser')
    table = soup.find_all("tr")[1:]

    for row in table:
        ticker = row.find_all('a')[0]
        link = str("https://finance.yahoo.com"+ ticker['href'])
        ticker = ticker.text
        name = row.find('td', {'aria-label' : 'Name'}).text
        price = row.find("td", {"aria-label": "Price (Intraday)"}).text
        price = float(price.replace(',',''))
        stock_datas = Stock()
        stock_datas.ticker = ticker
        stock_datas.link = link
        stock_datas.name = name
        stock_datas.price = price
        stock_datas.save()
    return stock_datas

if __name__ == '__main__':
    print('populating...')
    stock_scrape()
    print('complete!')
```

Figure 3.6.2-1 Creating new file named "stock_scrape.py" to populate new table.



```
Command Prompt

(virtual_env) F:\CapstoneFolder\capstone_project>python stock_scrape.py
populating...
complete!

(virtual_env) F:\CapstoneFolder\capstone_project>
```

Figure 3.6.2-2 Successfully running "stock_scrape.py"

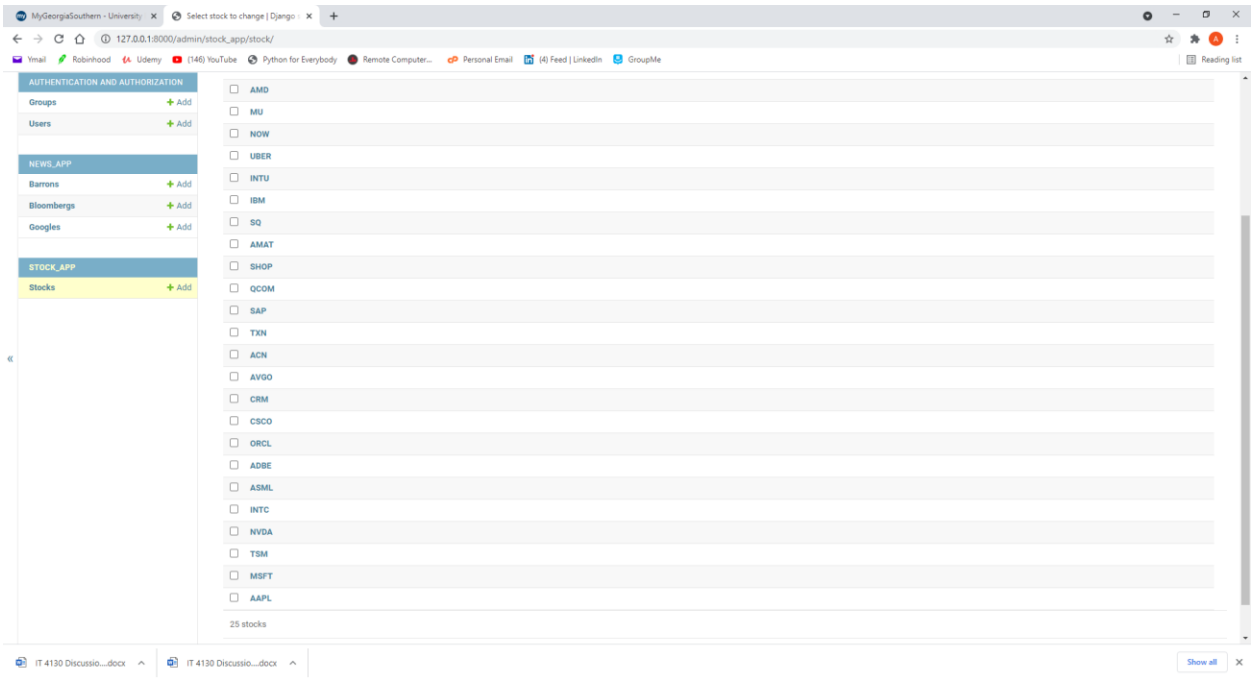


Figure 3.6.2-3 Populating "Stock" table.

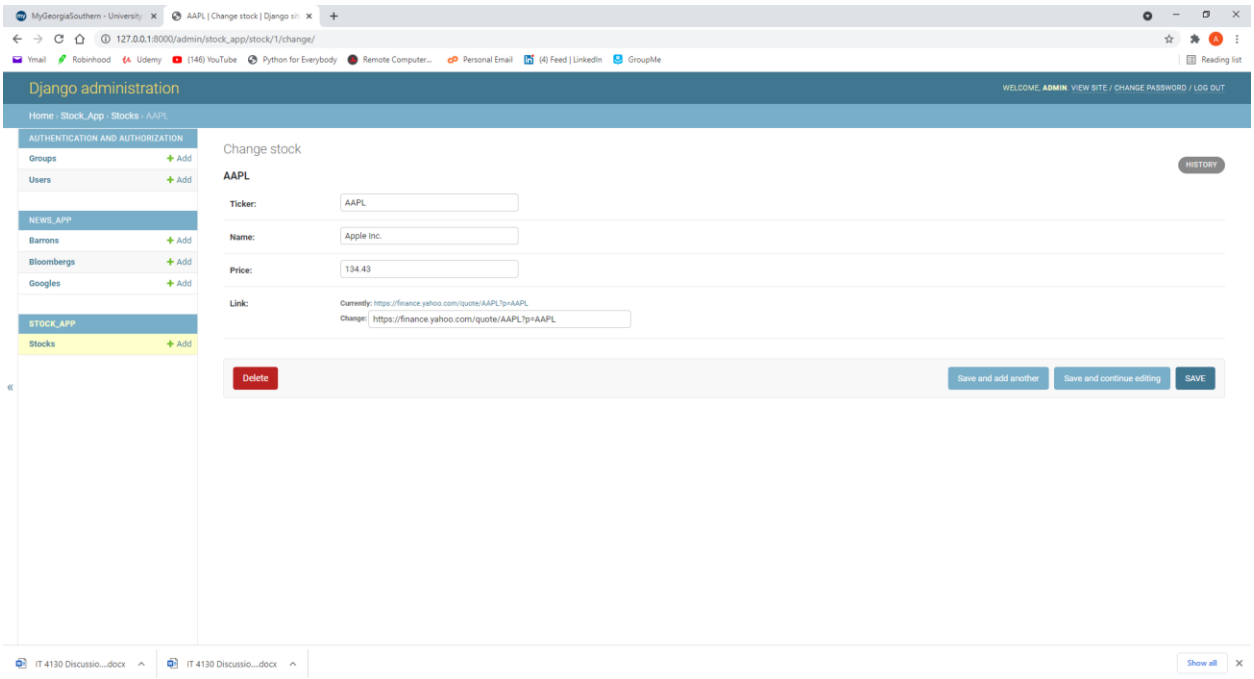
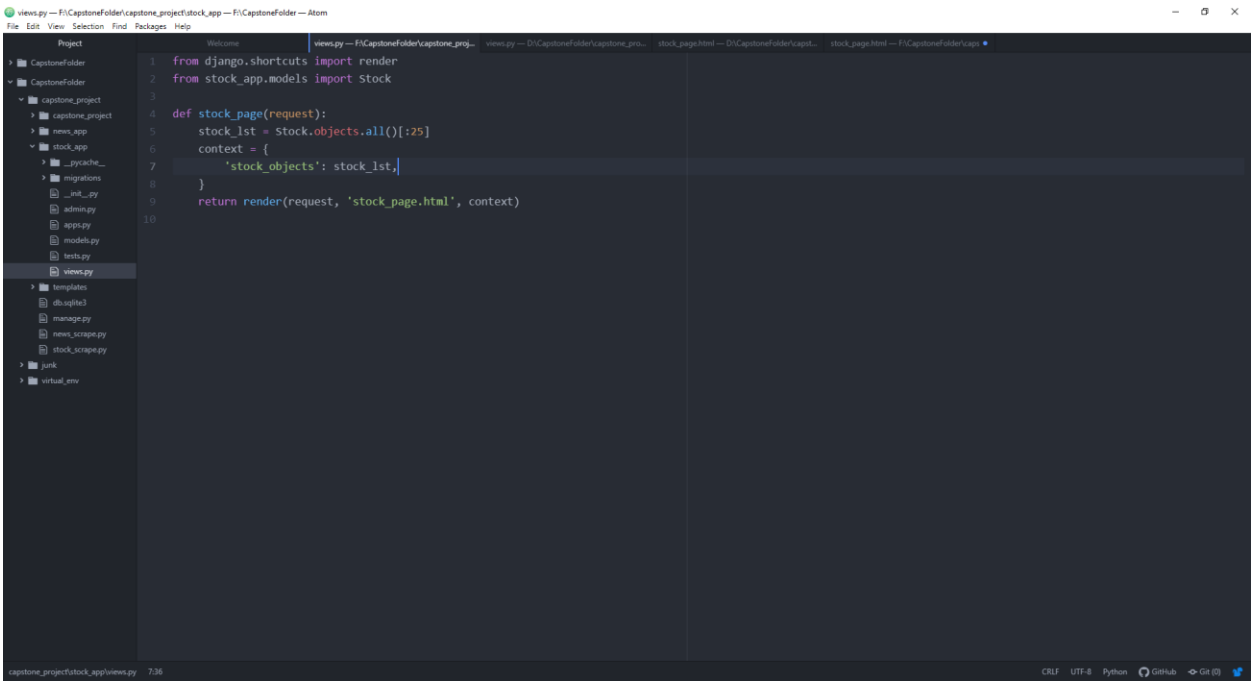


Figure 3.6.2-4 Successfully populating Stock model with correct fields and data types.

4.6.3 Developing the view

The screenshots above demonstrate the process I used to successfully populate the stock app model. Next, I will attempt rendering a new template in the browser that uses context processors to print a list of the objects in the database. I will begin by creating a new template and a function in my "views.py" file to define the context in which the data will be rendered.

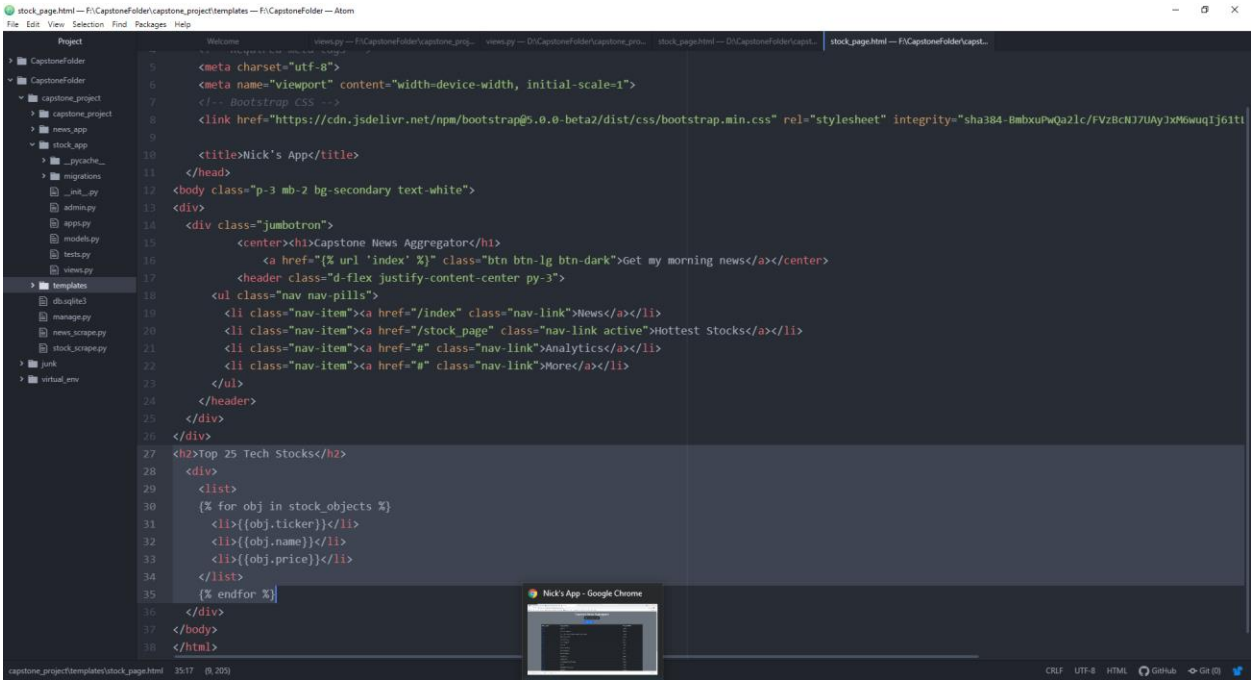


```python
from django.shortcuts import render
from stock_app.models import Stock

def stock_page(request):
    stock_lst = Stock.objects.all()[:25]
    context = {
        'stock_objects': stock_lst,
    }
    return render(request, 'stock_page.html', context)
```

Figure 3.6.3-1 "stock_app/views.py"

The screenshot above illustrates sending a request to the Stock model in the database and returning back the objects and rendering the template. Next, I will create a new template, "stock_page.html" and add the navbar from the index page in the header as well as the context processors containing a for loop to display my data in the form of a list.



```html
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<!-- Bootstrap CSS -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tL
<title>Nick's App</title>
</head>
<body class="p-3 mb-2 bg-secondary text-white">
<div>
<div class="jumbotron">
<center><h1>Capstone News Aggregator</h1>
<a href="{% url 'index' %}" class="btn btn-lg btn-dark">Get my morning news</a></center>
<header class="d-flex justify-content-center py-3">
<ul class="nav nav-pills">
<li class="nav-item"><a href="/index" class="nav-link">News</a></li>
<li class="nav-item"><a href="/stock_page" class="nav-link active">Hottest Stocks</a></li>
<li class="nav-item"><a href="#" class="nav-link">Analytics</a></li>
<li class="nav-item"><a href="#" class="nav-link">More</a></li>
</ul>
</header>
</div>
</div>
<h2>Top 25 Tech Stocks</h2>
<div>
<list>
{% for obj in stock_objects %}
<li>{{obj.ticker}}</li>
<li>{{obj.name}}</li>
<li>{{obj.price}}</li>
</list>
{% endfor %}
</div>
</body>
</html>
```

Figure 3.6.3-2 "templates/stock_page.html"

One of the last things to do before running the server is configuring the application URLs with the project URLs. To do this, I will edit the project “urls.py” file to look for a new “urls.py” file in the stock app directory. After the new file is created, I will give it a path to locate my “stock_page” function in the views file and an extension of “stock_page/” to give the webpage a unique address. Which explains the “href=/stock_page” in the navbar shown above. This will allow the “hottest stocks” link to dynamically follow the link to “stock_page.html” when clicked.

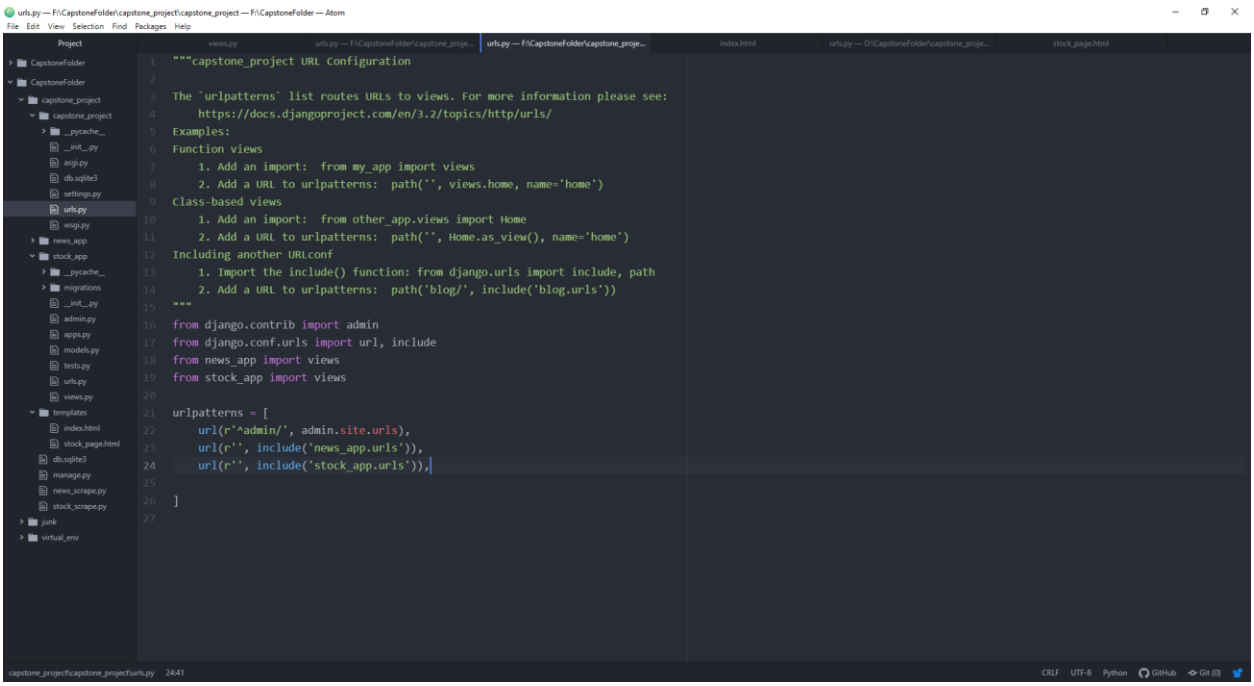


Figure 3.6.3-3 Configuring URLs for the project

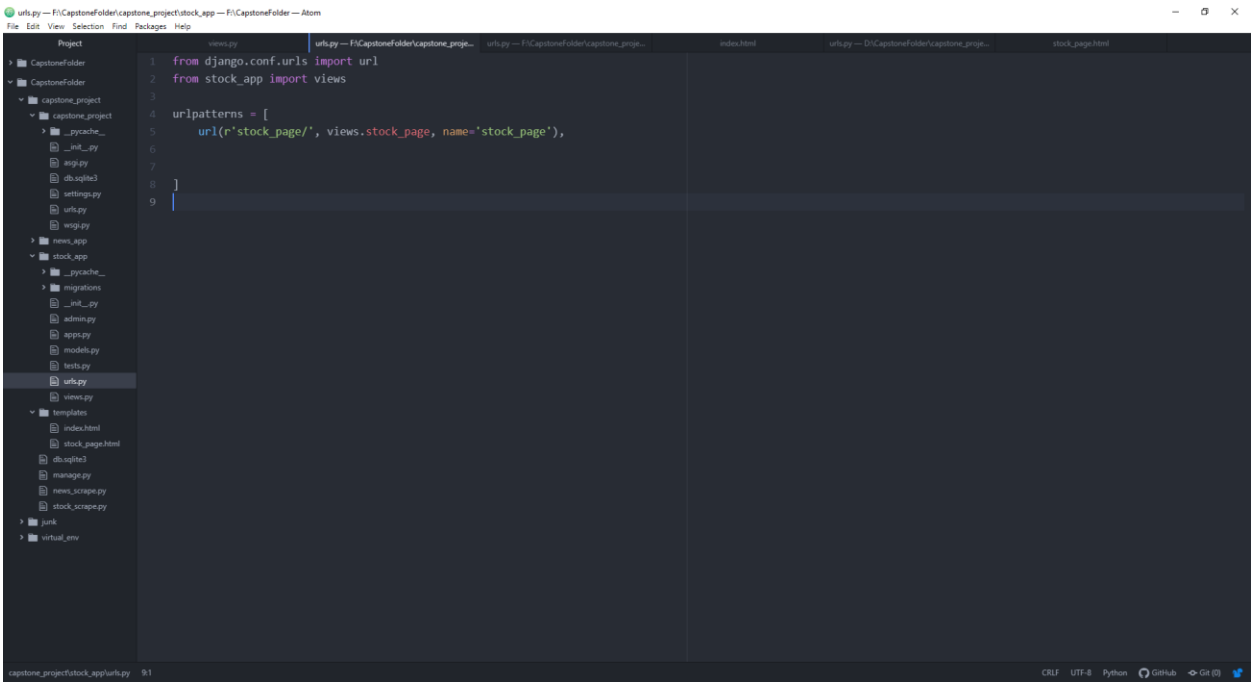


Figure 3.6.3-4 Configuring stock application URLs.

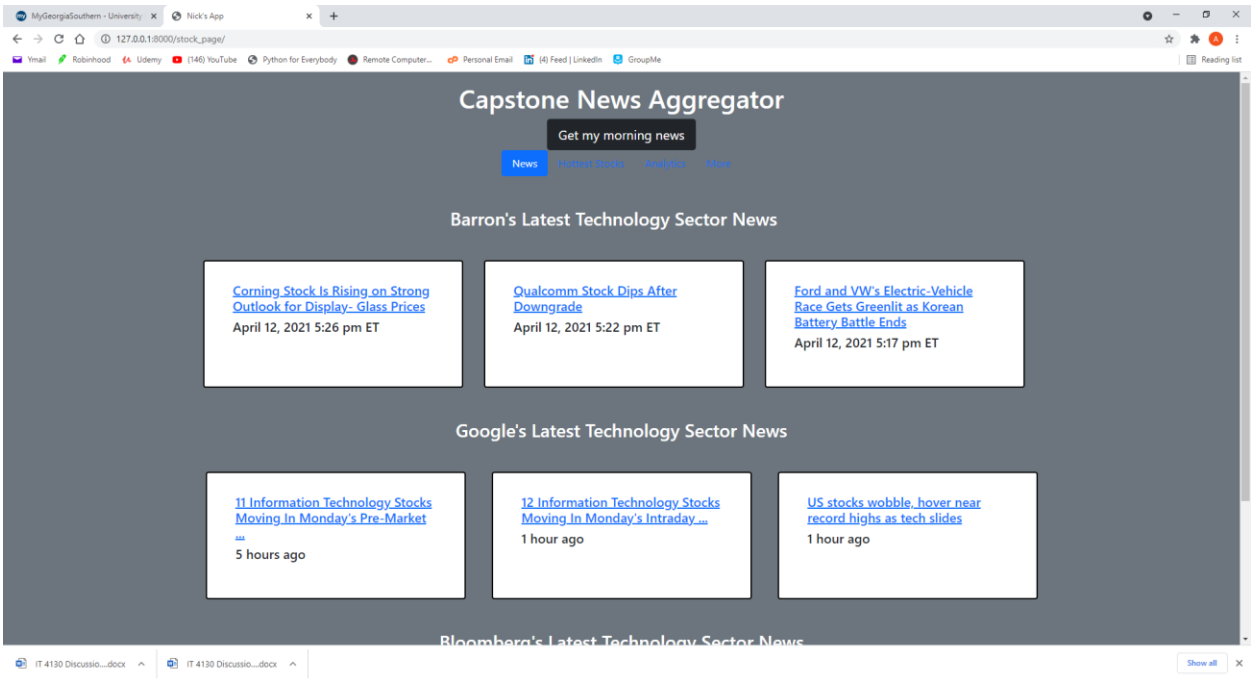


Figure 3.6.3-5 "/stock_page" extension in URL not rendering correct template.

If you notice at the top of the image above, in the address bar, "/stock_page" is added to the end of the local address, however, the index page is being displayed rather than the stock data. After some debugging, I found that my index.html page needed a unique extension as well, therefore it is now located at the local address followed by "/index".

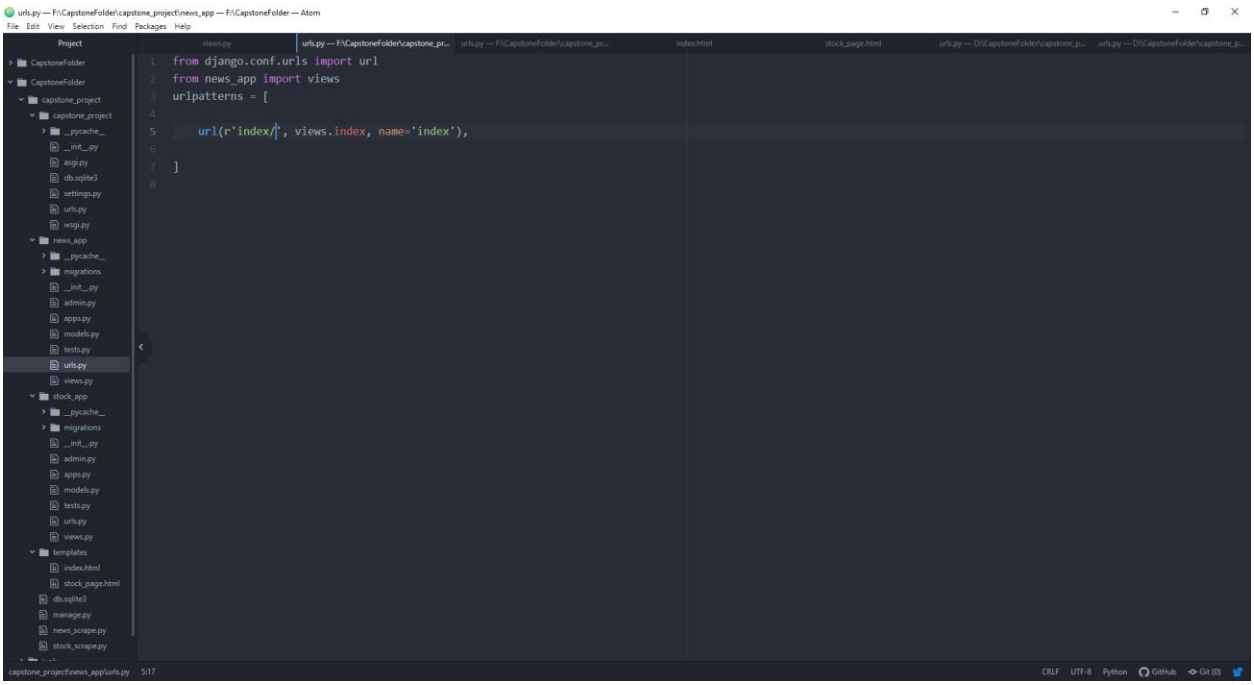


Figure 3.6.3-6 Configuring unique address for index page.

Now, the “stock_page.html” is being properly rendered when requested, as well as, displaying the stock data as a list. Also, the navbar is actively working, and each page has a unique address that can be seen in the address bar at the top of the images below.

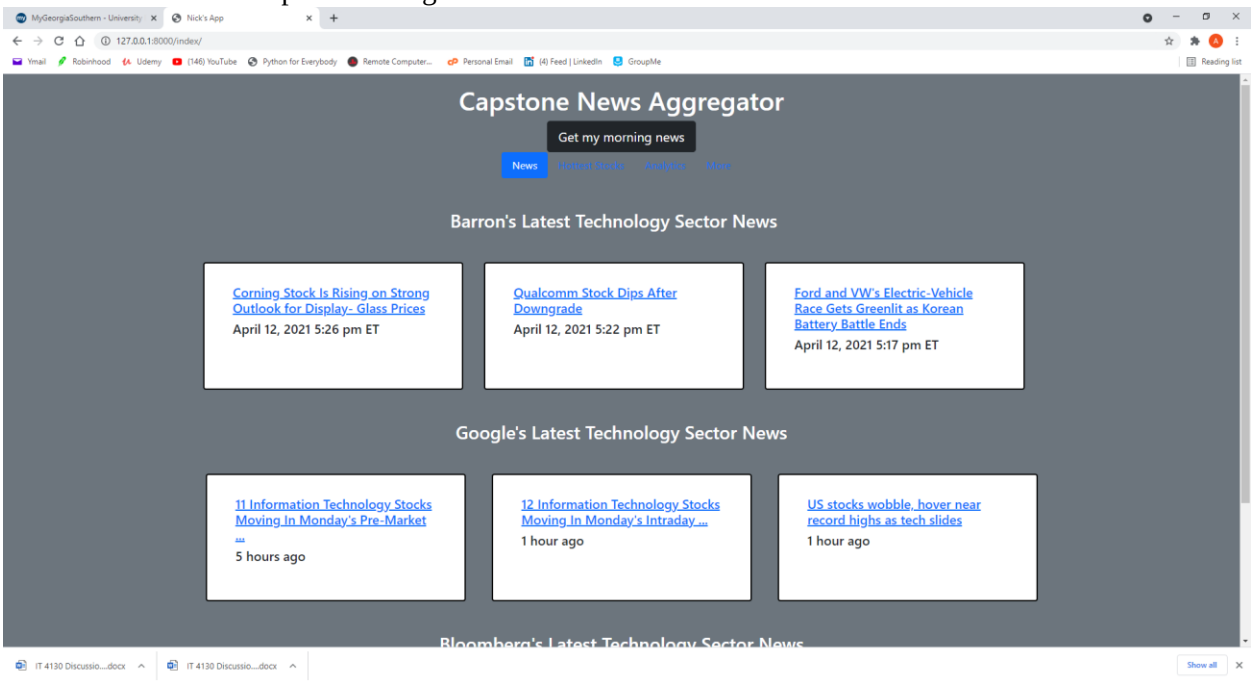


Figure3.6.3-7 News content being displayed at 127.0.0.1:8000/index/

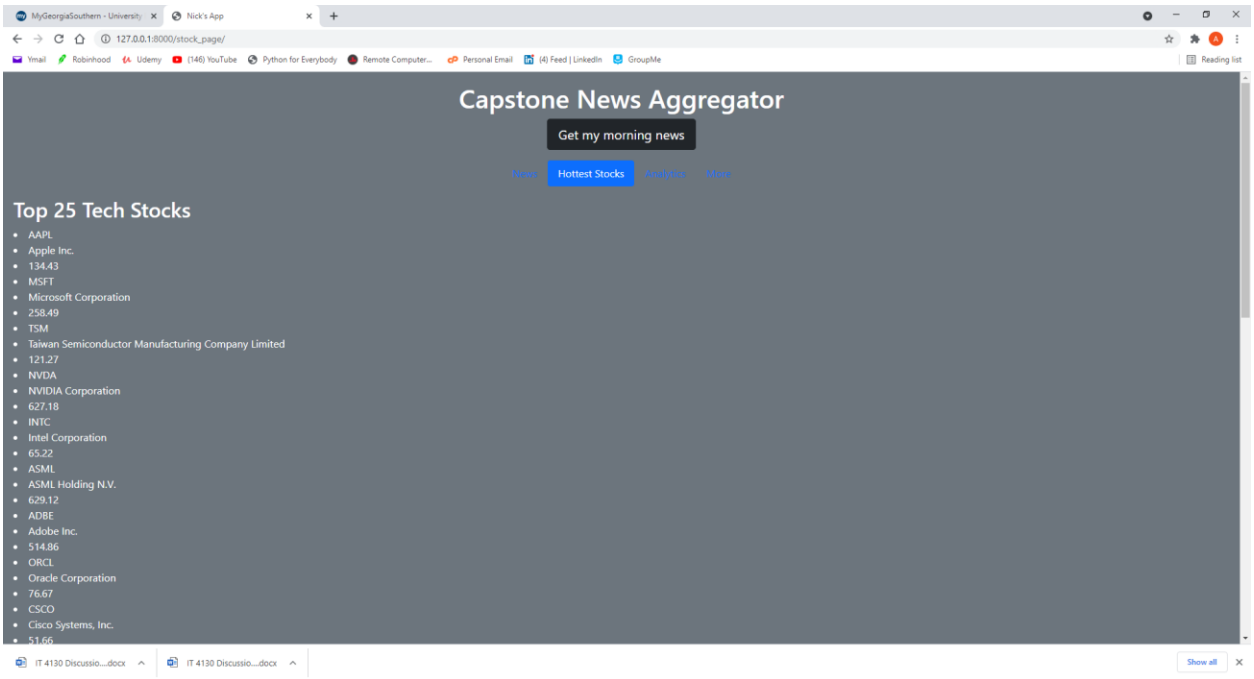


Figure3.6.3-8 Stock content being displayed at 127.0.0.1:8000/stock_page/

4.7 Designing the Stock page

Now that I have successfully displayed the data in the browser, I will now attempt converting the list of objects into tabular format. Fortunately, when I spent time browsing the Bootstrap documentation for the news article application, I saw that Django offered a variety of table element examples as well as great documentation on customizing them.

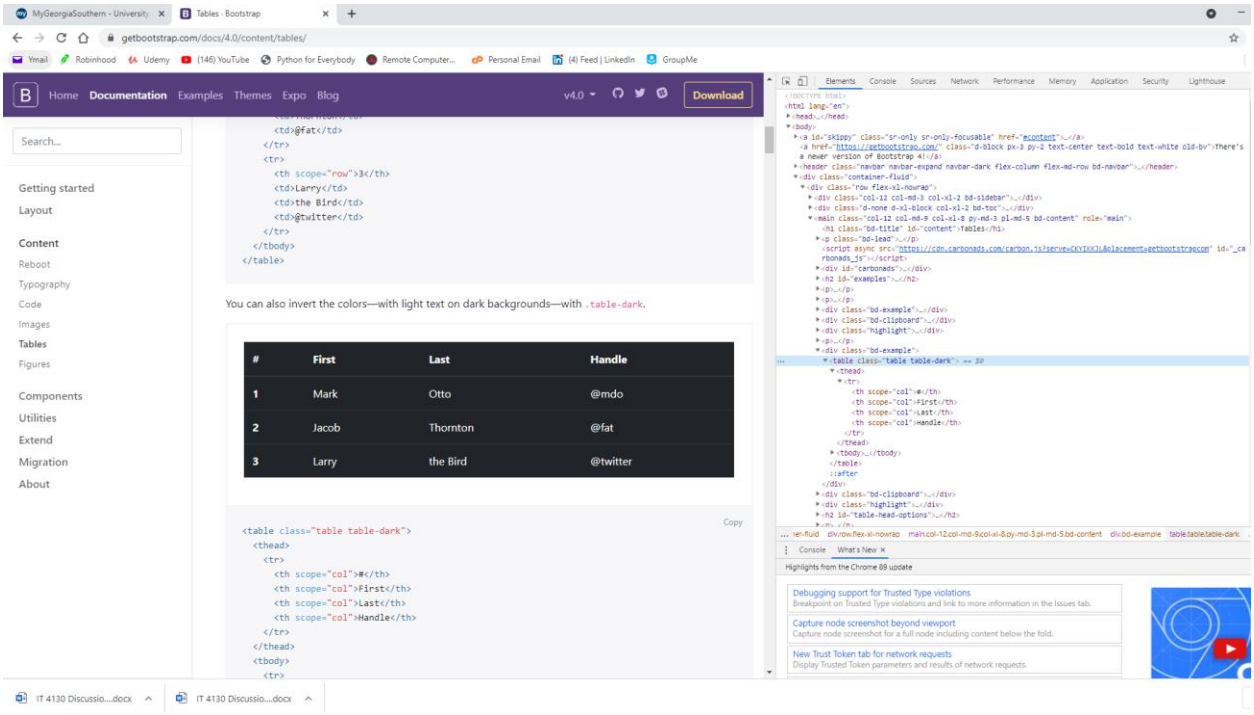
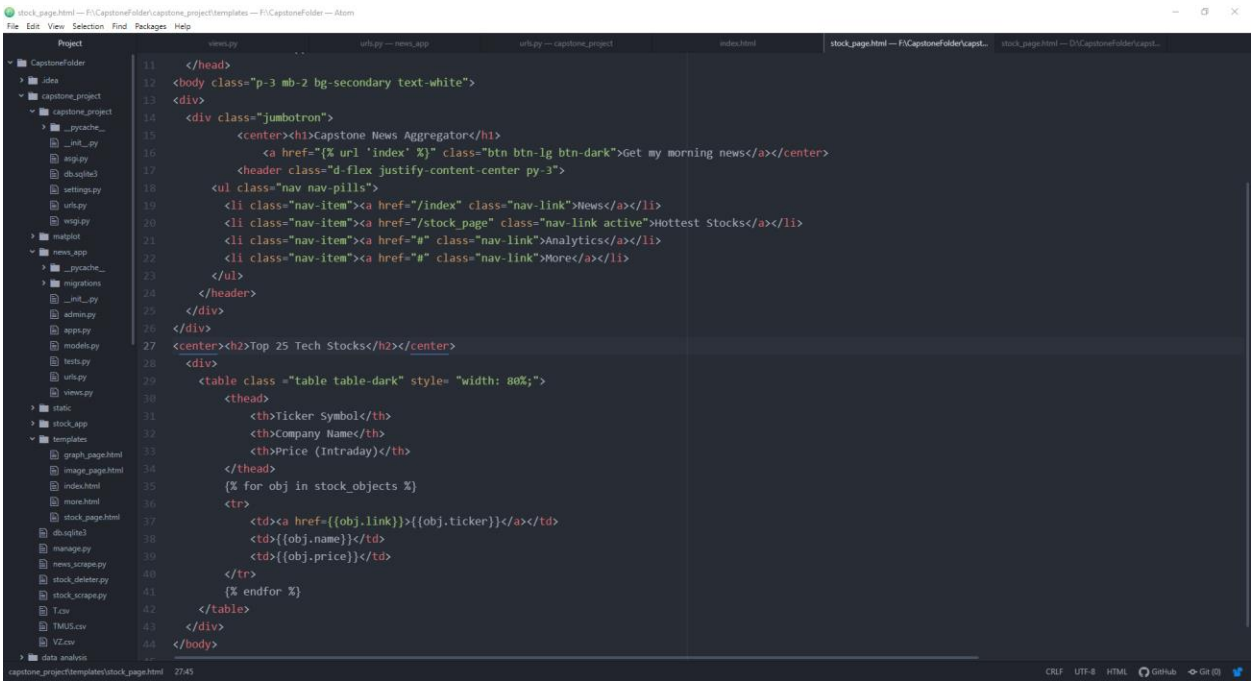


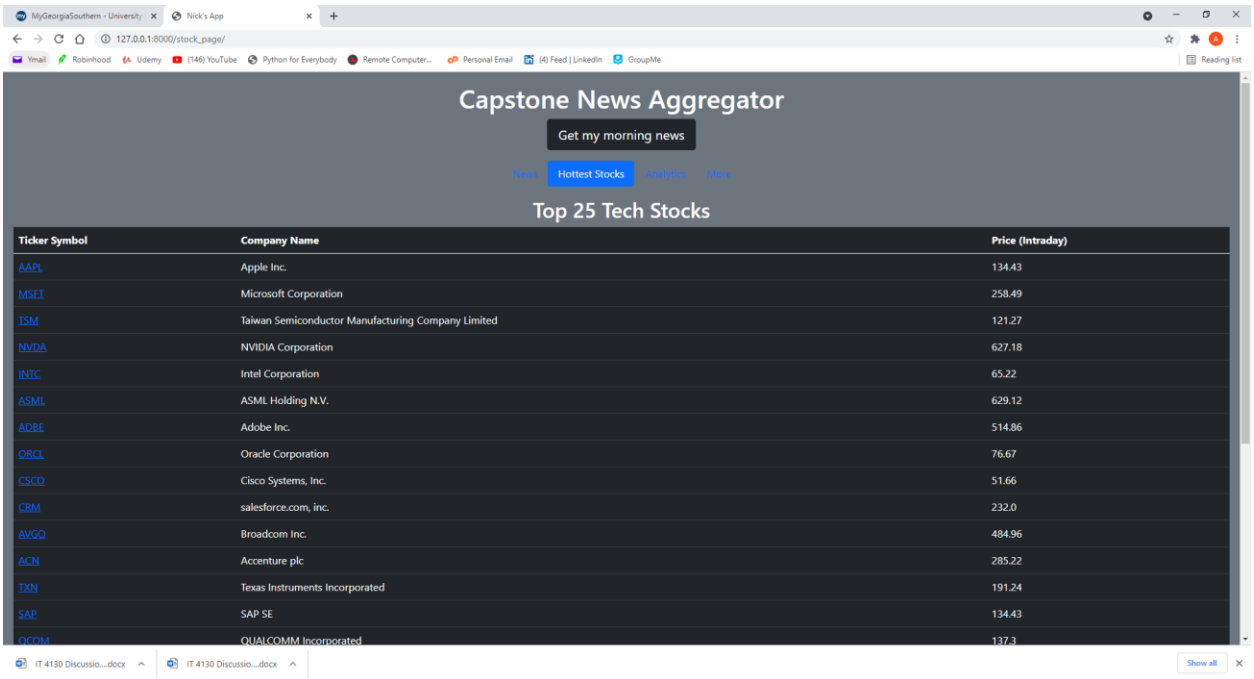
Figure3.7-1 Bootstrap table documentation.

I chose to go with the dark table theme for my table. Next, I will define the table size, headers, and implement context processors in the place of the table data elements (<td>) to inject my content into the table. I will also be converting the ticker symbols to hyperlinks by passing the Stock model's "link" object to the ticker elements "href" attribute.



```
  </head>
<body class="p-3 mb-2 bg-secondary text-white">
<div>
  <div class="jumbotron">
          <center><h1>Capstone News Aggregator</h1>
              <a href="{% url 'index' %}" class="btn btn-lg btn-dark">Get my morning news</a></center>
          <header class="d-flex justify-content-center py-3">
      <ul class="nav nav-pills">
        <li class="nav-item"><a href="/index" class="nav-link">News</a></li>
        <li class="nav-item"><a href="/stock_page" class="nav-link active">Hottest Stocks</a></li>
        <li class="nav-item"><a href="#" class="nav-link">Analytics</a></li>
        <li class="nav-item"><a href="#" class="nav-link">More</a></li>
      </ul>
    </header>
  </div>
</div>
<center><h2>Top 25 Tech Stocks</h2></center>
  <div>
    <table class ="table table-dark" style= "width: 80%;">
        <thead>
            <th>Ticker Symbol</th>
            <th>Company Name</th>
            <th>Price (Intraday)</th>
        </thead>
        {% for obj in stock_objects %}
        <tr>
            <td><a href={{obj.link}}>{{obj.ticker}}</a></td>
            <td>{{obj.name}}</td>
            <td>{{obj.price}}</td>
        </tr>
        {% endfor %}
    </table>
  </div>
</body>
```

Figure3.7-2 creating table element in stock page template.



Capstone News Aggregator

Get my morning news

News | Hottest Stocks | Analytics | More

Top 25 Tech Stocks

Ticker Symbol	Company Name	Price (Intraday)
AAPL	Apple Inc.	134.43
MSFT	Microsoft Corporation	258.49
TSM	Taiwan Semiconductor Manufacturing Company Limited	121.27
NVDA	NVIDIA Corporation	627.18
INTC	Intel Corporation	65.22
ASML	ASML Holding N.V.	629.12
ADBE	Adobe Inc.	514.86
ORCL	Oracle Corporation	76.67
CSCO	Cisco Systems, Inc.	51.66
CRM	salesforce.com, inc.	232.0
AVGO	Broadcom Inc.	484.96
ACN	Accenture plc	285.22
TXN	Texas Instruments Incorporated	191.24
SAP	SAP SE	134.43
QCOM	QUALCOMM Incorporated	137.3

Figure3.7-3 Unformatted table displayed.

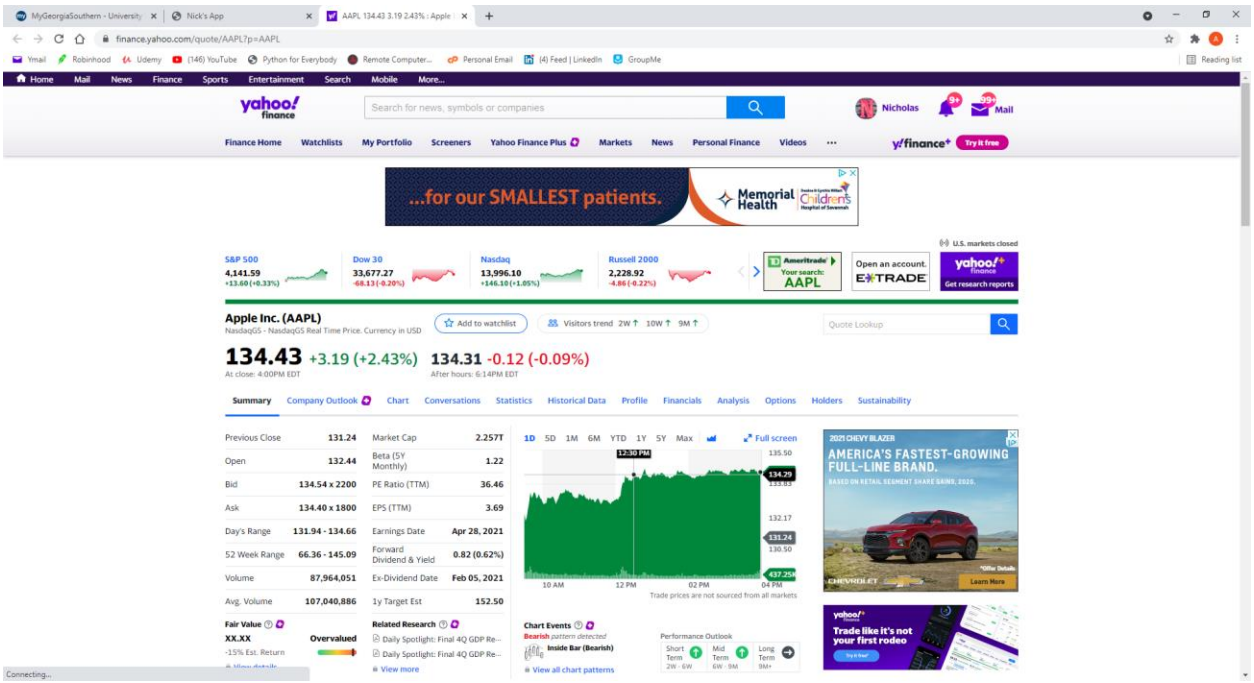


Figure3.7-4 Ticker hyperlinks working successfully.

4.7.1 Table Formatting

Since the table is currently stretched across the entire page but only contains three columns of data, I will decrease the width and make the proper alignments to center the table.

```
29      <table class ="table table-dark" style= "width: 80%;">
```

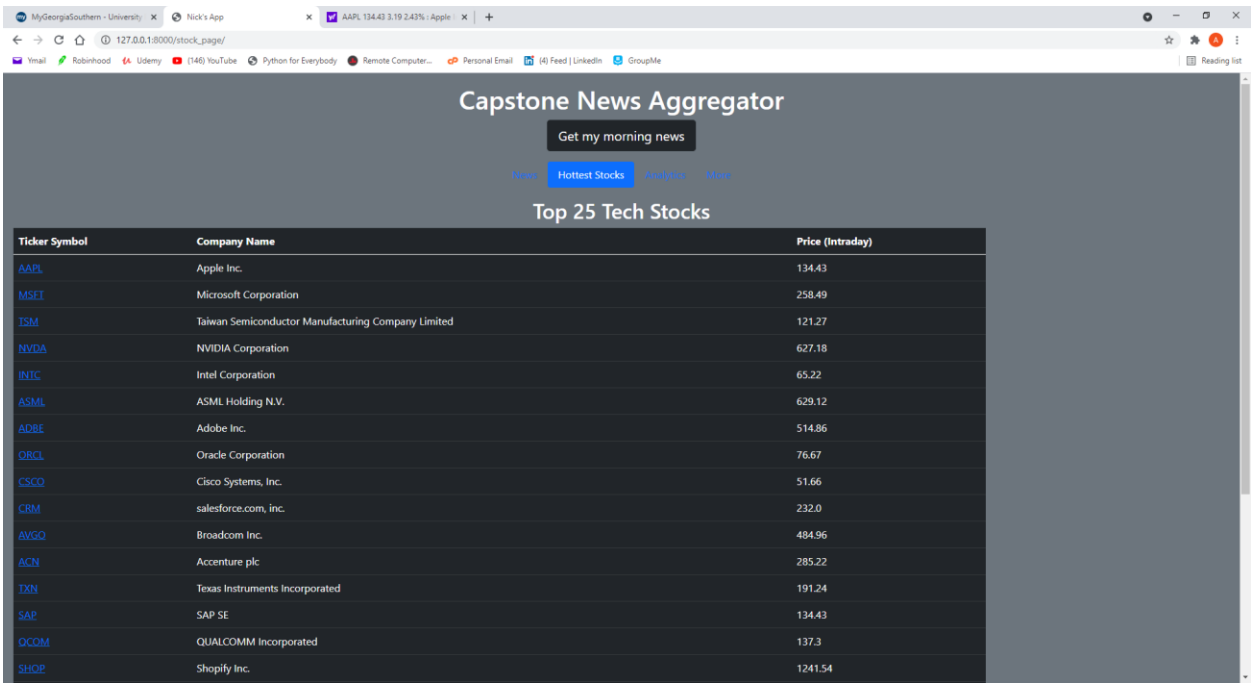
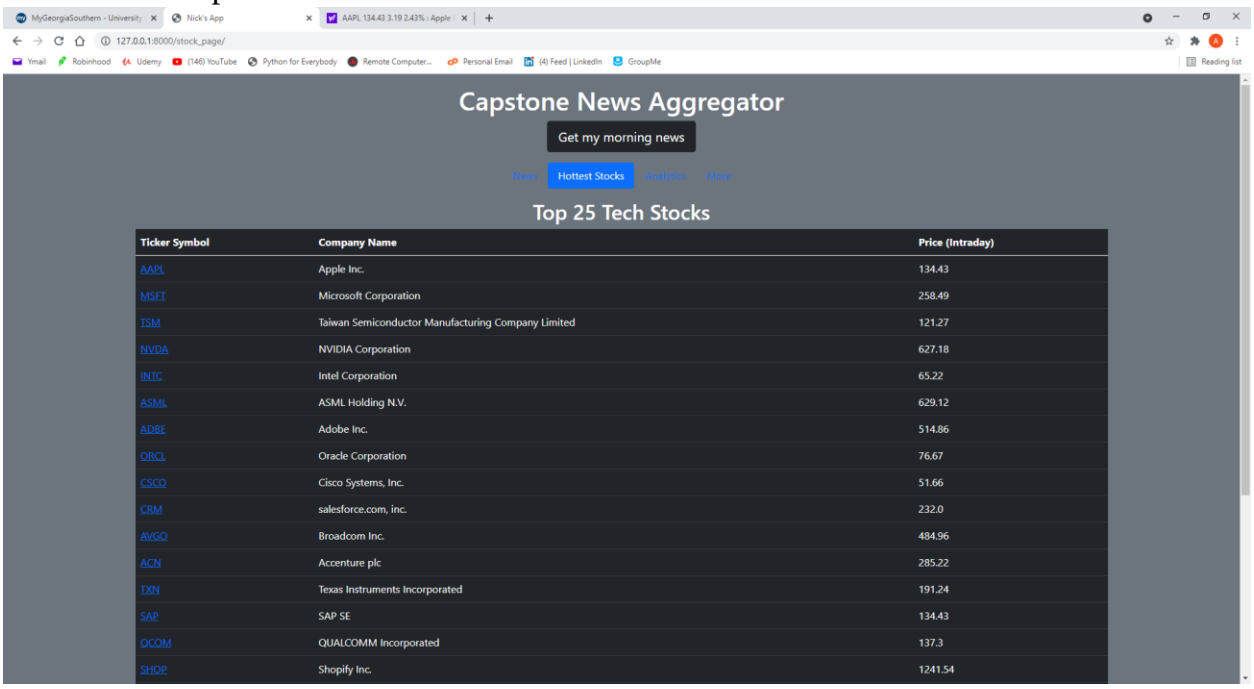


Figure 3.7.1-1 Reducing width of table.

4.8 Final Output



Ticker Symbol	Company Name	Price (Intraday)
AAPL	Apple Inc.	134.43
MSFT	Microsoft Corporation	258.49
TSM	Taiwan Semiconductor Manufacturing Company Limited	121.27
NVDA	NVIDIA Corporation	627.18
INTC	Intel Corporation	65.22
ASML	ASML Holding N.V.	629.12
ADBE	Adobe Inc.	514.86
ORCL	Oracle Corporation	76.67
CSCO	Cisco Systems, Inc.	51.66
CRM	salesforce.com, inc.	232.0
AVGO	Broadcom Inc.	484.96
ACN	Accenture plc	285.22
TXN	Texas Instruments Incorporated	191.24
SAP	SAP SE	134.43
QCOM	QUALCOMM Incorporated	137.3
SHOP	Shopify Inc.	1241.54

Figure 2- Centered and fully formatted table containing stock performance data.

5 Data Vizualization

5.1 Creating the app

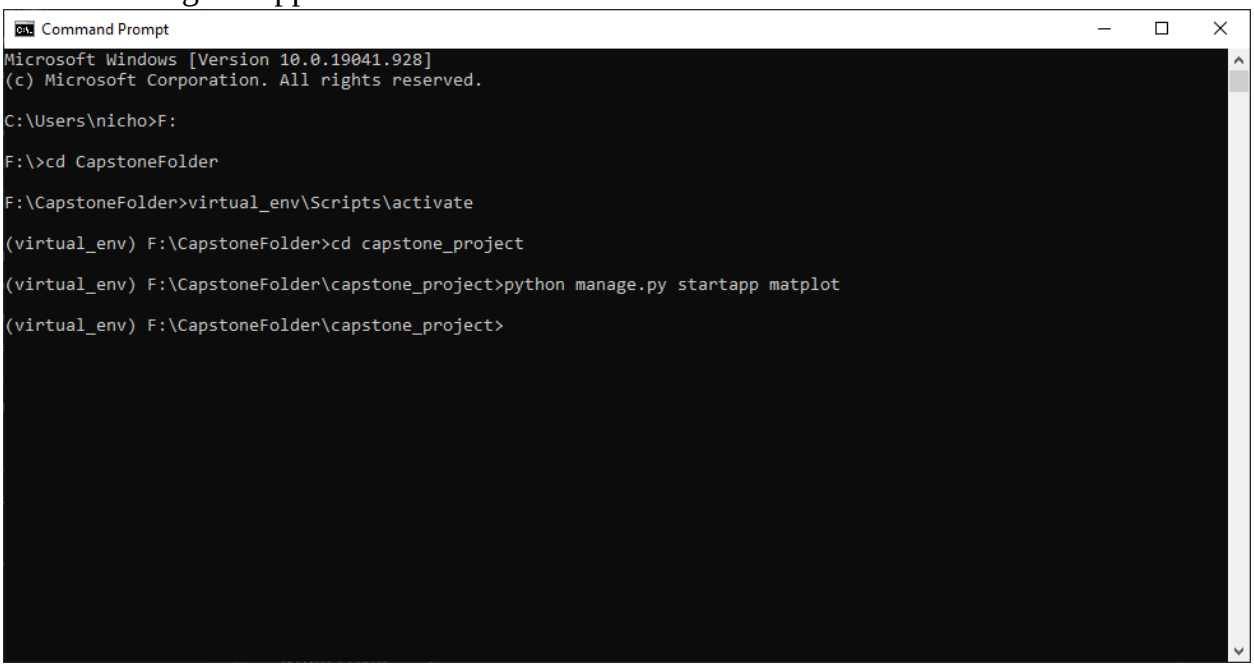


Figure4.1-1 creating matplot application.

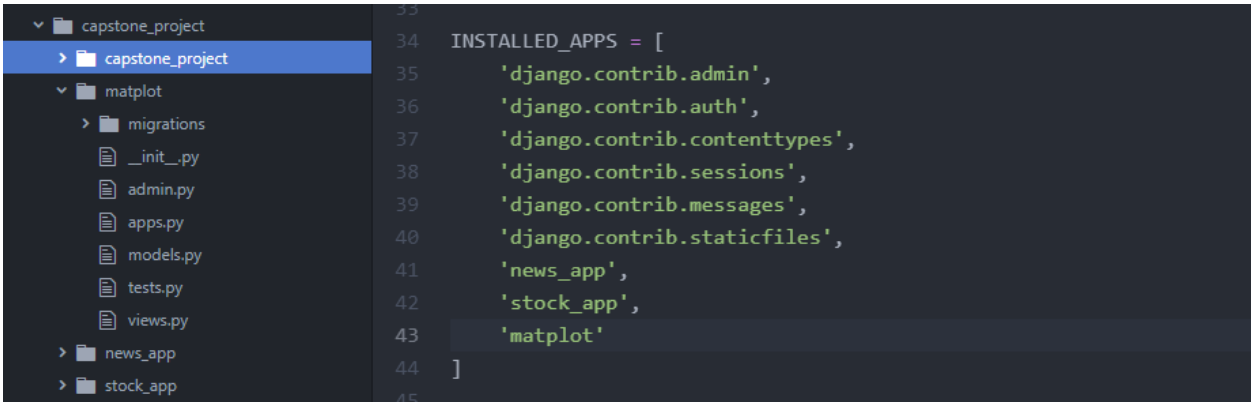


Figure4.1-2 New application directory created & configuration for project settings.

5.2 Introducing Bokeh

The next portion of the project deals with using the python modules "Matplotlib" and "Bokeh" to create detailed graphs and plots. However, another key dependency that will be introduced in this section is "pandas_datareader", which is a sub-package of the pandas library and is used to quickly create a data frame (tabular format) from various internet sources. Although the new application will be named "matplot", I will begin with using Bokeh to create a candlestick charts of the S&P 500 index over various periods of time.

The first step in visualizing any data is collecting the data. Therefore, I will begin by creating the variables to define the time interval of data I want to store for the first graph. Then, assign the "pandas_datareader" function to variable and use the necessary parameters to specify the data source, the ticker symbol, and the start/end dates. The first graph will represent data from the beginning of this year to April 2nd, 2021. I will be testing the graphs on a script stored externally from the rest of my project.

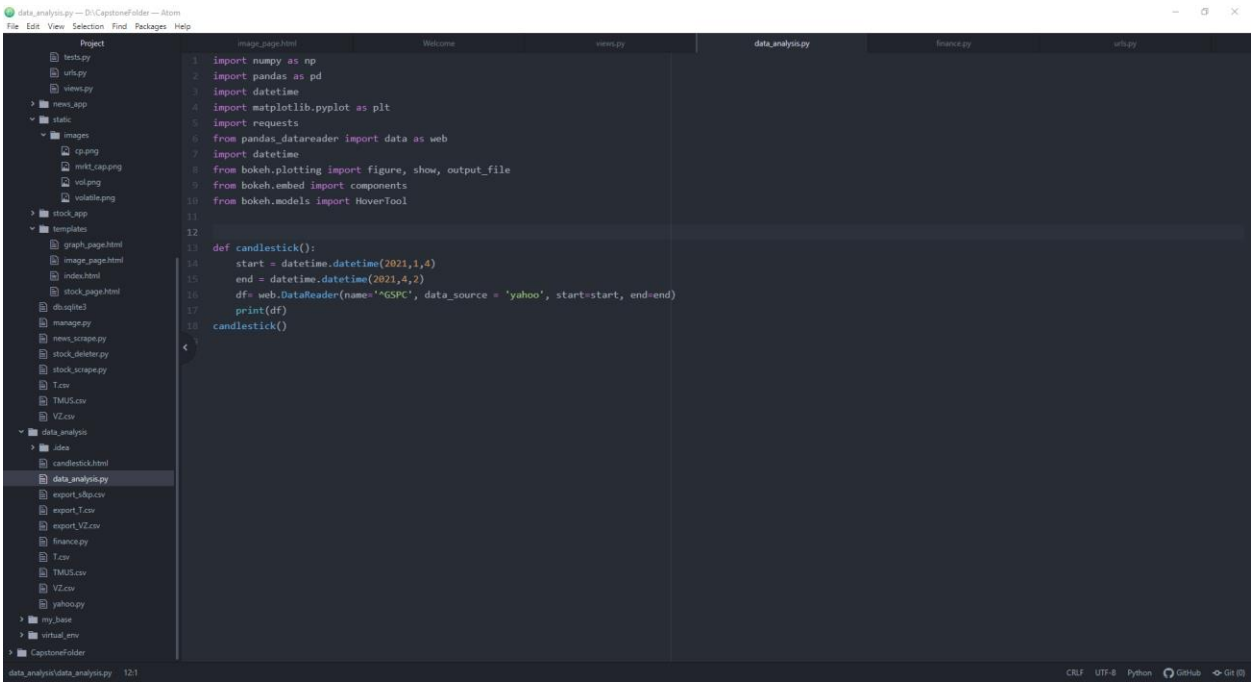


Figure4.2-1 Collecting data for 1/4/2021 through 4/2/2021 of S&P 500 index

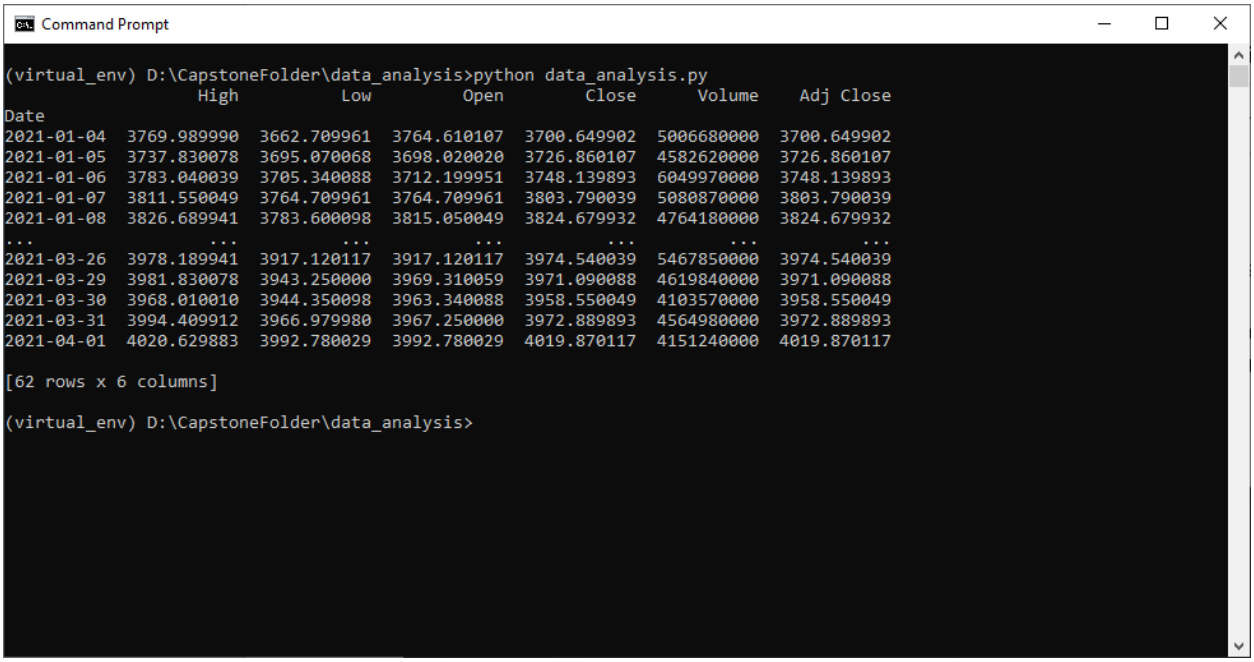


Figure4.2-2 S&P 500 data returned in structured format.

Now that I have the data stored in a structured format, I can access specific rows/columns, as well as add, delete, or manipulate the data. The next process may appear a bit unconventional, however, I will create another function inside the candlestick function shown above. This function will contain to two parameters, one for the closing price, one for the opening price. Based on whether the price of the index went up, down, or stayed the same on any given day, that row will be given a value of increase, decrease, or equal in a new column called “Status”.

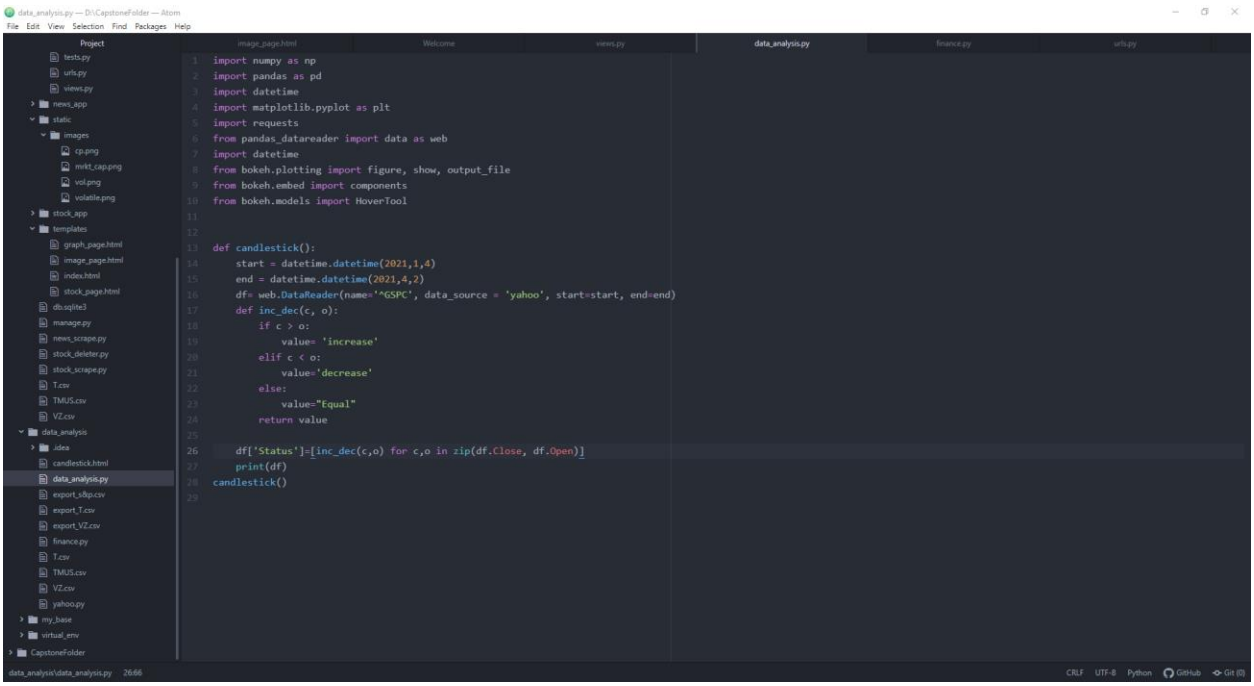


Figure4.2-3 Creating new column for daily return ‘Status’.

In the image above, the last line of code before the print statement initializes a new column and that column's name. Then then "inc_dec" function is called, and the parameters are used by the zip function, which is a powerful python function that allows iteration through two lists at the same time.

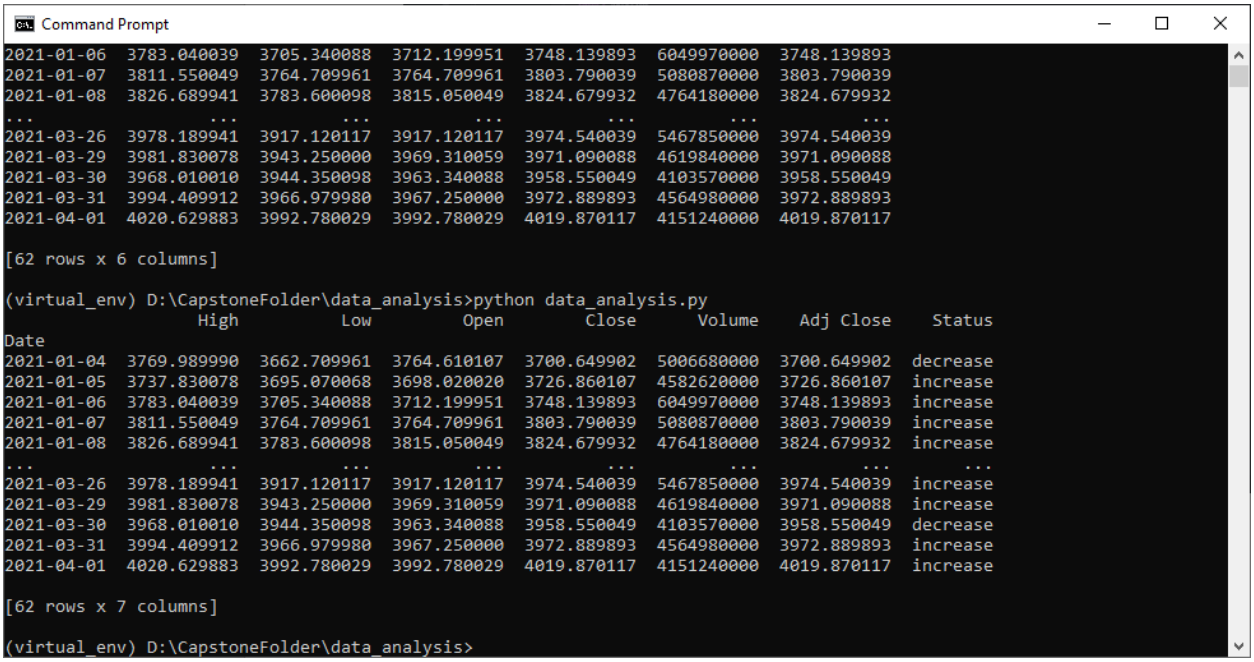
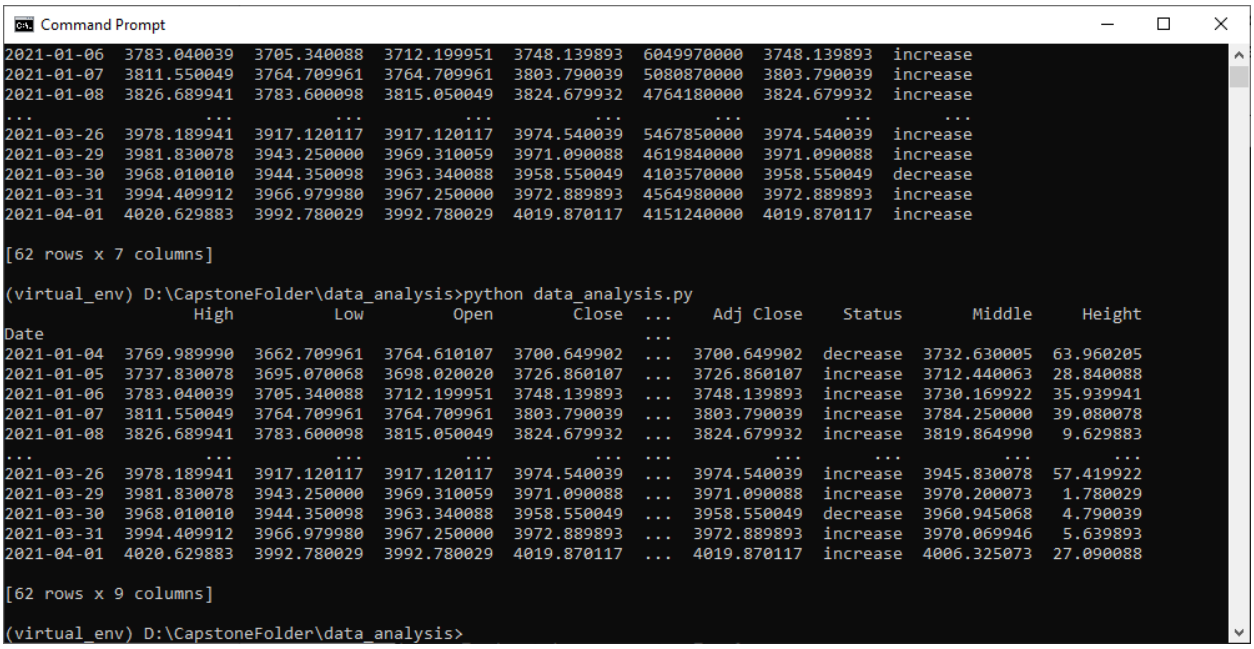


Figure4.2-4 Status column added to data frame.

Next, I will create two more columns to contain the "daily average price" and another to store the amounts of "daily price variation". The average price column name will be "Middle", and the variance will be "Height". The column names are used for clearer context when building the plot begins, which will be demonstrated shortly.

```
25
26      df['Status']=[inc_dec(c,o) for c,o in zip(df.Close, df.Open)]
27      df['Middle']=(df.Open+df.Close)/2
28      df['Height']=abs(df.Open - df.Close)
29      print(df)
30  candlestick()
31
```

Figure4.2-5 Creating columns for daily: average price & price variance



Now that the data has been collected and properly stored, it is time to begin using Bokeh to build the chart.

5.2.1 Building the Chart

To begin, I will define the size of the Bokeh plot and set the x-axis (horizontal) to hold the dates. Next, I will use Bokeh to build rectangular shapes in the graph. There height and width will be determined by the daily values calculated and stored in the 3 new columns. I will create one rectangle for “increases” and one for “decreases” (based on value in ‘Status’ column), and attempt displaying the chart.

```
import numpy as np
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import requests
from pandas_datareader import data as web
import datetime
from bokeh.plotting import figure, show, output_file
from bokeh.embed import components
from bokeh.models import HoverTool


def candlestick():
    start = datetime.datetime(2021,1,4)
    end = datetime.datetime(2021,4,2)
    df= web.DataReader(name='^GSPC', data_source = 'yahoo', start=start, end=end)
    def inc_dec(c, o):
        if c > o:
            value= 'increase'
        elif c < o:
            value='decrease'
        else:
            value="Equal"
        return value

    df['Status']=[inc_dec(c,o) for c,o in zip(df.Close, df.Open)]
    df['Middle']=(df.Open+df.Close)/2
    df['Height']=abs(df.Open - df.Close)

    p=figure(x_axis_type='datetime', width=1000, height=500)
    p.title='S&P 500: Year to Date'
    hours= 12*60*60*1000
    p.rect(df.index[df.Status=='increase'], df.Middle[df.Status=='increase'], hours, df.Height[df.Status=='increase'])
    p.rect(df.index[df.Status=='decrease'], df.Middle[df.Status=='decrease'], hours, df.Height[df.Status=='decrease'])
    show(p)

candlestick()
```

Figure4.2.1-1 creating Bokeh rectangles (components).

Figure4.2.1-2 nondetailed candlestick chart displayed.

Although the graph did display the rectangular shapes and axis values correctly, it does not provide a viewer with very insightful detail. Therefore, I will add colors to the increasing (green) and decreasing (red) rectangles, as well as, add a Bokeh segment (line) to represent Highs and Lows.

```
25
26      df['Status']=[inc_dec(c,o) for c,o in zip(df.Close, df.Open)]
27      df['Middle']=(df.Open+df.Close)/2
28      df['Height']=abs(df.Open - df.Close)
29
30      p=figure(x_axis_type='datetime', width=1000, height=500)
31      p.title='S&P 500: Year to Date'
32      hours= 12*60*60*1000
33
34      p.rect(df.index[df.Status=='increase'], df.Middle[df.Status=='increase'],
35             hours, df.Height[df.Status=='increase'], fill_color='green', line_color='black')
36      p.rect(df.index[df.Status=='decrease'], df.Middle[df.Status=='decrease'],
37             hours, df.Height[df.Status=='decrease'], fill_color='red', line_color='black')
38      p.segment(df.index, df.High, df.index, df.Low, color='black')
39      show(p)
```

Figure4.2.1-3 adding red and green colors & segment lines to represent highs/lows.

Figure4.2.1-4 Output after adding colors and segments.

In the image above, the final output of Bokeh's powerful ability to create detailed, interactive visualizations is displayed. Next, I will use the same process to display the same kind of chart except the time interval will be from the beginning of the Covid Pandemic to April 2nd, 2021 (around a year).

```
40      start = datetime.datetime(2020,3,28)
41      end = datetime.datetime(2021,4,2)
```

Figure4.2.1-5 Defining Time interval for Post-Covid price performance.

Next, I will use the exact same script used to create the previous graph, to create this one. However, keep in mind that the start/end date has been altered.

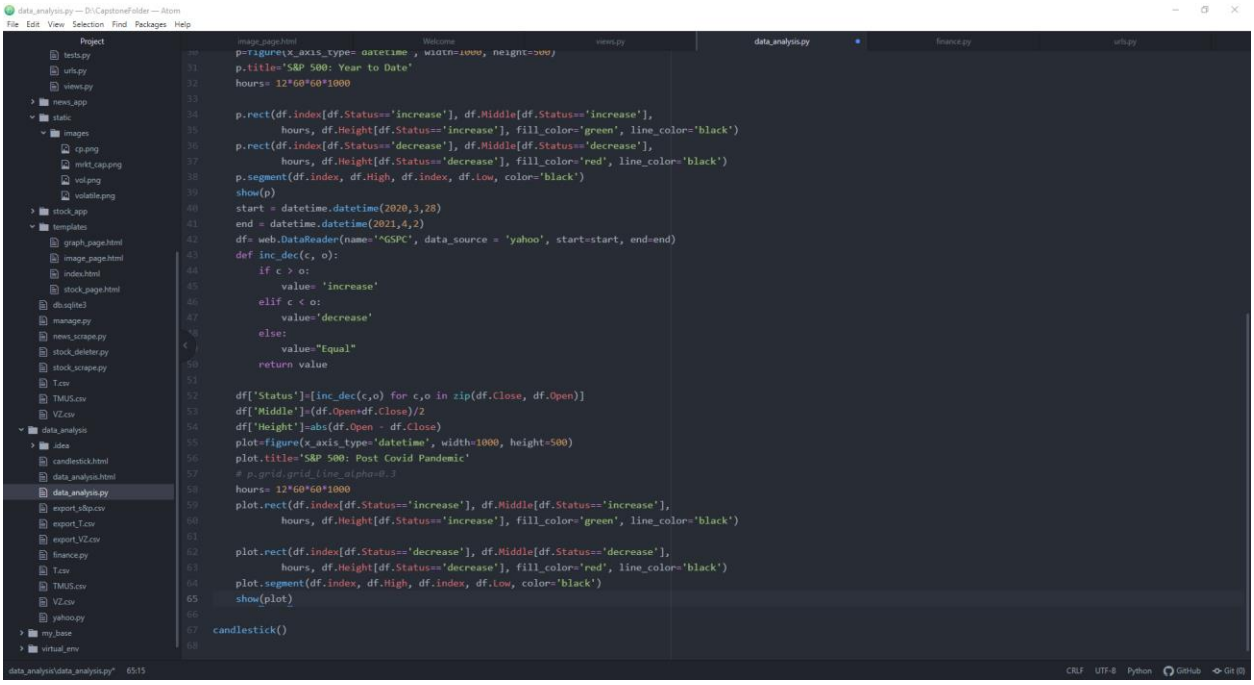


Figure4.2.1-6 Creating the Post-Covid candlestick chart.

Figure4.2.1-7 Post-Covid candlestick chart Output.

The new graph properly displayed with correct dates and prices. However, the candlestick chart may not be the best visualization technique for this type of analysis. While there are normally long bars to

represent the candlesticks, the rectangles almost appear as dots. Luckily, this Bokeh plot has the capability of zooming in, as well as some of other interactive functionalities located in the toolbar along the left side of the chart.

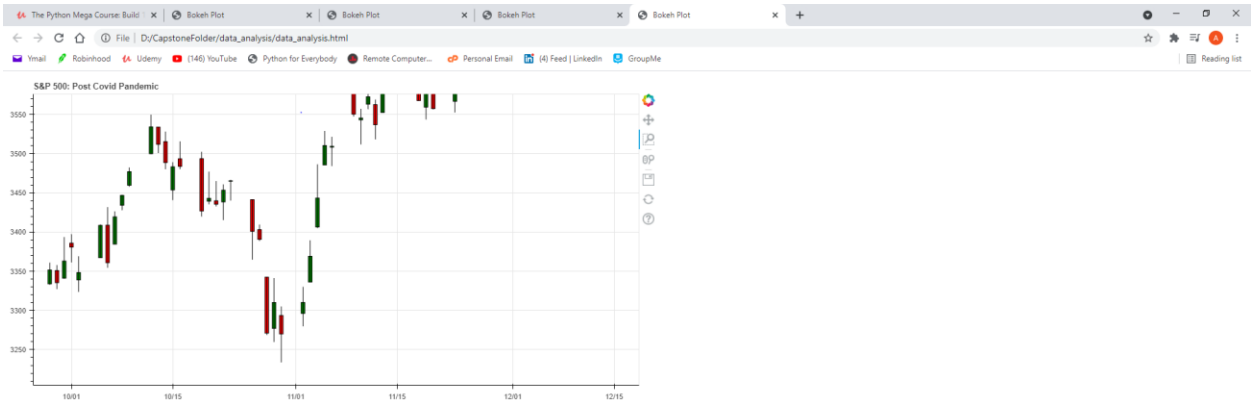


Figure4.2.1-8 Using Bokeh's interactive tools.

Lastly, I will create one more candlestick chart to visualize the S&P 500 index over the past 5 years. I will also use the same process that has been used so far in that I will be changing the start date to begin.

```
66      start = datetime.datetime(2016,4,2)
67      end = datetime.datetime(2021,4,2)
```

Figure4.2.1-9 Defining time interval for 5-year price performance.

```
    start = datetime.datetime(2016,4,2)
    end = datetime.datetime(2021,4,2)
    df= web.DataReader(name='^GSPC', data_source = 'yahoo', start=start, end=end)
    def inc_dec(c, o):
        if c > o:
            value= 'increase'
        elif c < o:
            value='decrease'
        else:
            value="Equal"
        return value

    df['Status']=[inc_dec(c,o) for c,o in zip(df.Close, df.Open)]
    df['Middle']=(df.Open+df.Close)/2
    df['Height']=abs(df.Open - df.Close)
    plots=figure(x_axis_type='datetime', width=1000, height=500)
    plots.title='S&P 500: 2016-present'
    hours= 12*60*60*1000
    plots.rect(df.index[df.Status=='increase'], df.Middle[df.Status=='increase'],
            hours, df.Height[df.Status=='increase'], fill_color='green', line_color='black')

    plots.rect(df.index[df.Status=='decrease'], df.Middle[df.Status=='decrease'],
            hours, df.Height[df.Status=='decrease'], fill_color='red', line_color='black')
    plots.segment(df.index, df.High, df.index, df.Low, color='black')
    show(plots)

candlestick()
```

Figure4.2.1-10 Creating the 5-year period candlestick chart.

Figure4.2.1-11 5-year period candlestick chart Output.

Like the 2nd graph, the candlesticks are not very vivid on such a large time scale, fortunately, all of the graphs have the zoom capability.

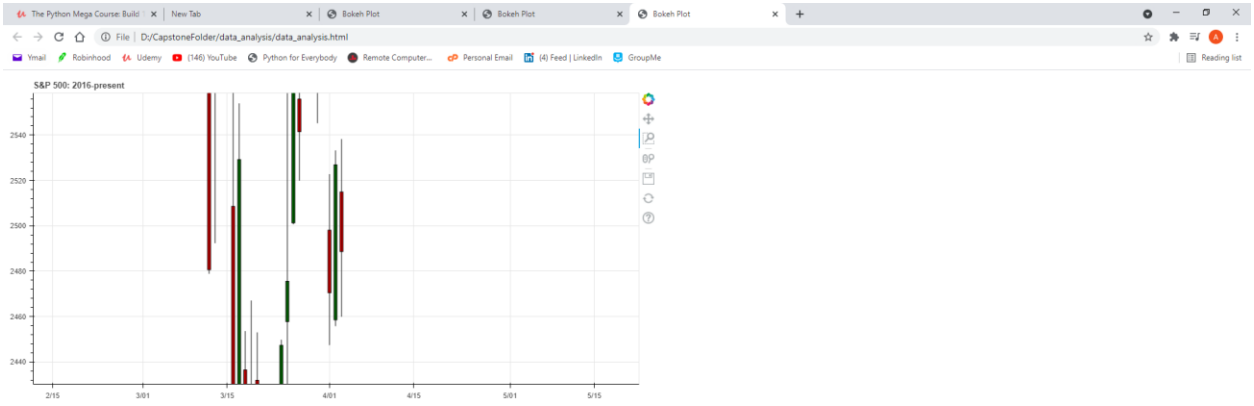
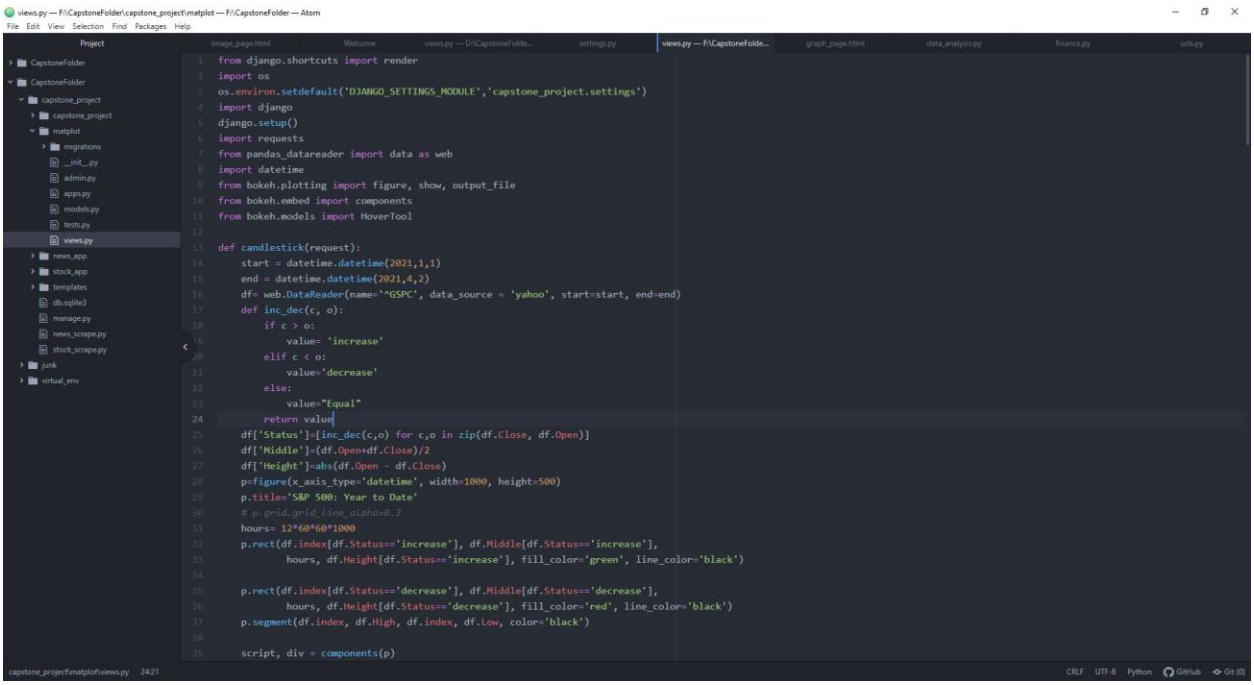


Figure4.2.1-12 Using Bokeh's interactive tools.

5.2.2 Integrating Bokeh & Django

Now that I have created and modified the charts I will place the candlestick function in the "matplot" application's "views.py" script, and create a new "urls.py" file and configure it with the project urls.py file as I have for the past two applications. Then, I will create a new template to display the graphs called, "graph_page.html" and use context processors to inject the chart/chart objects into the new template.



```
from django.shortcuts import render
import os
os.environ.setdefault('DJANGO_SETTINGS_MODULE','capstone_project.settings')
import django
django.setup()
import requests
from pandas_datareader import data as web
import datetime
from bokeh.plotting import figure, show, output_file
from bokeh.embed import components
from bokeh.models import HoverTool

def candlestick(request):
    start = datetime.datetime(2021,1,1)
    end = datetime.datetime(2021,4,2)
    df= web.DataReader(name='^GSPC', data_source = 'yahoo', start=start, end=end)
    def inc_dec(c, o):
        if c > o:
            value= 'increase'
        elif c < o:
            value='decrease'
        else:
            value="Equal"
        return value
    df['Status']=[inc_dec(c,o) for c,o in zip(df.Close, df.Open)]
    df['Middle']=(df.Open+df.Close)/2
    df['Height']=abs(df.Open - df.Close)
    p=figure(x_axis_type='datetime', width=1000, height=500)
    p.title='S&P 500: Year to Date'
    # p.grid.grid_line_alpha=0.3
    hours= 12*60*60*1000
    p.rect(df.index[df.Status=='increase'], df.Middle[df.Status=='increase'],
           hours, df.Height[df.Status=='increase'], fill_color='green', line_color='black')

    p.rect(df.index[df.Status=='decrease'], df.Middle[df.Status=='decrease'],
           hours, df.Height[df.Status=='decrease'], fill_color='red', line_color='black')
    p.segment(df.index, df.High, df.index, df.Low, color='black')

    script, div = components(p)
```

Figure4.2.2-1 Placing "candlestick" function inside matplot/views.py.

At the end of each plot objects ("p"), I will create two new variable for each plot, these variables will contain the "components" function which takes the plot object itself as a parameter. This is the way Bokeh stores and transports objects created in the plot object (rectangles, segments, etc.).

```
    p.rect(df.index[df.Status=='increase'], df.Middle[df.Status=='increase'],
           hours, df.Height[df.Status=='increase'], fill_color='green', line_color='black')

    p.rect(df.index[df.Status=='decrease'], df.Middle[df.Status=='decrease'],
           hours, df.Height[df.Status=='decrease'], fill_color='red', line_color='black')
    p.segment(df.index, df.High, df.index, df.Low, color='black')

    script, div = components(p)
```

Figure4.2.2-2 Creating script, div variables to store plot components.

Next, I will give the last two charts their own set of unique variables to store the components of the plots (similarly to the script, div variables). These variables names will be used as context processors in the template. Therefore, they will all go in the render/request argument along with the new graph template that I will create next.

```
    script1, div1 = components(plots)

    return render(request, 'graph_page.html', {'script':script, 'div':div, 'scripts': scripts, 'divs': divs,'script1':script1, 'div1':div1})
```

Figure4.2.2-3 Rendering all 3 plot objects with context variables in the candlestick function.

5.2.3 Building the template

I will begin by creating a new html file in the templates directory, named “graph_page.html”. After a little research, I found that embedding a Bokeh plot inside HTML code requires a “ ” element in the HTML body and a “<script>” element at the end of the template that contains a JavaScript file. The image below was taken from Bokeh’s official library of documentation (link) and does an excellent job of explaining the method. I will be changing the “x.y.z” at the end of the scripts to my current Bokeh version (2.3.0) and pass the needed context processors.

You can insert or template this script and its companion `<div>` in an HTML document and, when the script executes, your plot replaces the `<div>`.

For this to work you first need to load BokehJS, either locally or from a content delivery network (CDN). To load BokehJS from a CDN, add the following lines to your HTML text or template with the appropriate version replacing the `x.y.z`:

```
<script src="https://cdn.bokeh.org/bokeh/release/bokeh-x.y.z.min.js"
        crossorigin="anonymous"></script>
<script src="https://cdn.bokeh.org/bokeh/release/bokeh-widgets-x.y.z.min.js"
        crossorigin="anonymous"></script>
<script src="https://cdn.bokeh.org/bokeh/release/bokeh-tables-x.y.z.min.js"
        crossorigin="anonymous"></script>
```

Figure4.2.3-1 Bokeh documentation for embedding plots in HTML templates.

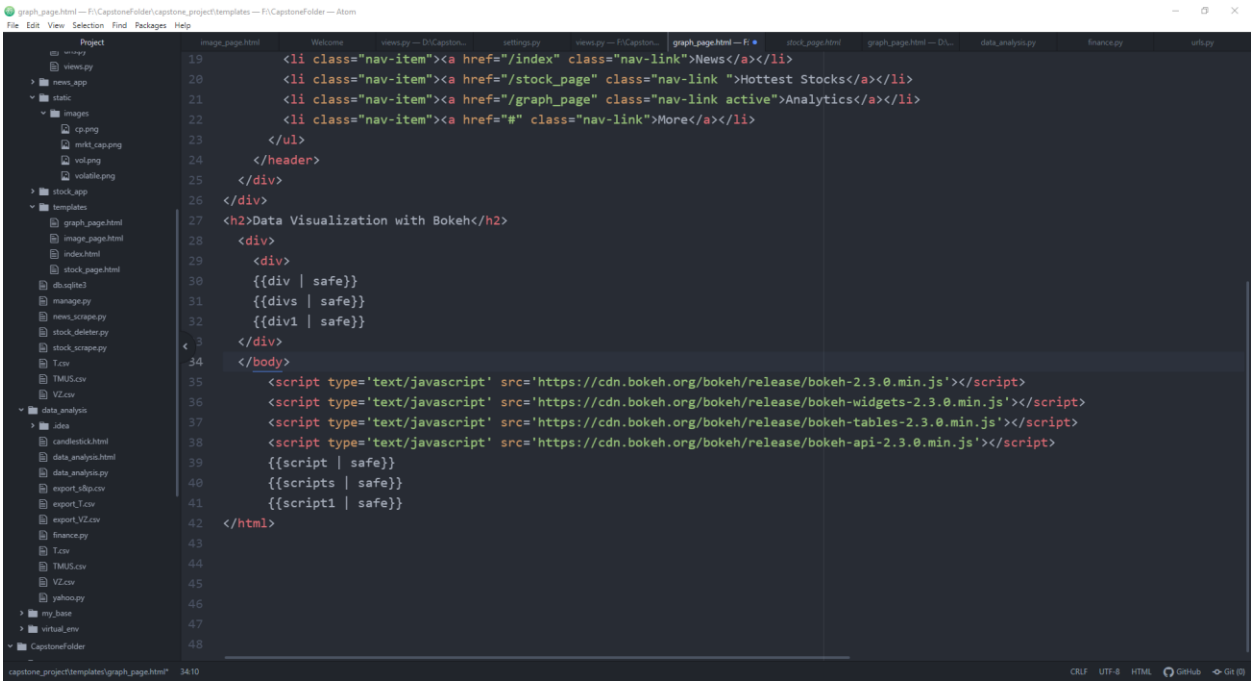


Figure4.2.3-2 Placing script elements & context processors in graph template.

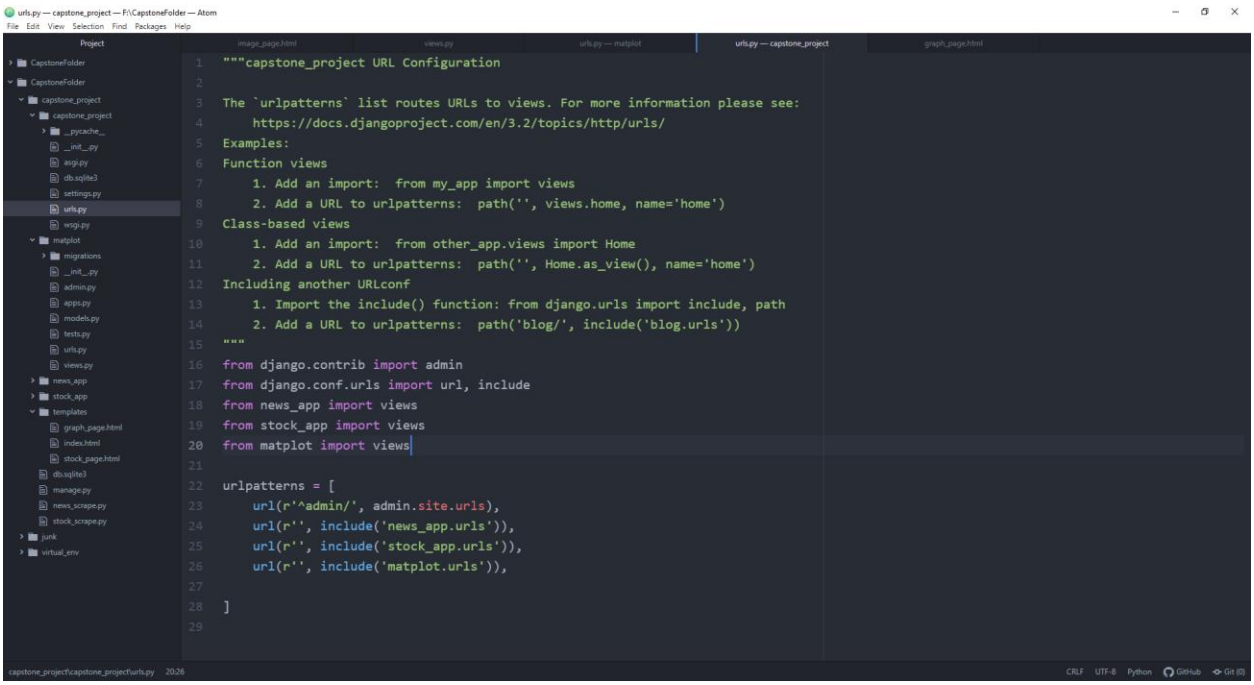


Figure4.2.3-3 Configuring project URLs.

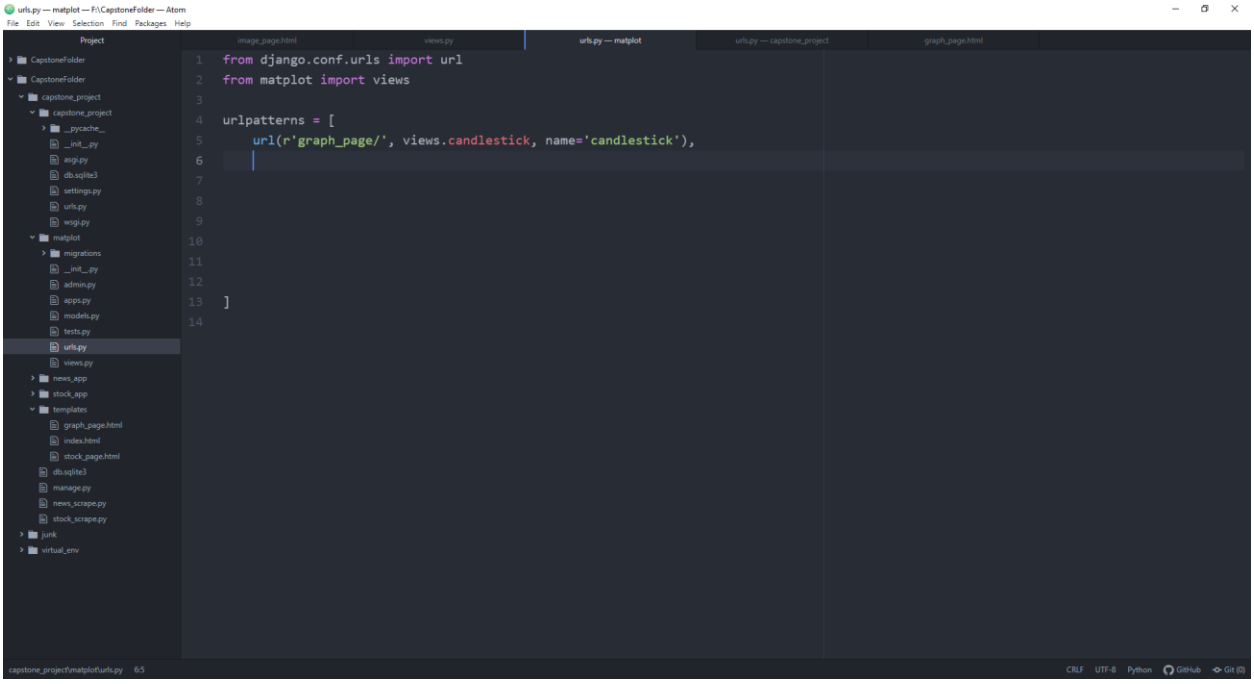


Figure4.2.3-4 Configuring matplot application's URLs.

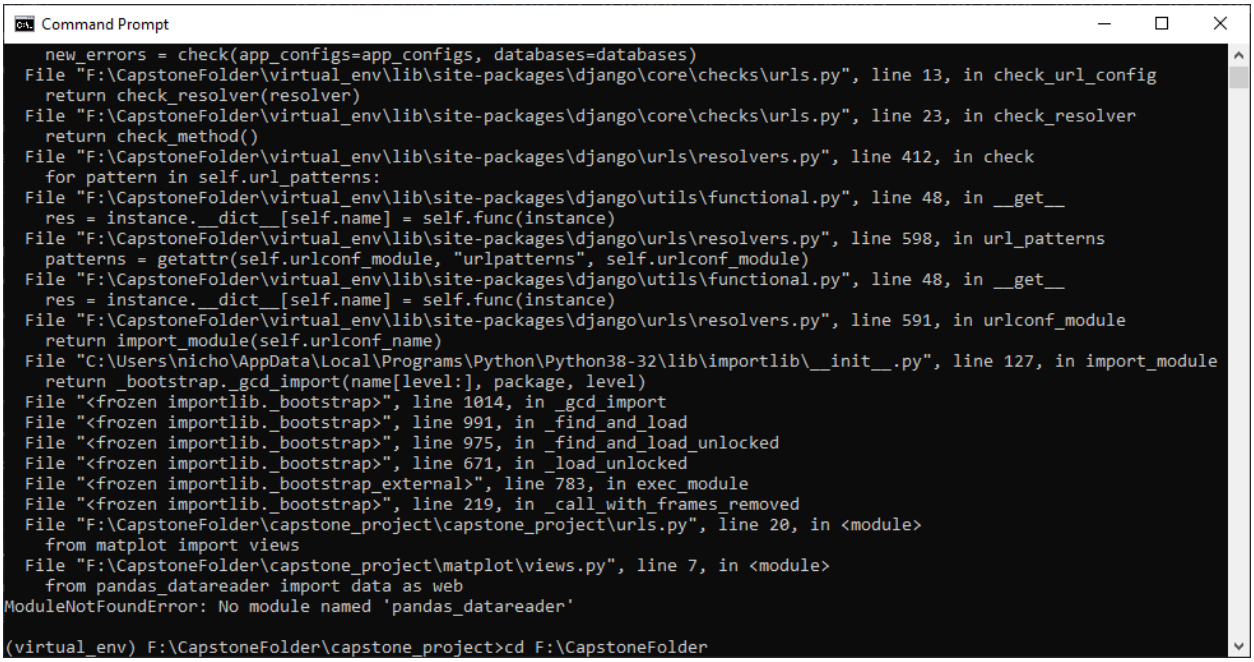


Figure4.2.3-5 ERROR – module not found.

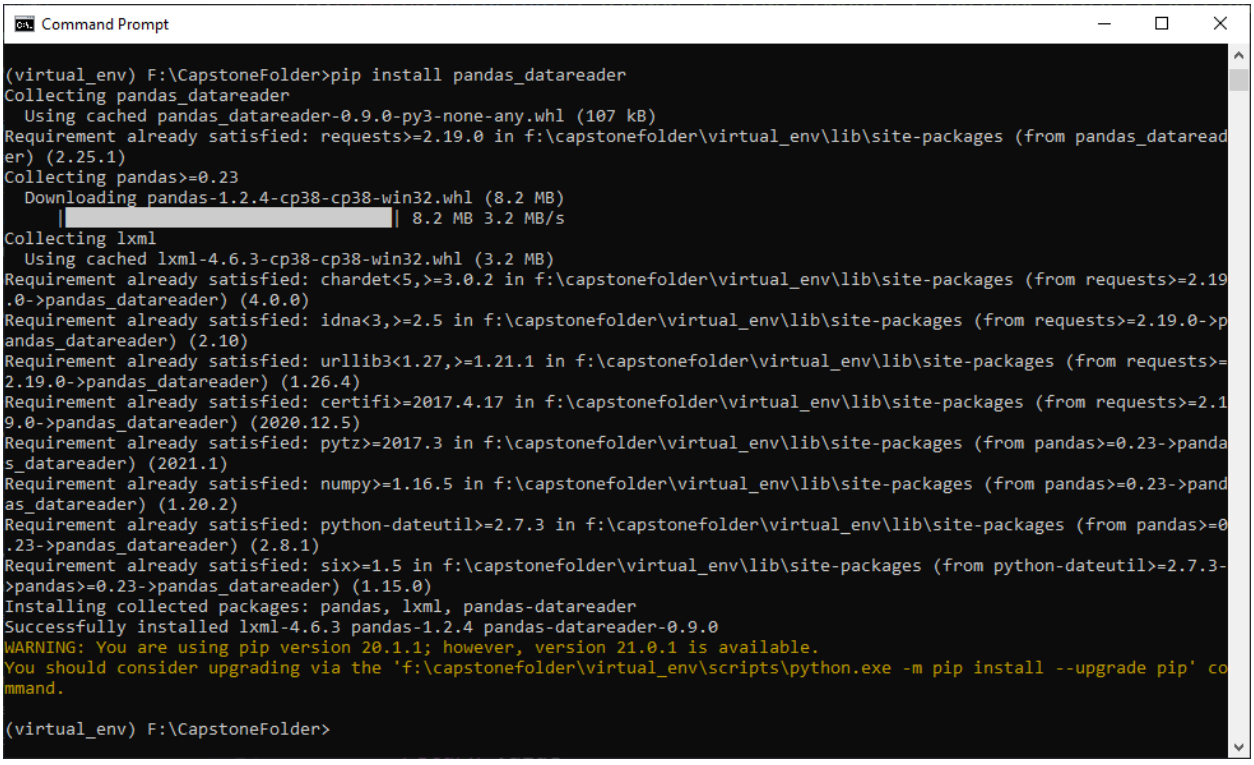


Figure4.2.3-7 Installing panda data-reader dependency.

```
    from matplot import views
  File "F:\CapstoneFolder\capstone_project\matplot\views.py", line 9, in <module>
    from bokeh.plotting import figure, show, output_file
ModuleNotFoundError: No module named 'bokeh'
```

Figure4.2.3-8 ERROR – no module named 'Bokeh'.

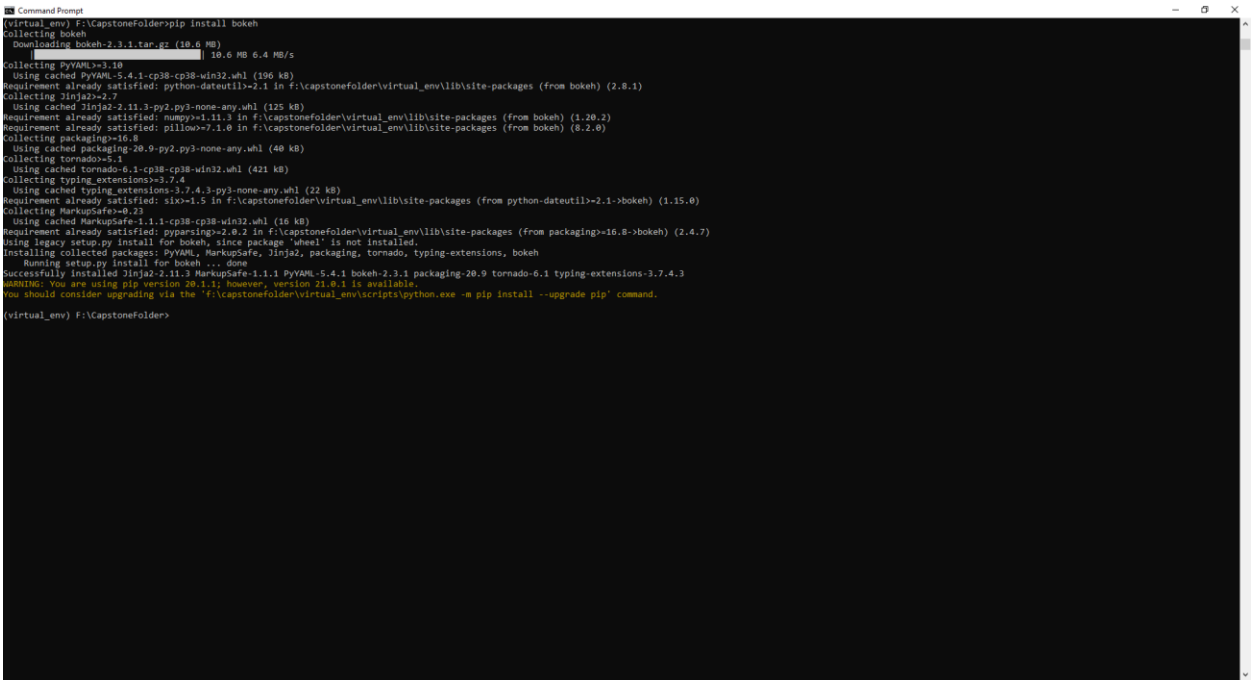


Figure4.2.3-9 Installing Bokeh.

After downloading all the required dependencies, which should have been done at the beginning of this phase, I will restart the server and use this URL to point to the graph template:

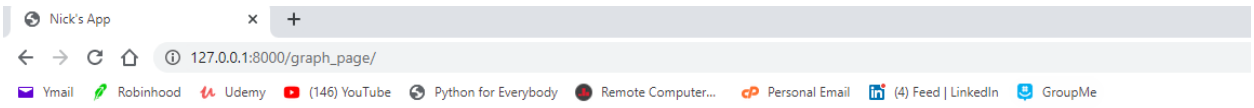


Figure4.2.3-10 Requesting the graph template.

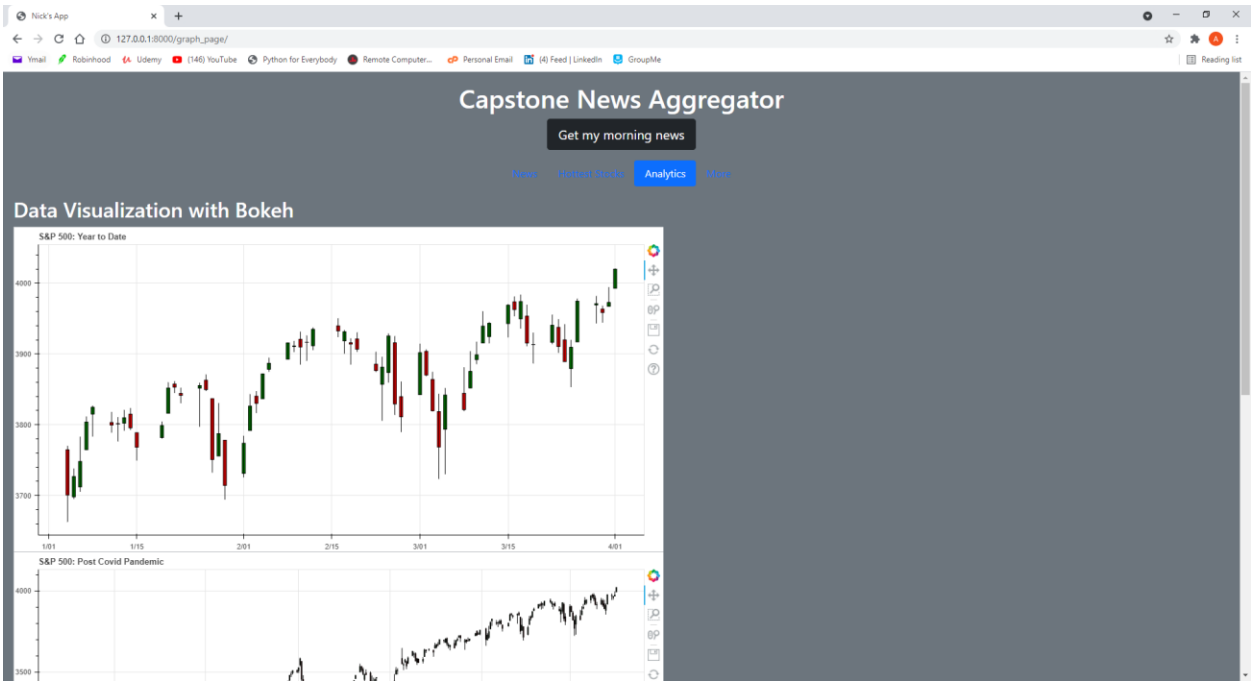


Figure4.2.3-11 Bokeh charts displayed in browser.

Figure4.2.3-11 Bokeh charts displayed in browser cont'd.

Since all the graphs were displayed correctly. The last thing I need to do is edit the other templates to configure the navbar, then center the new page elements (Bokeh plots).

5.3 Final Output

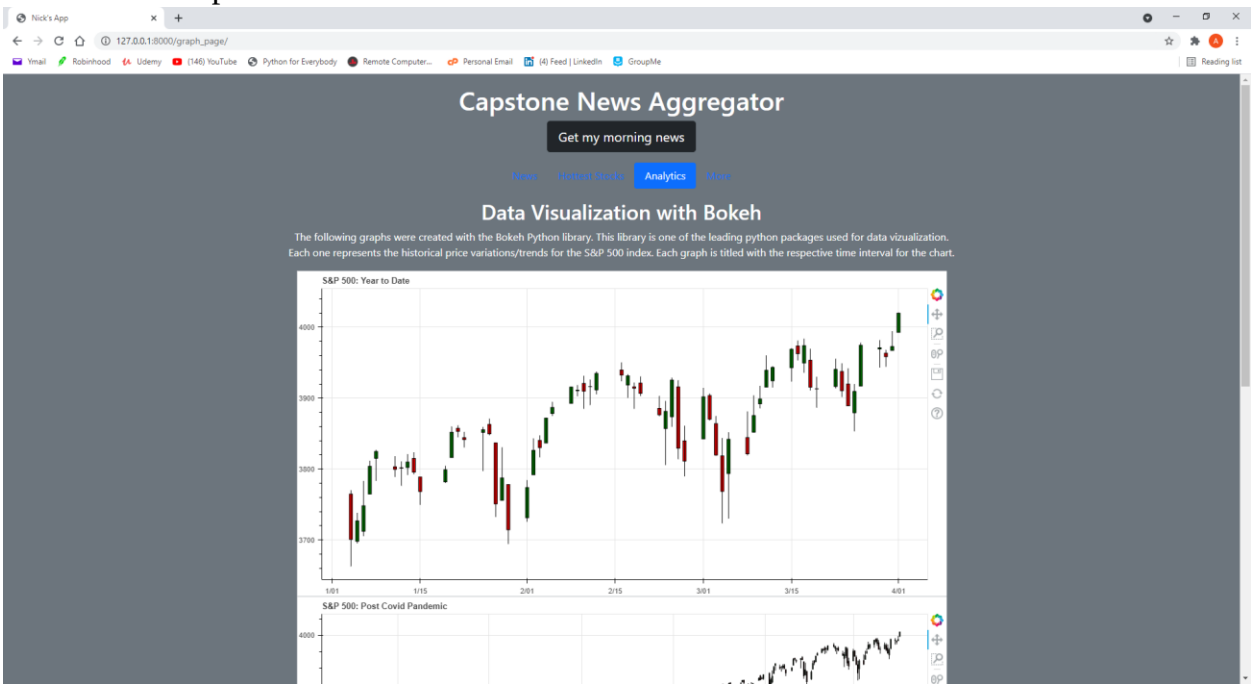


Figure4.3-1 Centered and fully formatted Bokeh candlestick charts.

5.4 Matplotlib & Pandas

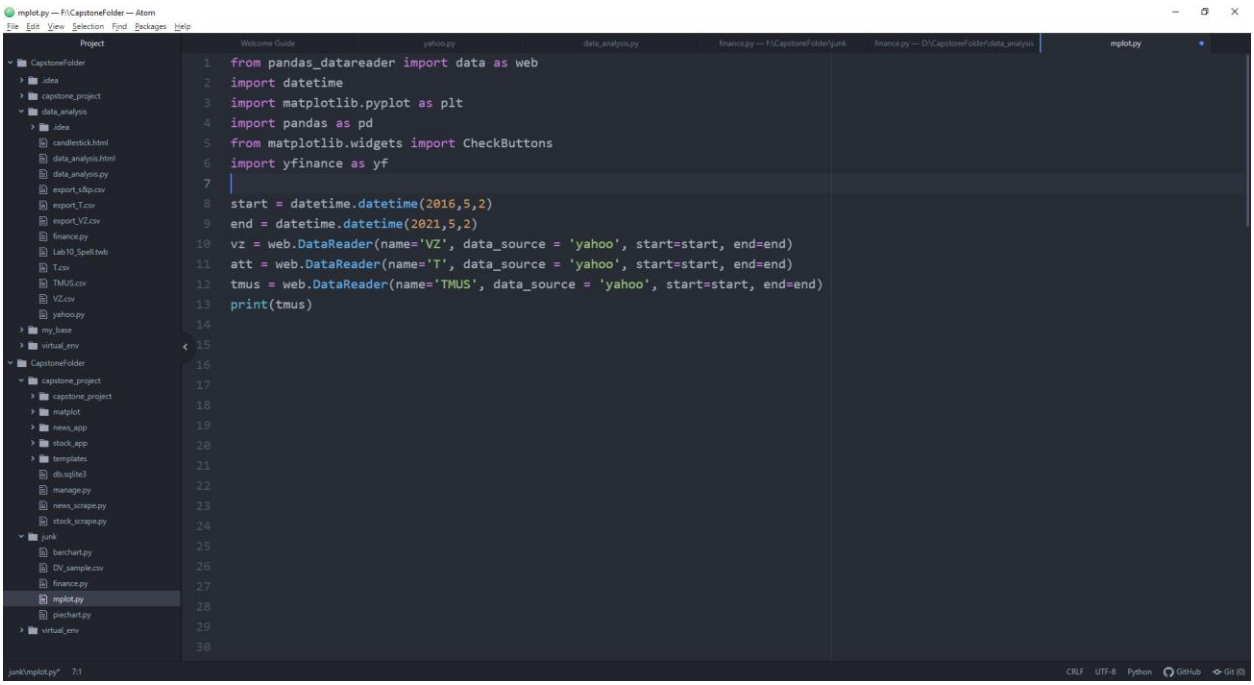
Like Bokeh, Matplotlib is one of the leading Python libraries for data visualization. Therefore, majority of the coding and techniques used is this section will be very similar to those in creating a Bokeh plot. However, I will introduce a few new dependencies that will be needed to successfully collect and graph the data.

5.4.1 Context for the Analysis

I will be analyzing Verizon, AT&T, and T-Mobile by using various *price evaluation techniques* and analyzing graphical representations for certain trends.

5.4.2 Collecting the data

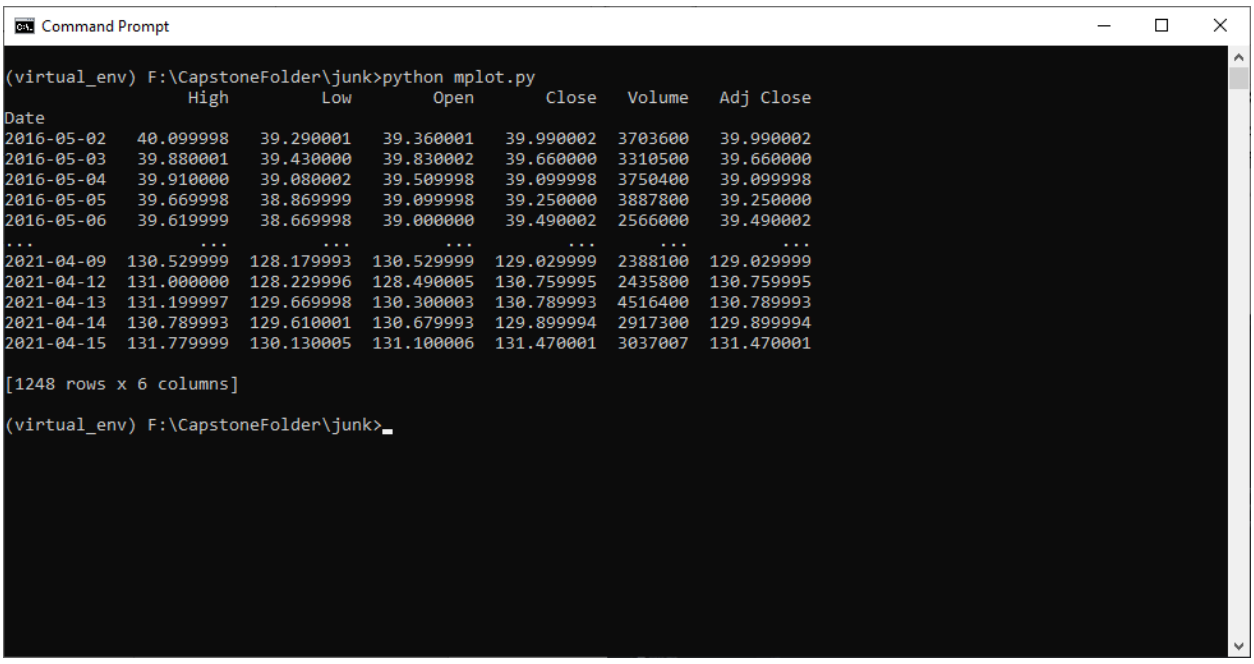
To begin, I will be using relatively the same technique to collect the data as I did for the S&P 500. The first set of charts will be based on 5 years of historical data. However, the "pandas_data reader" library will not be sufficient for finding all the financial values I will need (metrics excluding open, close, low, high, and volume). Therefore, I will be introducing the "yfinance" library, which is a powerful, convenient Python tool that gives access to a very wide range of financial data/statistics for any publicly traded security on finance.yahoo.com. However, this module will not be utilized until later in this section. For now, the following screenshots illustrate the same procedure/syntax I used in collecting the data for the Bokeh plots with "pandas_datareader". Also, I created a new file for this script stored outside of the Django project directory for testing purposes.



```
from pandas_datareader import data as web
import datetime
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.widgets import CheckButtons
import yfinance as yf

start = datetime.datetime(2016,5,2)
end = datetime.datetime(2021,5,2)
vz = web.DataReader(name='VZ', data_source = 'yahoo', start=start, end=end)
att = web.DataReader(name='T', data_source = 'yahoo', start=start, end=end)
tmus = web.DataReader(name='TMUS', data_source = 'yahoo', start=start, end=end)
print(tmus)
```

Figure4.4.2-1 Collecting the data for all 3 network carriers.



```
Command Prompt

(virtual_env) F:\CapstoneFolder\junk>python mplot.py
                High         Low        Open       Close   Volume   Adj Close
Date
2016-05-02   40.099998   39.290001   39.360001   39.990002  3703600   39.990002
2016-05-03   39.880001   39.430000   39.830002   39.660000  3310500   39.660000
2016-05-04   39.910000   39.080002   39.509998   39.099998  3750400   39.099998
2016-05-05   39.669998   38.869999   39.099998   39.250000  3887800   39.250000
2016-05-06   39.619999   38.669998   39.000000   39.490002  2566000   39.490002
...                ...         ...         ...         ...      ...         ...
2021-04-09  130.529999  128.179993  130.529999  129.029999  2388100  129.029999
2021-04-12  131.000000  128.229996  128.490005  130.759995  2435800  130.759995
2021-04-13  131.199997  129.669998  130.300003  130.789993  4516400  130.789993
2021-04-14  130.789993  129.610001  130.679993  129.899994  2917300  129.899994
2021-04-15  131.779999  130.130005  131.100006  131.470001  3037007  131.470001

[1248 rows x 6 columns]

(virtual_env) F:\CapstoneFolder\junk>
```

Figure4.4.2-2 Output for printing the T-Mobile data frame.

Now that the data has been collected and stored in a structred dataframe. I will first use the “closing prices” to create a line graph that illustrates each company’s price trend over the past five years.

5.4.3 Building the Charts

Inside the new “closing_prices” function, I used basic matplotlib methods to create a line graph showing all 3 companies closing price over a 5-year period.

```
13
14  def closing_prices():
15      vz['Close'].plot(label='Verizon')
16      att['Close'].plot(label='AT&T')
17      tmus['Close'].plot(label='T-Mobile')
18      plt.title('Closing Prices')
19      plt.legend()
20      plt.show()
21  closing_prices()
22
23
```

Figure4.4.3-1 Creating line graph for closing prices.

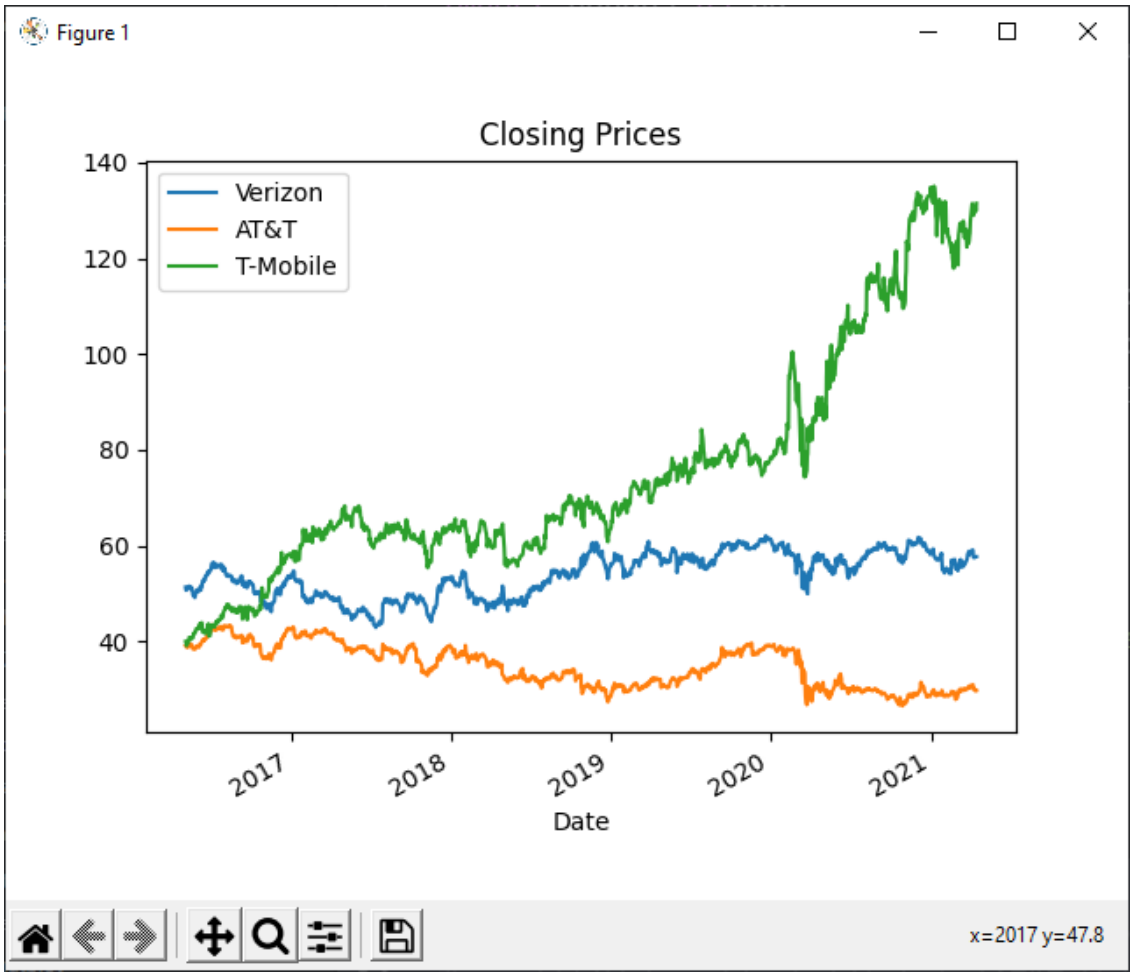


Figure4.4.3-2 Closing Prices Output

Next, I will do the same thing for the following graph; however, I will be using the stock's "volume" as the measure, rather than closing price. A stock's volume is a representation of the total amount of stocks/securities traded (bought/sold) over a period. In this case, we will be using the daily volume over the 5-year period.

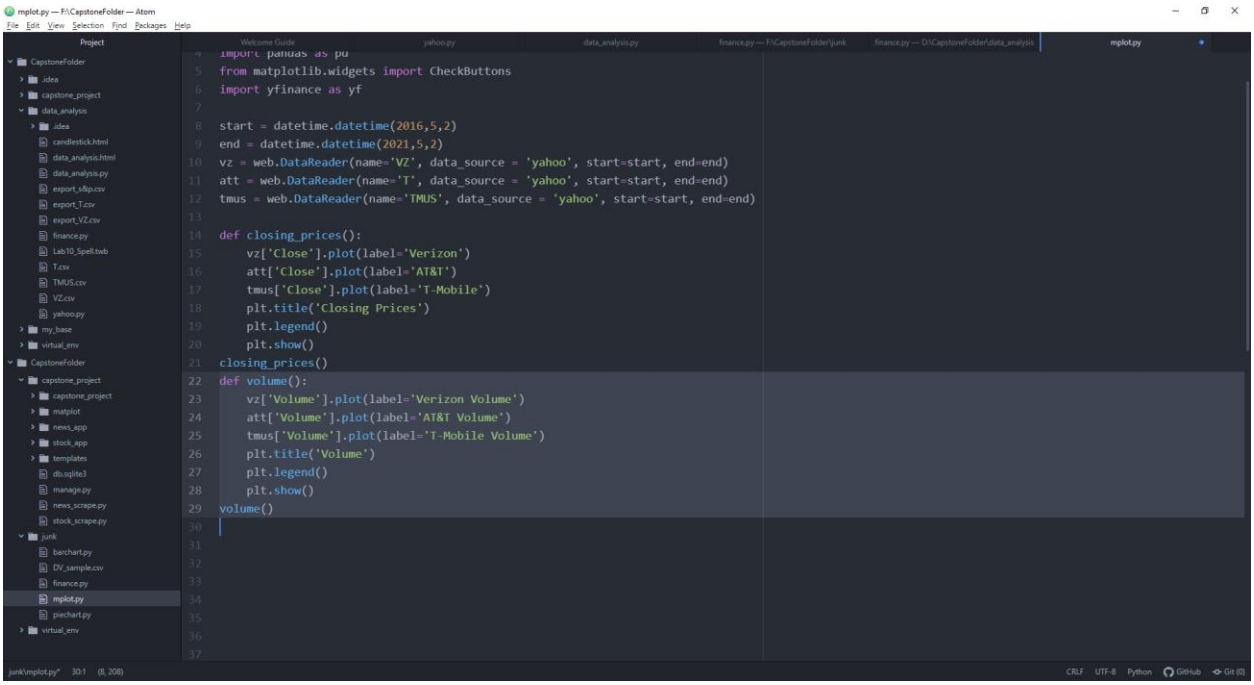


Figure4.4.3-3 Creating line graph for volume.

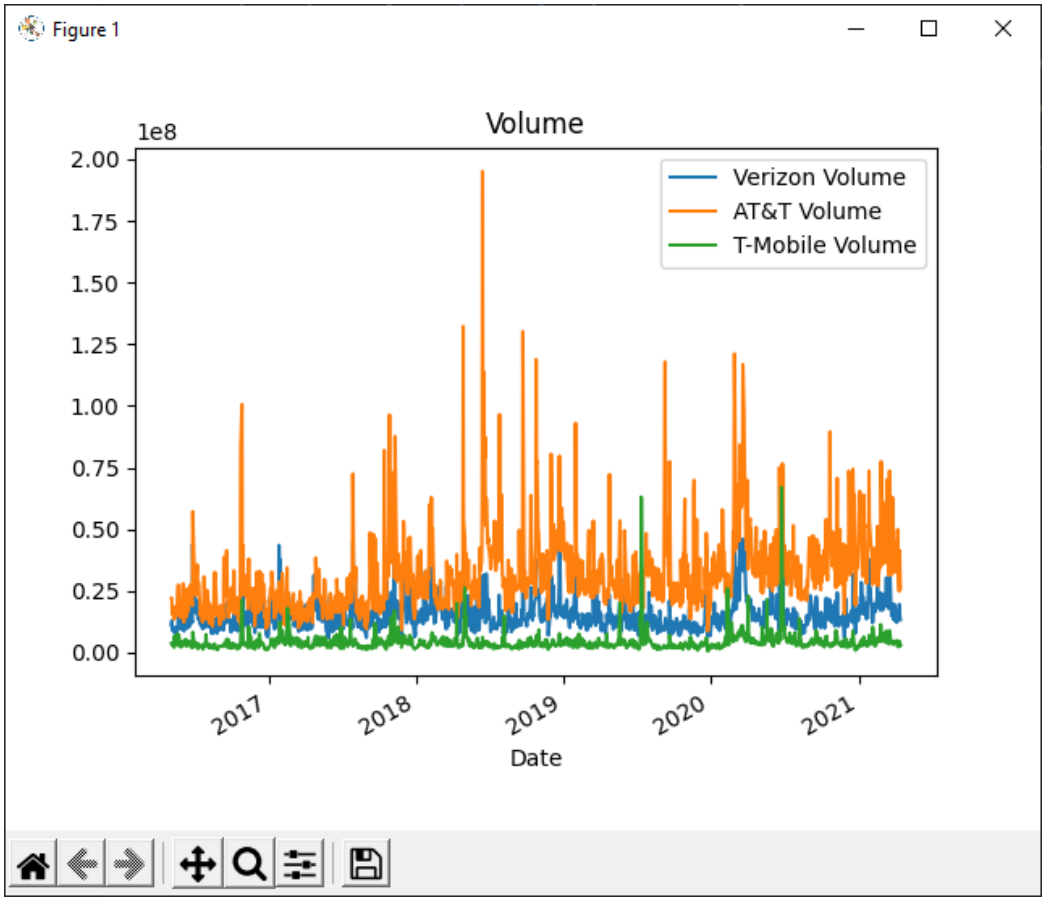


Figure4.4.3-4 Volume Output

Next, I will create a histogram measuring “volatility”. Volatility is calculated by using the daily returns of each company. Then, the return values can be used to indicate a measure of variance or change in the company's stock price over time. This allows us to assess the amount of risk buying a stock may impose based on the likelihood of changing prices slightly or drastically.

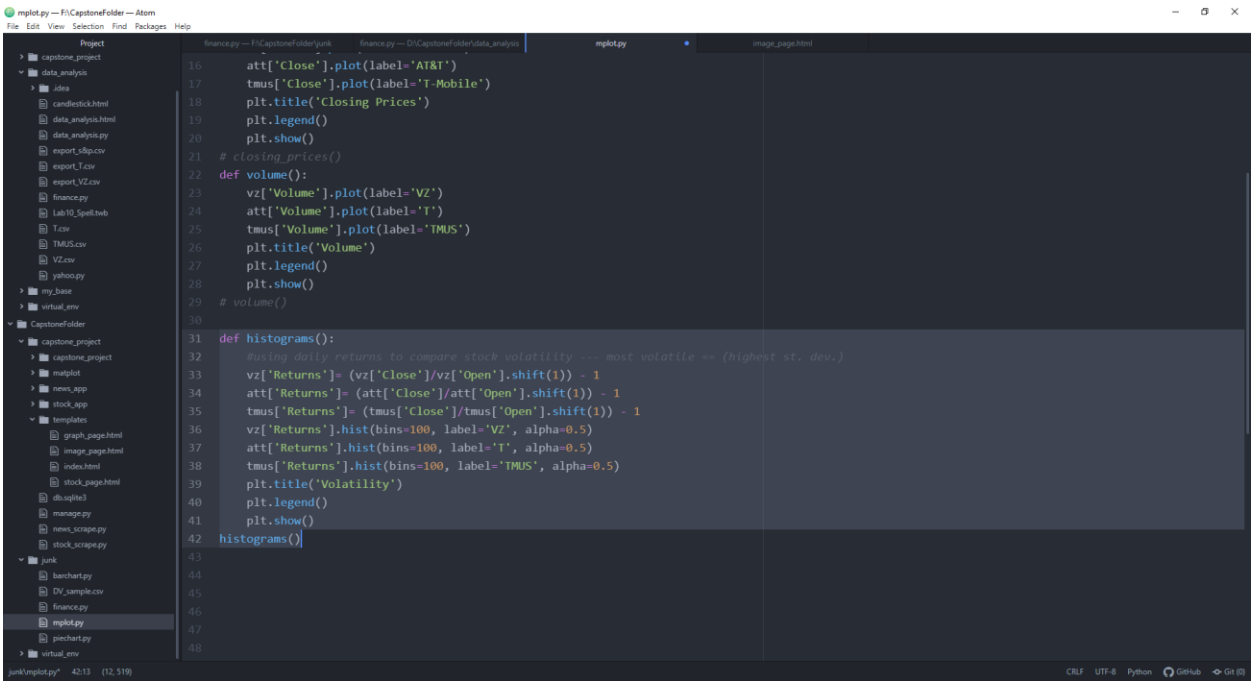


Figure4.4.3-5 Creating histogram for Volatility.

In the first three lines of code inside the “histograms” function displayed above, I am creating a new column in each data frame that calculates daily returns based off *the absolute value in the difference of the open/close prices* of that day. Next, I use basic matplotlib syntax to create a histogram representing the new columns.

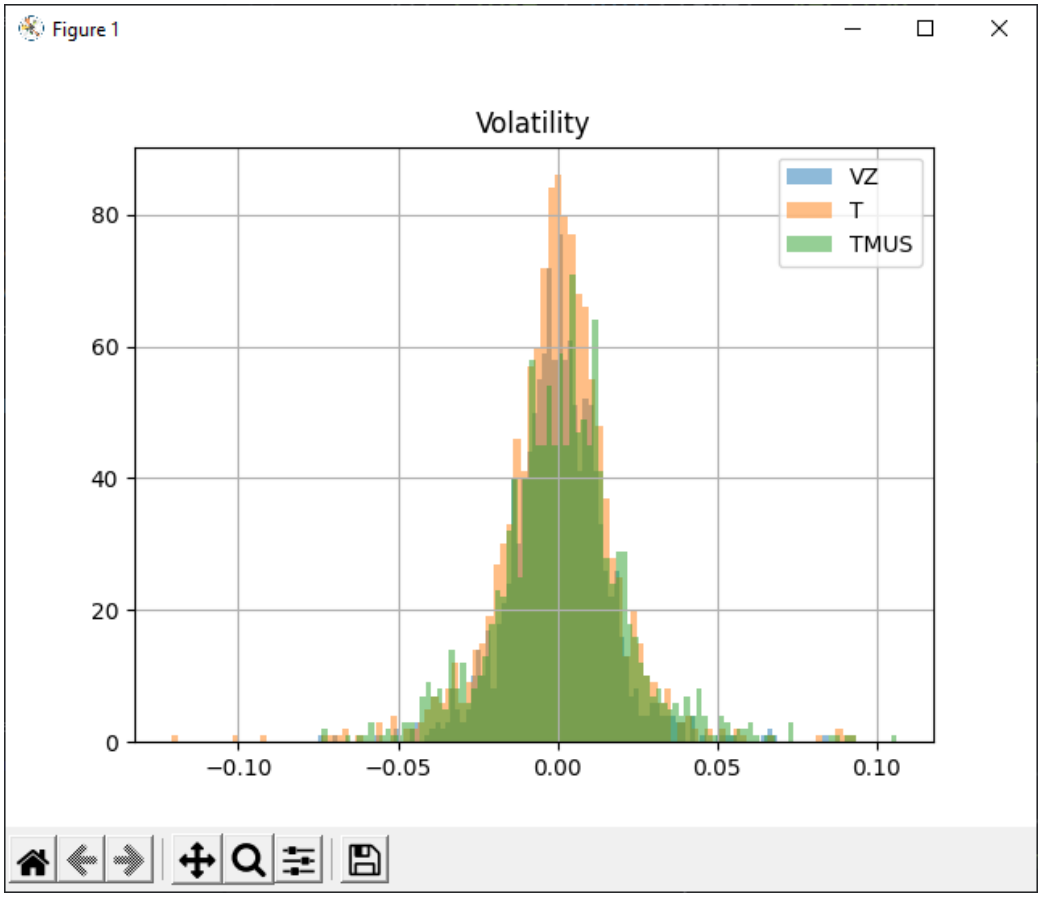


Figure4.4.3-6 Volatility Output

The last graph I will create using the 5-year period sets of data, is another histogram that illustrates each company's market capitalization. Market capitalization refers to the total value of all a company's stock shares. It is calculated by multiplying the price of an individual stock by its total number of outstanding shares. The calculations for market capitalization are demonstrated in the first 3 lines of code inside the new "market_cap" function displayed below. Lastly, I used the same plotting technique to create this histogram as I used for the "volatility" graph.

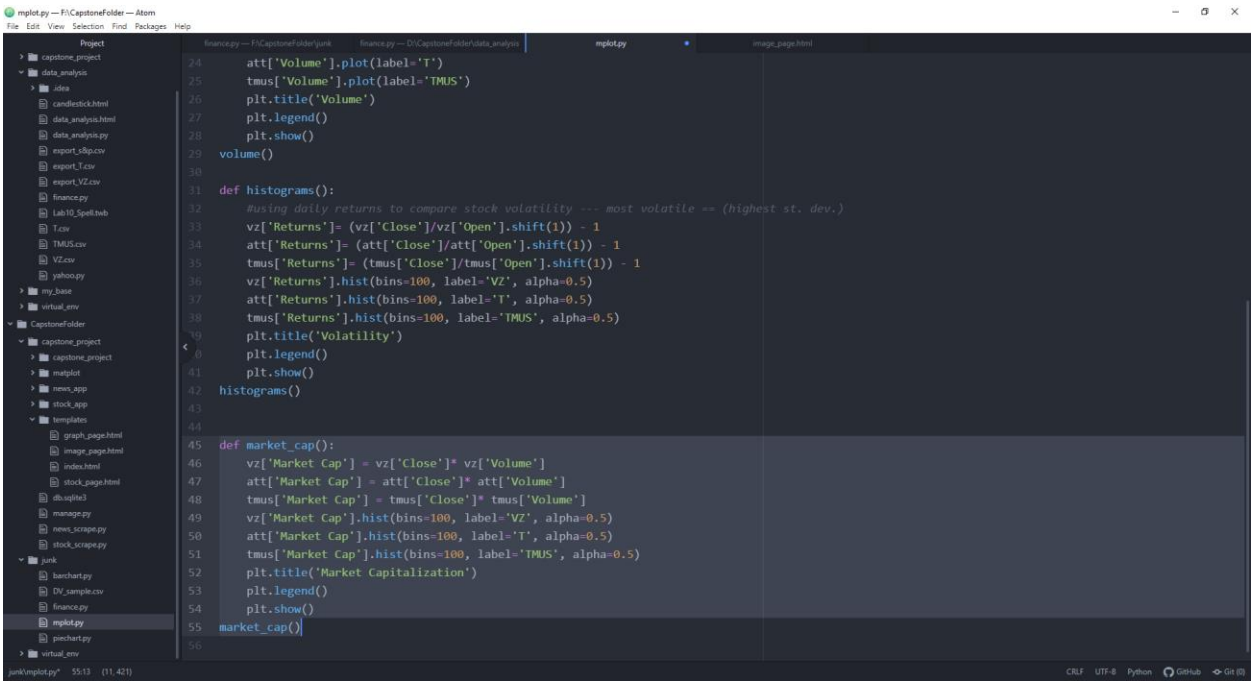


Figure4.4.3-7 Creating histogram for market capitalization.

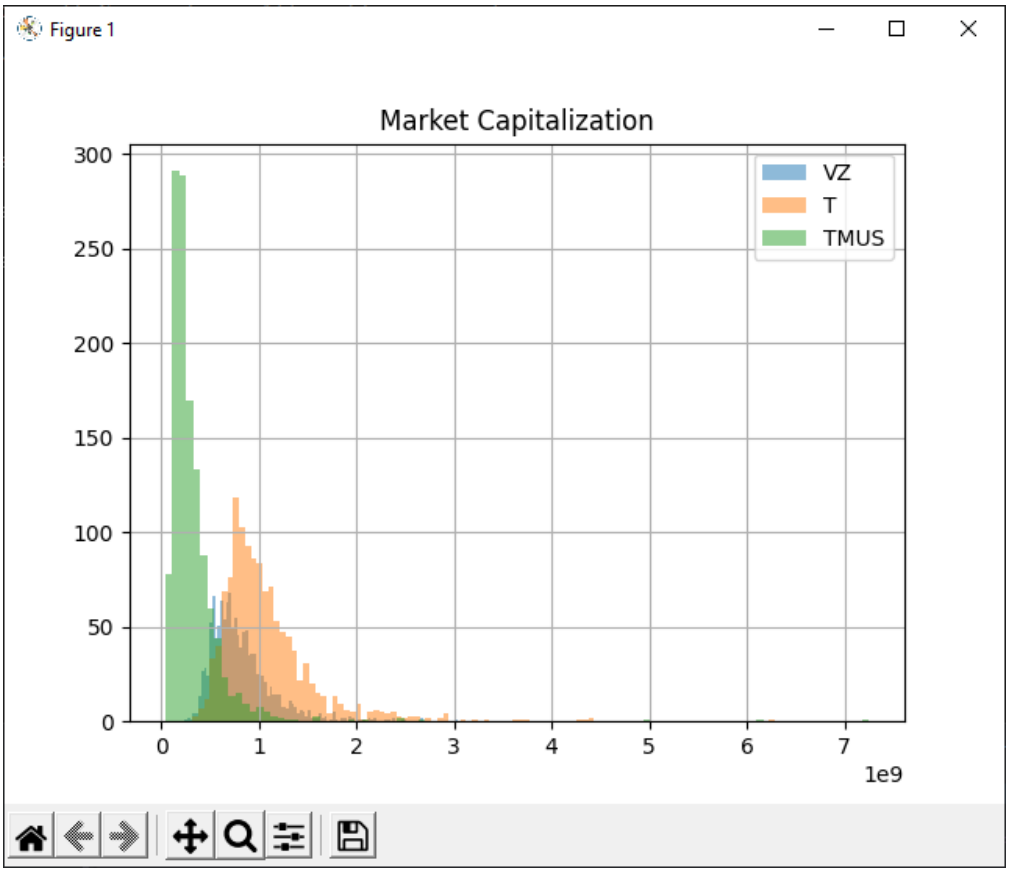


Figure4.4.3-8 Market Capitalization Output.

5.5 Matplotlib & Yfinance

Previously in this section and the last, I have been creating data visualizations with the assistance of the "pandas_datareader". This worked flawlessly for the purpose I needed for, however, to take a step further in the analysis, I will need a new library that gives accesses to more complex financial data (besides open, close, high, volume, etc.). This library is known as, "Yfinance". It is a popular open-source library to access the financial data available on Yahoo Finance.

5.5.1 Collecting the data

First, I will use the "info" method to show the general capabilities of this new library. When, the code is run, a large dictionary is returned containing a wide range of financial information for that company.

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  import yfinance as yf
4
5  vz = yf.Ticker('VZ').info
6  print(vz)
7
8
```

Figure4.5.1-1 Testing the info method for Verizon's financial data.

Figure4.5.1-2 info method output.

Next, I will use a for loop to obtain the "info" for all three companies and append the data to a new list.

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  import yfinance as yf
4
5  tickers = ['VZ','T','TMUS']
6  infos = []
7  for x in tickers:
8      infos.append(yf.Ticker(x).info)
9
```

Figure4.5.1-3 Storing each company's financial information in a list.

Once this list has been created, I will define the values I want to obtain from the information in a new list. Although, we are not using the "pandas_datareader" sub-package in this section, we are using the "pandas" parent Python library to create data frames with the data collected using "Yfinance". Once the data frame is created, I will set the index equal to the ticker symbols to make each row identifiable. The last line of code before the print statement, defines the columns of the data frame as the values in the "evals" list – the screenshot below illustrates this process and its output.

```python
tickers = ['VZ','T','TMUS']
infos = []
for x in tickers:
    infos.append(yf.Ticker(x).info)

evals = ['dividendYield', 'marketCap', 'beta', 'forwardPE']
df = pd.DataFrame(infos)
x=df.set_index('symbol')
x=x[x.columns[x.columns.isin(evals)]]
print(x)
```

Figure4.5.1-4 Creating new columns to store evaluation values for each company.

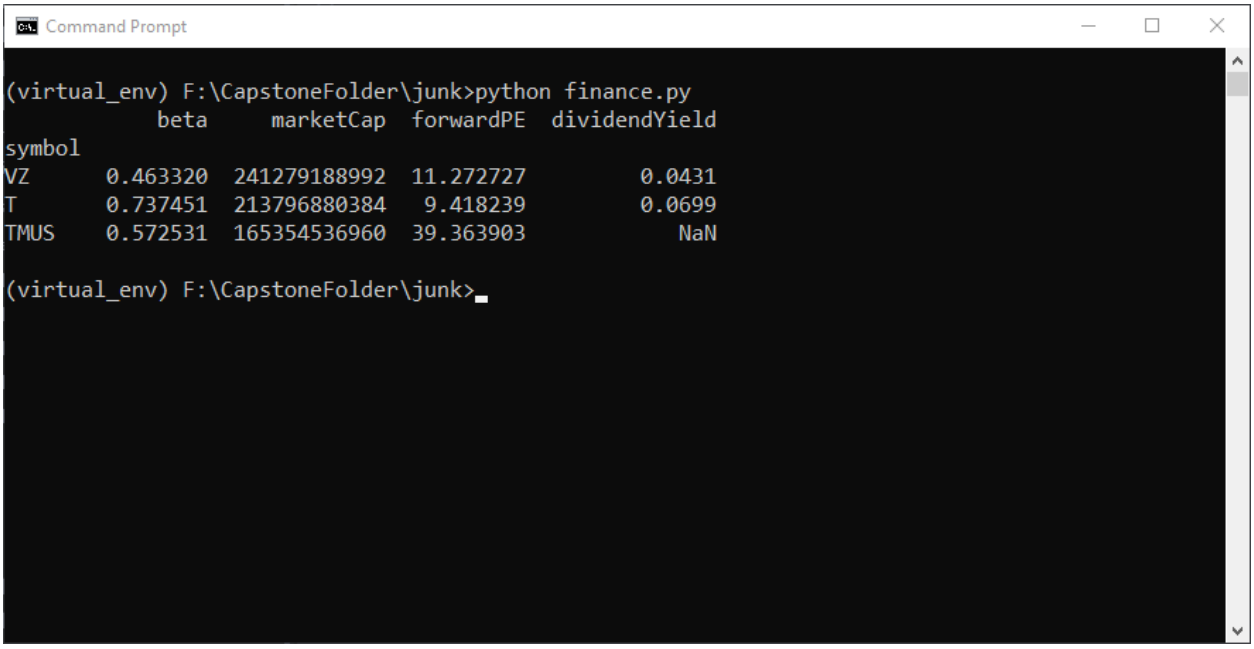


Figure4.5.1-5 Output for new data frame.

5.5.2 Building the charts

Now that the data is collected it is time to put Matplolib to work. Since I will be representing/comparing numerical values rather than analyzing change over time, I will use simple bar charts to visualize the data. Therefore, most of the work was completed by just collecting the data and storing it in a structured format. First, I will be creating a bar chart for the *price-to-earnings ratio* of each company. To do this, I will just need to set the x-axis to show the tickers and the y-axis to measure the values. P/E ratio is one of the most widely used metrics for investors and analysts to determine stock valuation. In addition to showing whether a company's stock price is overvalued or undervalued, the P/E can reveal how a stock's valuation compares to its industry group or a benchmark like the S&P 500 index.

```
14
15  pe = plt.bar(x.index,x.forwardPE, color=('k','orange','cyan'))
16  pe = plt.title('Price-to-Earning Ratio')
17  pe = plt.show()
18
19
```

Figure4.5.2-1 Creating bar chart for P/E ratio.

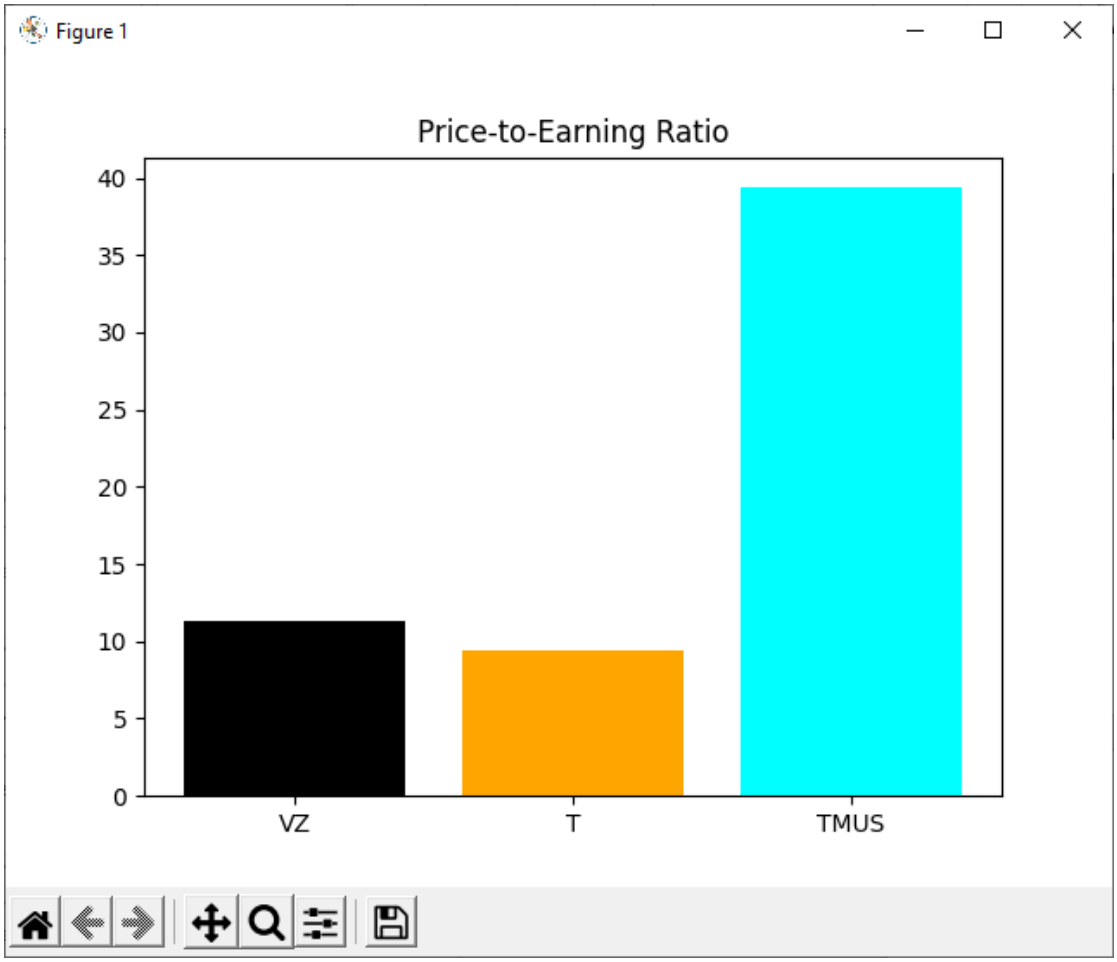


Figure4.5.2-2 P/E ratio Output.

Next, I will use the same process to create similar charts to represent each company's dividend yield, beta, and create another market capitalization chart using more accurate data.

```
18
19  beta = plt.bar(x.index,x.beta, color=('k','orange','cyan'))
20  beta = plt.title('Beta')
21  beta = plt.show()
22
23  mc = plt.bar(x.index,x.marketCap, color=('k','orange','cyan'))
24  mc = plt.title('Market Capitalization')
25  mc = plt.show()
26
27  divd = plt.bar(x.index,x.dividendYield, color=('k','orange','cyan'))
28  divd = plt.title('Dividend Yield')
29  divd = plt.show()
```

Figure4.5.2-3 Creating bar charts for Beta, Dividend Yield, & Market Capitalization

Beta is a measure of a stock's volatility in relation to the overall market (in this case the S&P 500, since all 3 companies are in this index). If a stock moves less than the market, the stock's beta is less than 1.0. High-beta stocks are supposed to be riskier but provide higher return potential; low-beta stocks pose less risk but also lower returns.

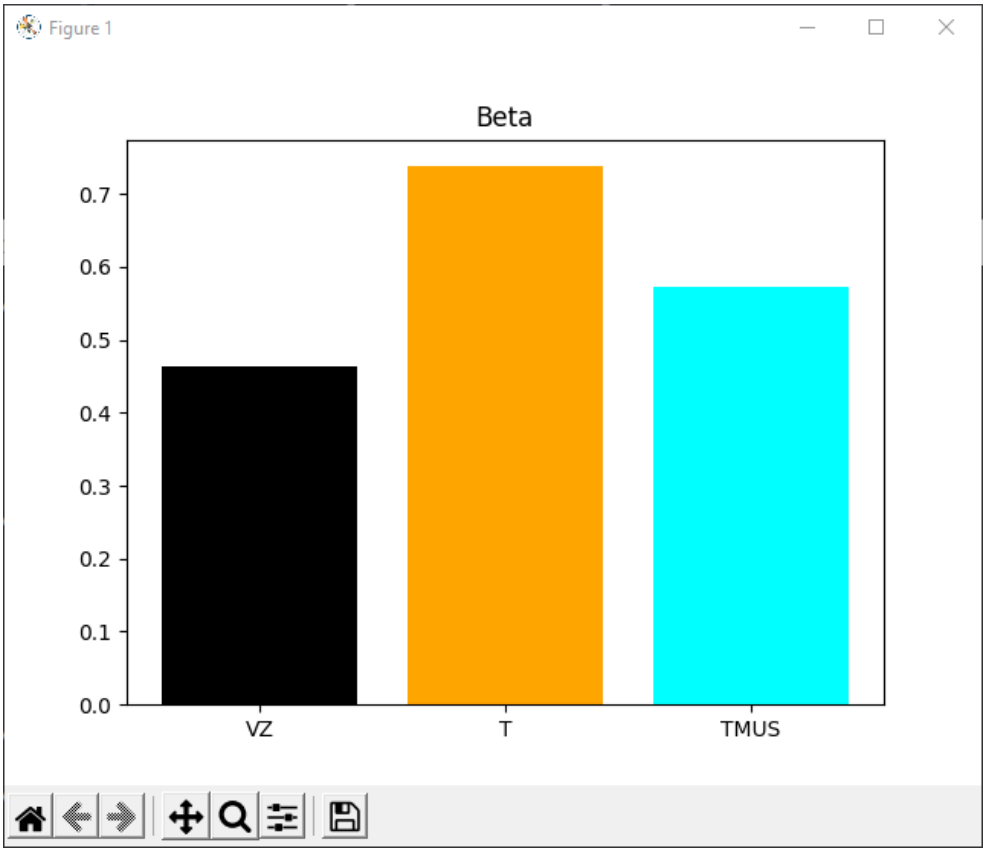


Figure4.5.2-4 Beta Output.

Market capitalization refers to the total value of all a company's shares of stock. It is calculated by multiplying the price of a stock by its total number of outstanding shares.

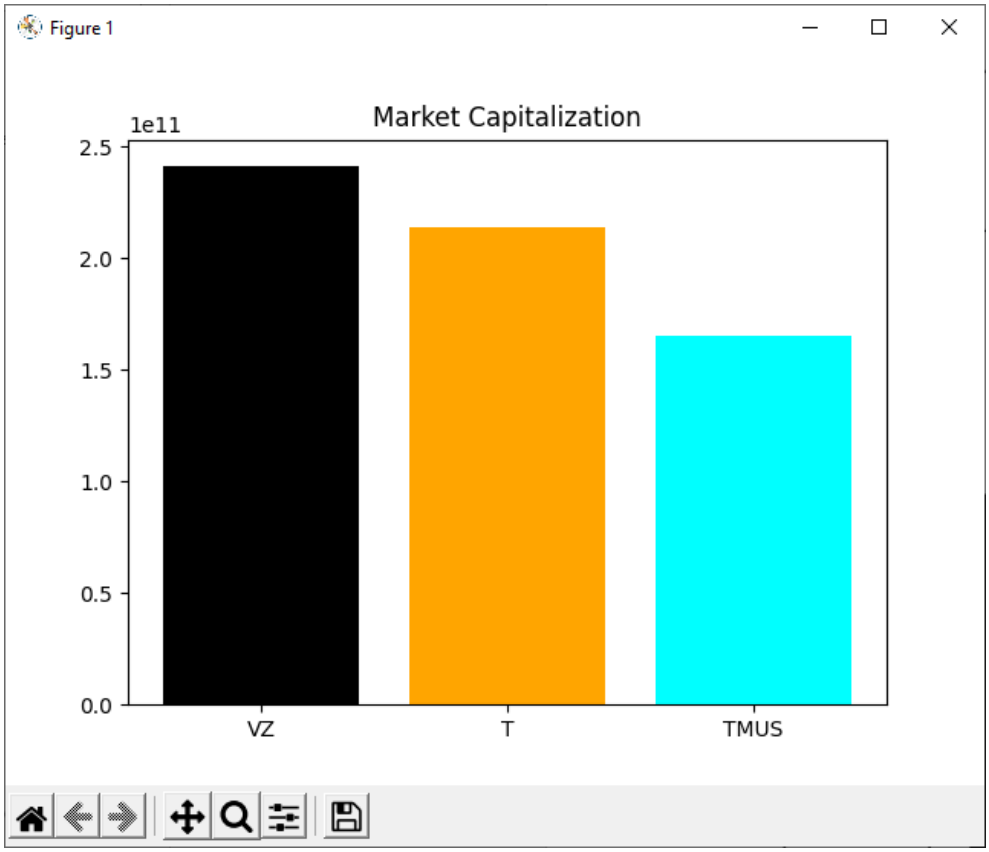


Figure5.4.2-5 Market Capitalization Output.

A stock dividend is a dividend payment to shareholders that is made in shares rather than as cash.

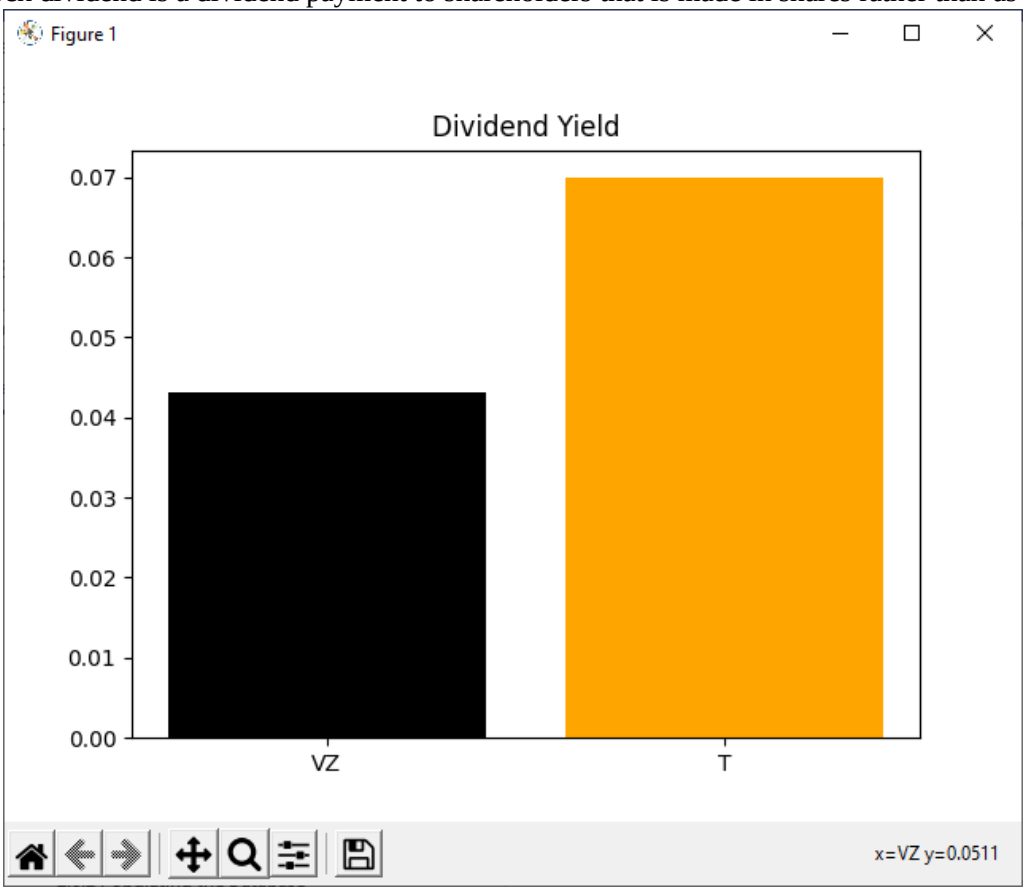


Figure5.4.2-6 Dividend Yield

5.6 Integrating Matplotlib & Django

Since I am not currently aware of how to embed a matplotlib graph into an html template as I did with Bokeh, I will take time to research the topic and chose the best solution.

After an extensive amount of research, unfortunately I did not find a common, conventional way to embed multiple graphs into a single HTML template. However, since Matplotlib graphs have the capability of being saved as a png file, I will save each plot locally and configure my Django project to handle static files such as images.

5.6.1 Static file configuration

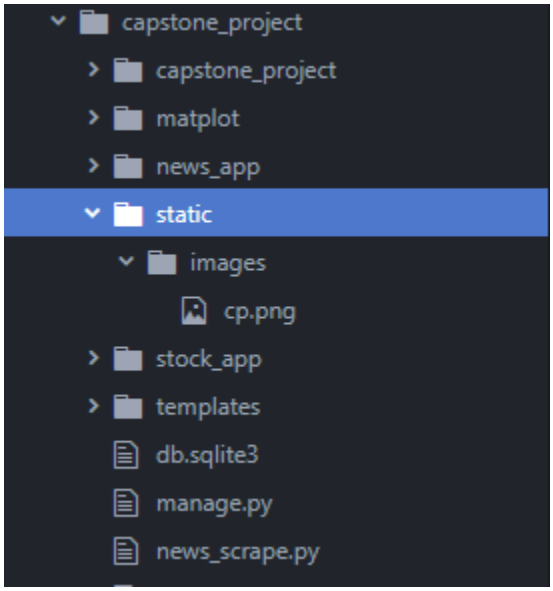
Like the process of configuring my Django project to point to the "template" directory for HTML files, I will also have to tell Django how to handle static files. This process begins by creating two new variables in the project "settings.py" file. This variable serves the same purpose as the "TEMPALATE_DIR" variable in that it tells Django where to look for static files in the main directory.

```
15  BASE_DIR = os.path.join(os.path.dirname(os.path.abspath(__file__)))
16  TEMPLATE_DIR = os.path.join(BASE_DIR, '../templates')
17  STATIC_DIR = os.path.join(BASE_DIR,'../static')

125
126  STATIC_URL = '/static/'
127
128  STATICFILES_DIRS = [
129      STATIC_DIR,
130  ]
```

Figure4.6.1-1 Configuring Django settings for static files.

Next, I will go create a new directory called “static” in the same place I created the template directory (inside the django_Project folder) then, create another folder inside the static folder to hold images (called “images”).



After creating the new folders, I placed the “cp.png” image inside and will now try displaying the image in the browser by specifying the path in the address bar to my image folder.

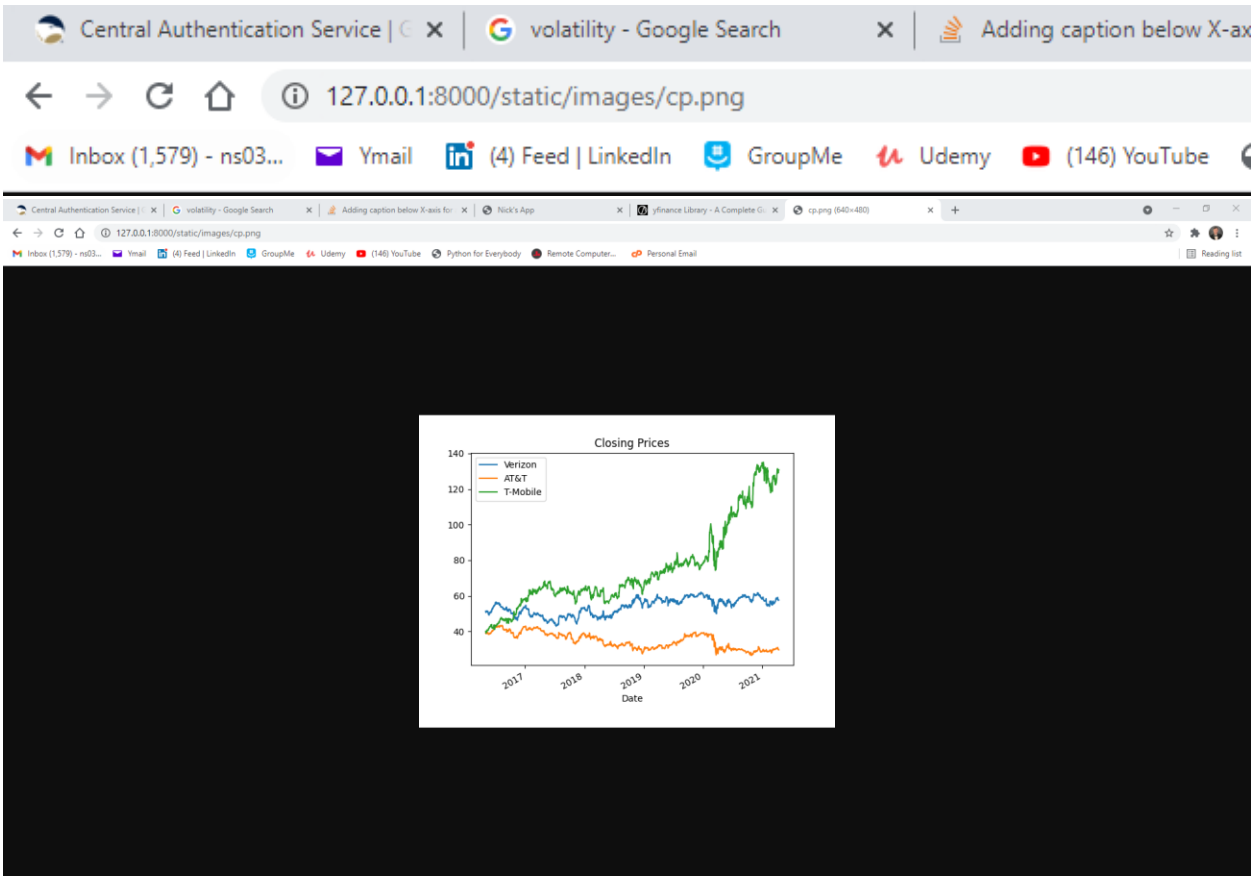


Figure4.6.1-2 Django successfully serving static image.

5.6.2 Creating the template

Now that my Django project knows how to handle static files, I will create a new template called "image_page.html" that will display all my Matplotlib graphs. To embed an image in a Django template, I will be using similar context processors as I have for all other templates except for I will use a special processor at the beginning of the HTML body to handle static files. The next picture encapsulates the HTML body containing the image tags for each chart previously created. I will also make sure each image has been added to my "static/images" directory.

```
1  <!DOCTYPE html>
2  {% load static %}
```

Figure4.6.2-1 Unique context processor for static files.

```
    <br>
    <img src="{% static "images/cp.png" %}" alt='error'>
    <img src="{% static "images/vol.png" %}" alt='error'>
    <img src="{% static "images/volatile.png" %}" alt='error'>
    <img src="{% static "images/new_mc_line.png" %}" alt='error'>
    <img src="{% static "images/mc_bar.png" %}" alt='error'>
    <img src="{% static "images/pe_ratio.png" %}" alt='error'>
    <img src="{% static "images/beta.png" %}" alt='error'>
    <img src="{% static "images/divds.png" %}" alt='error'>
    <br>
</body>
```

Figure4.6.2-2 Creating image element for each graph.

The last part of the process entangles a new function in the “matplot” application’s “views.py” file and configuring it in the app’s “urls.py” file as well. All we will need inside the function is a return statement that renders the new template and its appropriate context processors.

```
99
100 def mplot(request):
101     return render(request, 'image_page.html')
```

Figure4.6.2-3 Creating new function in the views to render new template.

```
1 from django.conf.urls import url
2 from matplot import views
3
4 urlpatterns = [
5     url(r'graph_page/', views.candlestick, name='candlestick'),
6     url(r'image_page/', views.mplot, name='mplot'),
```

Figure4.6.2-4 Configuring application URLs to find my new function.

5.7 Final output

Figure4.7-1 "image_page.html" final Output.

The last steps of this template only include front-end designing and adding paragraphs of text for contextual purposes.

6 Project Deployment

After taking the time to research a few well-known hosting services such as Heroku, azure, etc. I finally came across a free service called “Python Anywhere” (link). I will explain the service in more detail, why I chose it, and the process I went through to deploy my Django project. However, Python anywhere is not the only service I will need. Before I can use it, I will need to move my project to an online repository where it will then be maintained by a version control system. To do this, I will be using Github.

6.1 Setting up Github

The first step in setting up Github requires installing “Git”, which is version control system that helps keep track of changes in code. Github is an independent company/website that helps manage this control system, host files on their server, and excellent collaboration features. To install Git, I went through the installation process on their official website (link). Next, I will log into my free Github account and create a new repository to host my project.

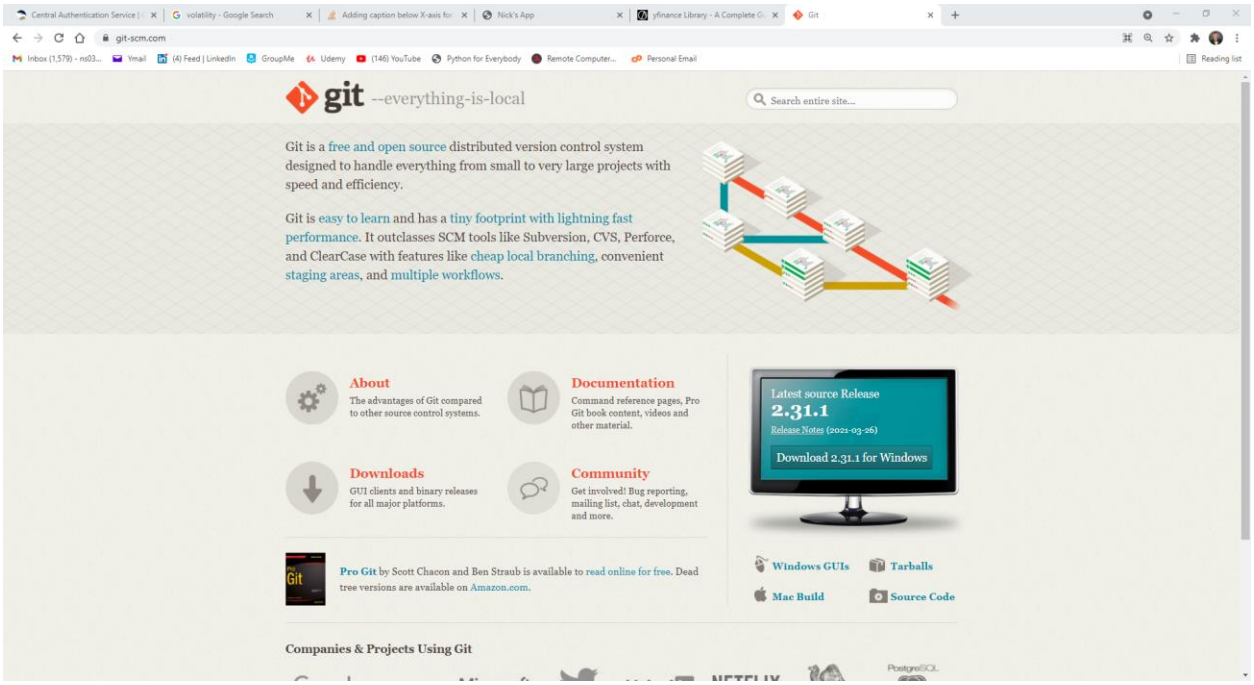


Figure5.1-1 Downloading Git.

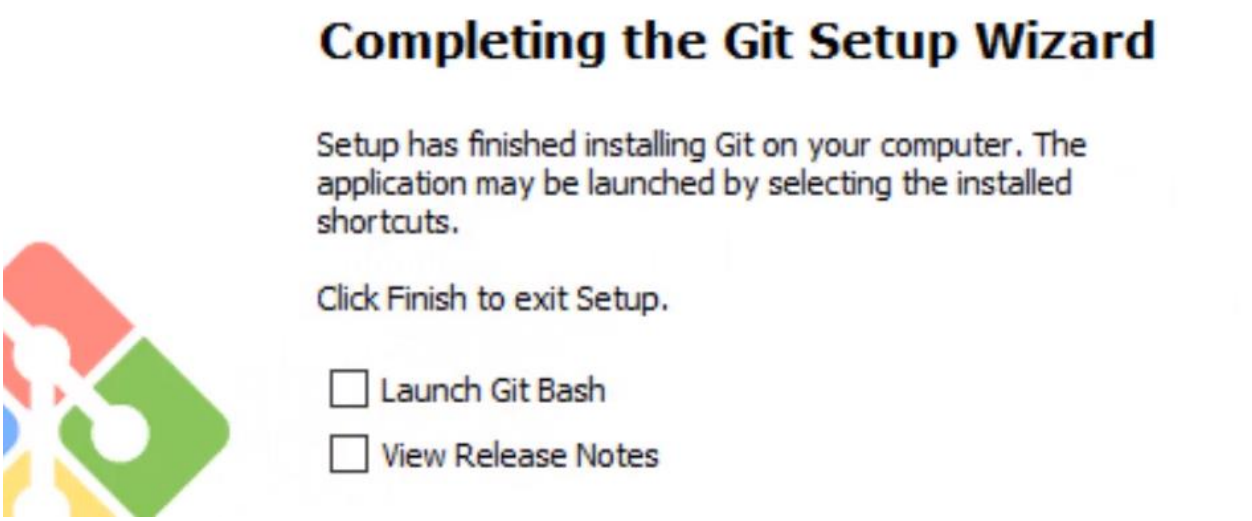


Figure5-1.1 Installing Git

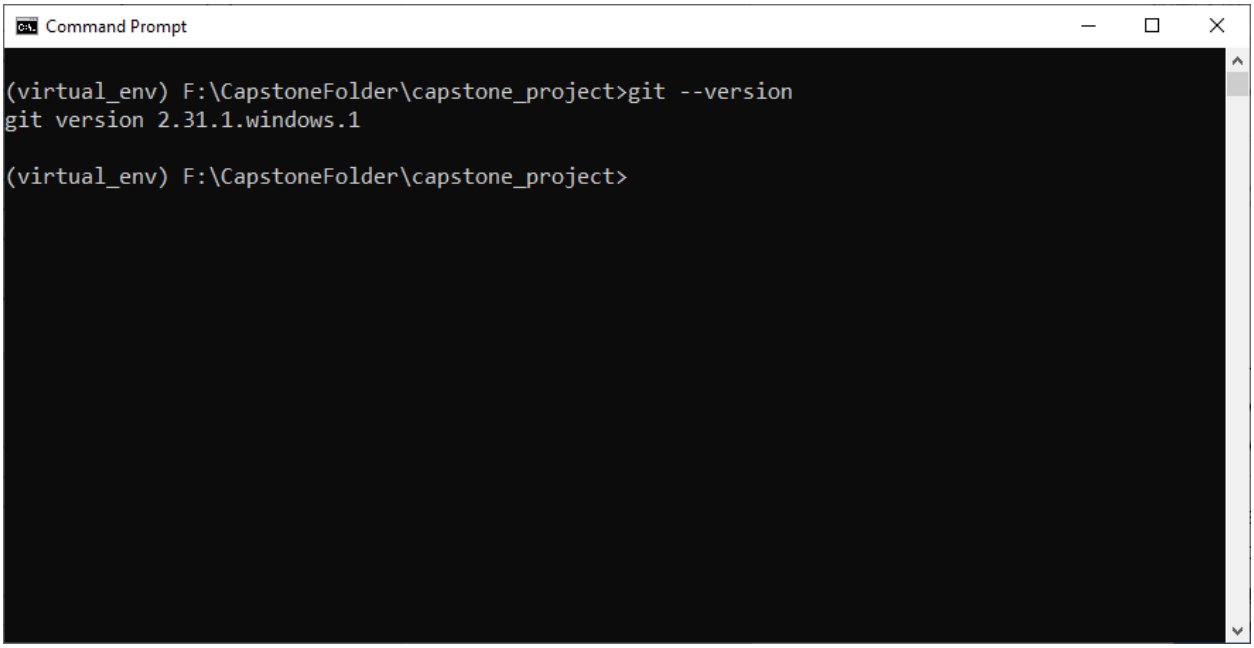


Figure5.1-3 Git successfully installed

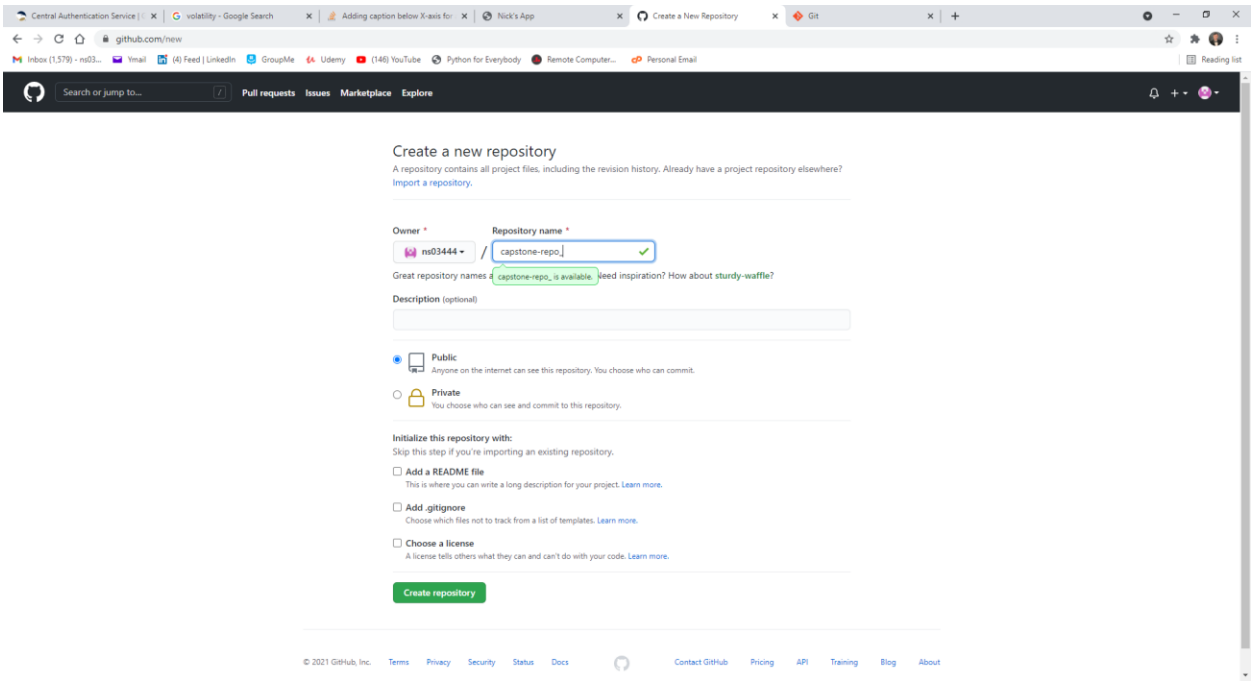


Figure5.1-4 Creating new Github repository.

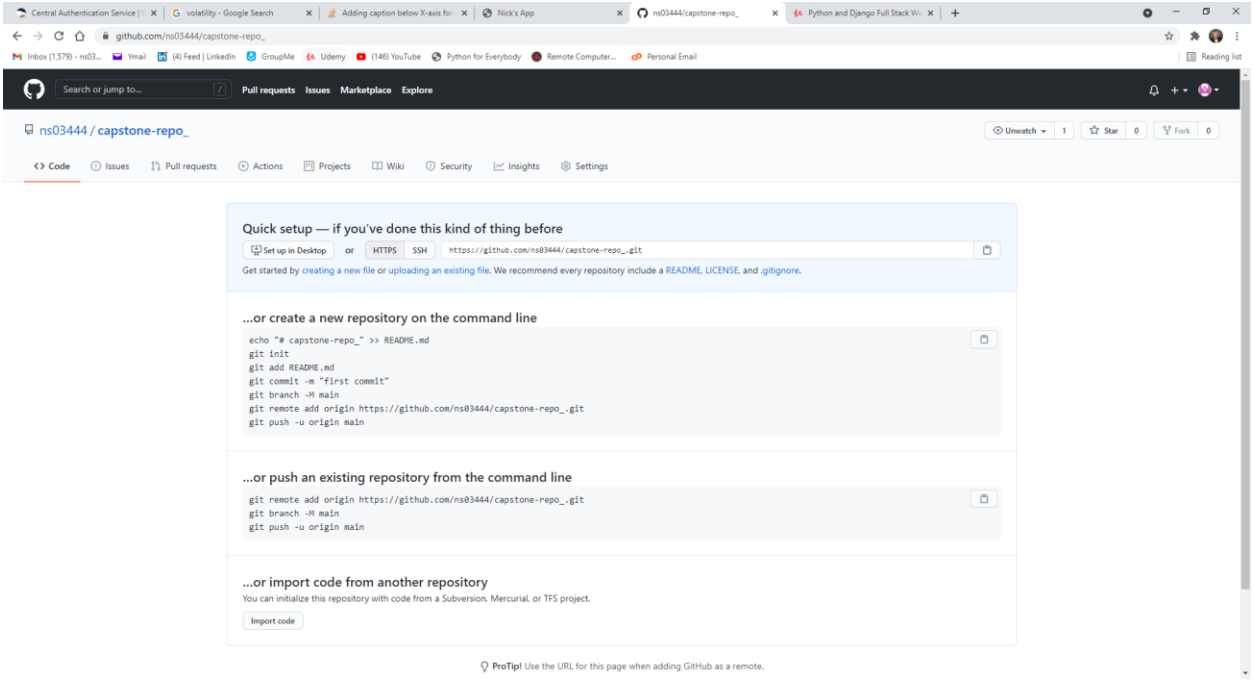


Figure5.1-5 Successfully created Github repository.

After creating my "capstone-repo", Github prompted a new screen that displays a list of commands for uploading, pushing, etc. files to a repository via the command prompt. However, before I can run any of the commands, I will create a new folder, "my_base" inside of the top-level Django project folder to contain the actual git repository. Then, copy the actual Django project and paste it inside this new folder.

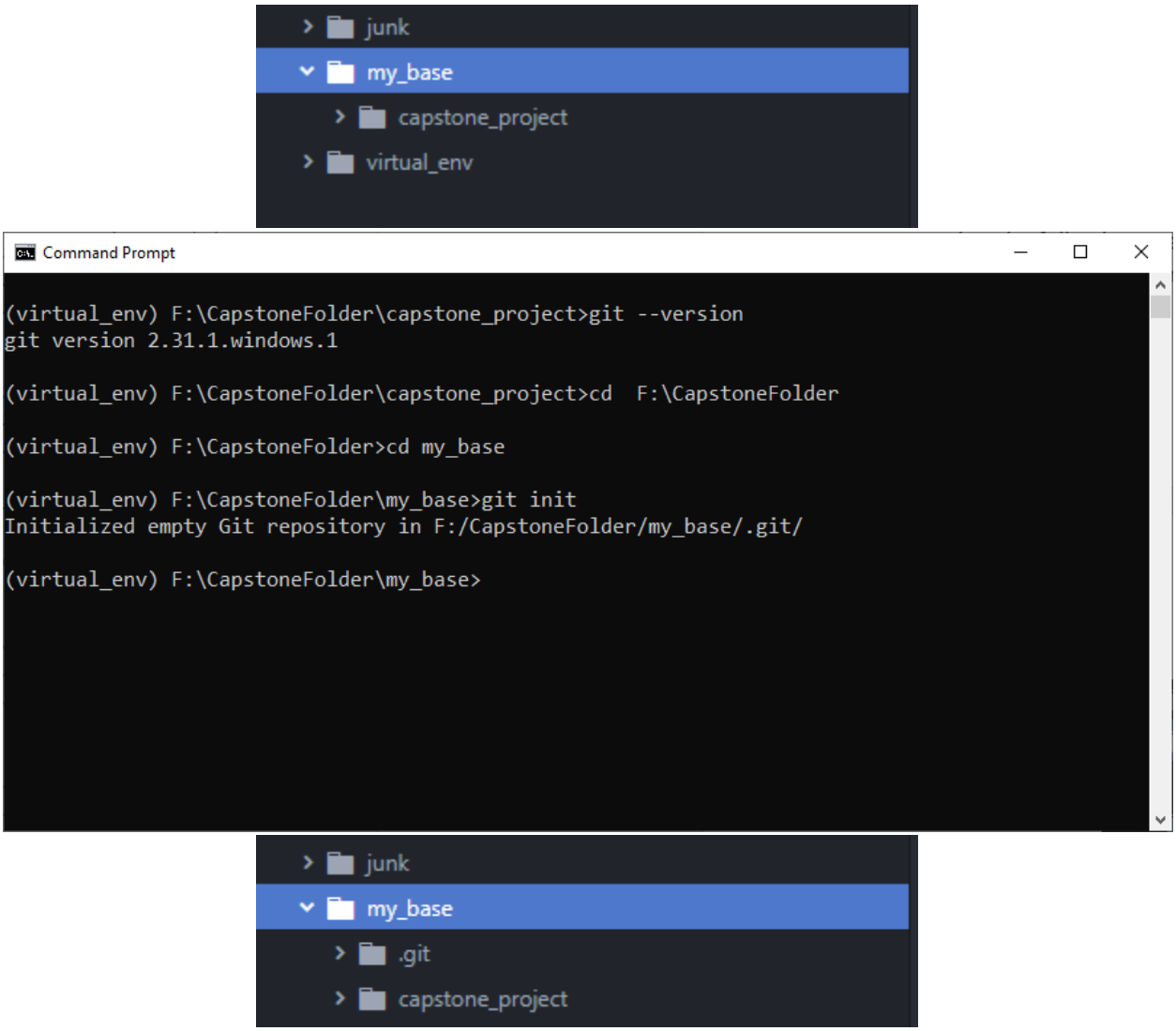


Figure5.1-6 Git repository initialized.

I will run the next command followed by "." to add *all* of the files in the capstone_project folder to the git repositroy.

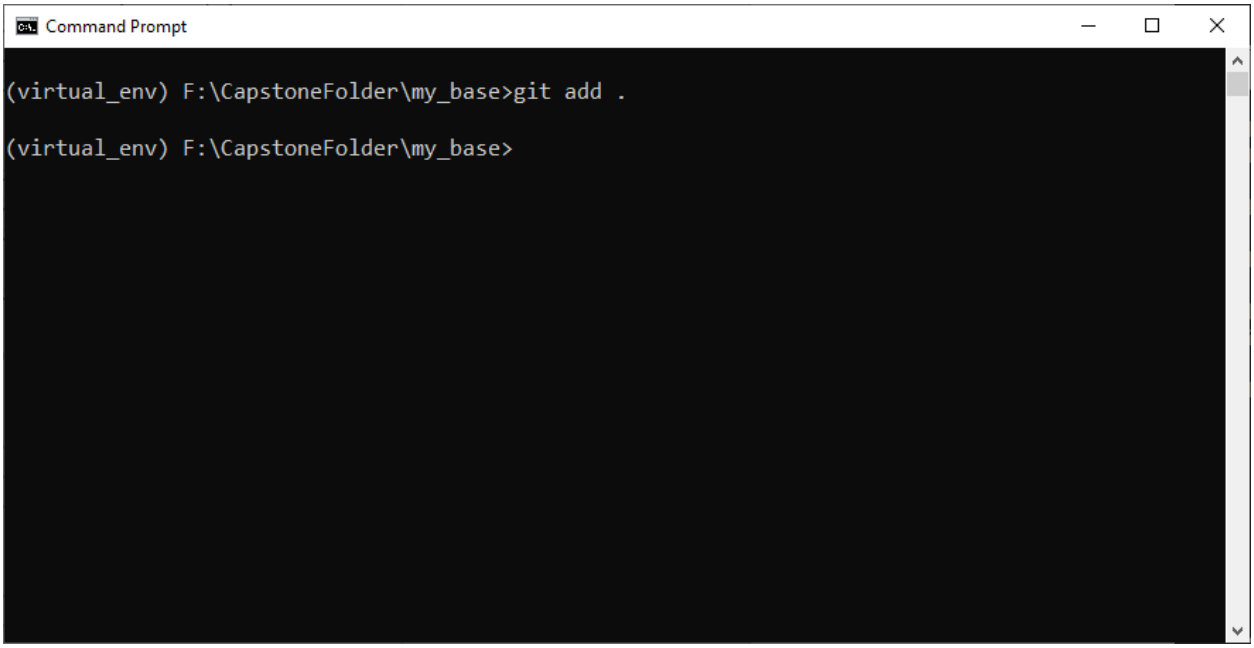


Figure5.1-7 Adding all project files to repository.

Now that everything has been added, it will need to be *committed* to Github.

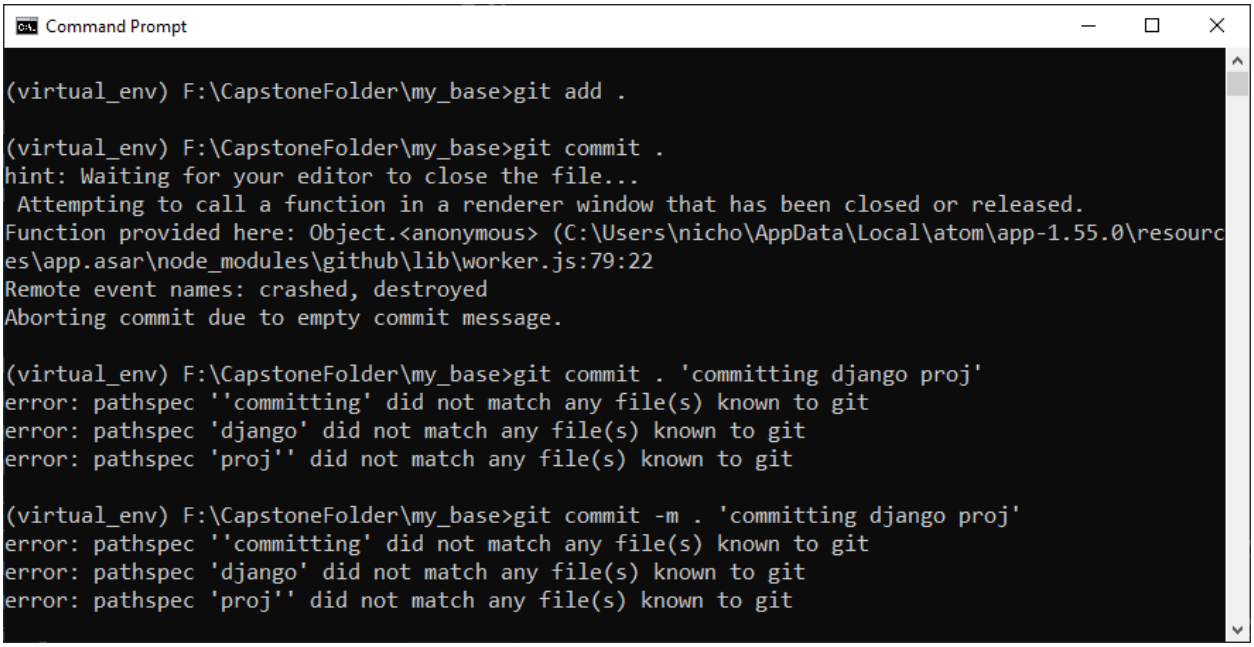


Figure5.1-8 ERROR - Typo in command

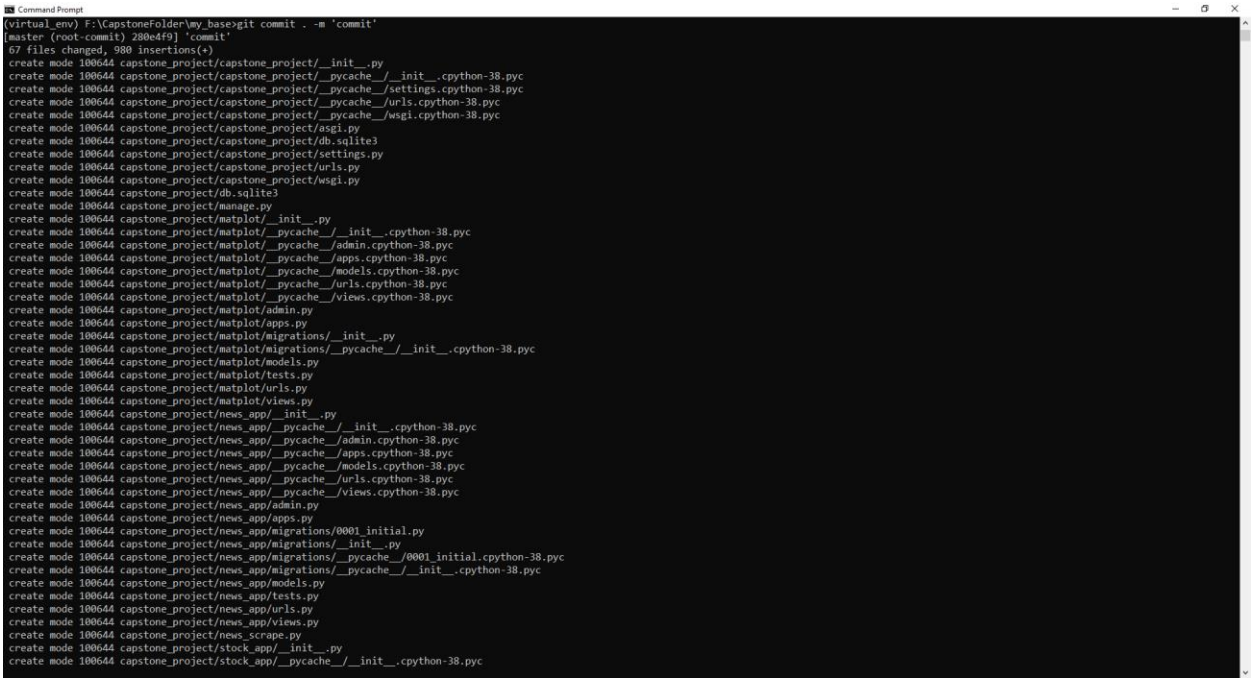


Figure5.1-9 Typo fixed; files successfully committed.

Now I will be pushing the *git* repository to my actual *Github* repository (stored on their website) by including the URL to the repository in the next command. Then, use the "push" command to make the physical changes on Github.

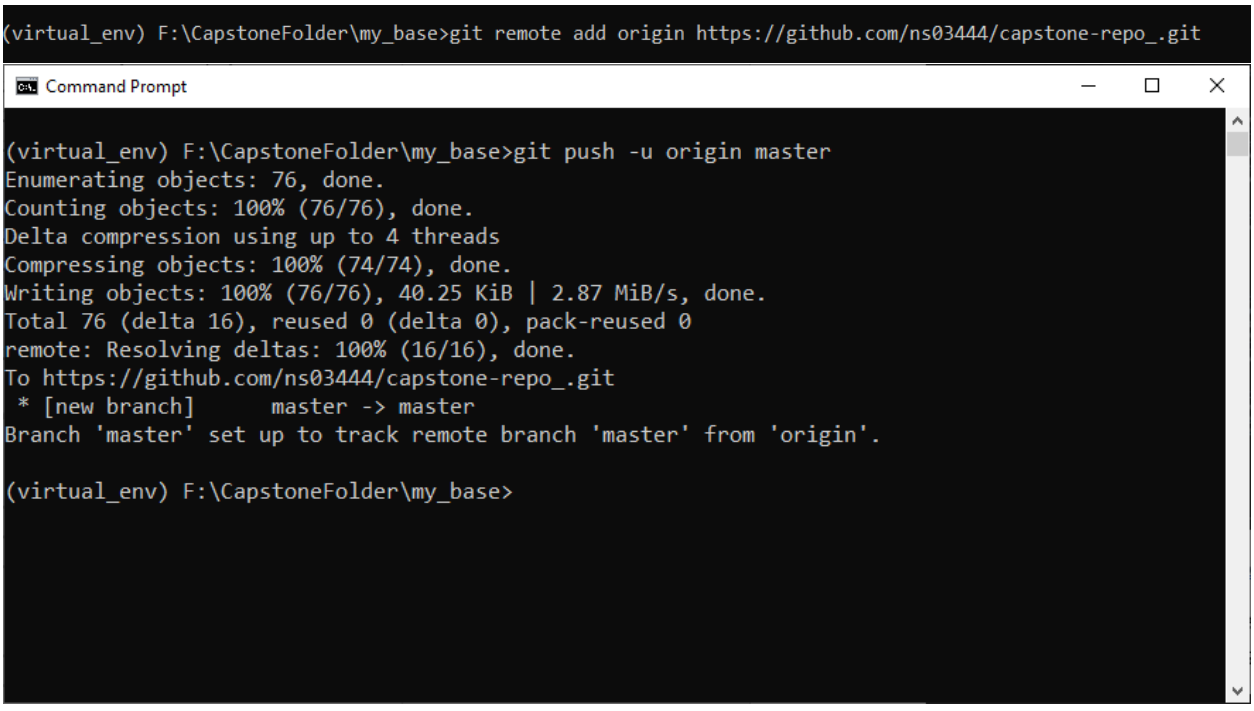


Figure5.1-10 Files completely pushed to Github Repository.

Now, when I go back to Github's website to view my repository, my Django project and all of the files included can be viewed on their website.

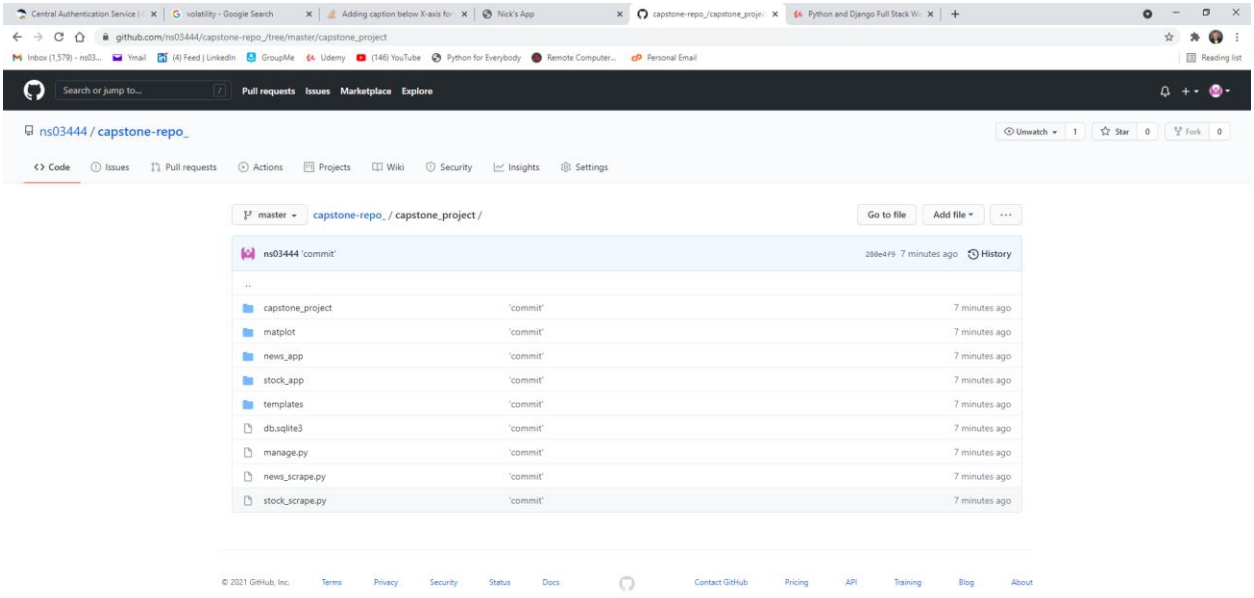


Figure5.1-11 Django project moved to Github Repository successfully.

6.2 Github to Python Anywhere

I will begin by creating a free account with Python Anywhere, which is an *online, integrated development environment and web hosting service*. Then, I will open a new bash console which functions like the command prompt in that it allows interaction with bash files, the Python interpreter, etc. and, then create a new virtual environment (similar to "venv", used locally) to host my project. After the environment has been created, I will install Django and all other dependencies that were installed to the local virtual environment.

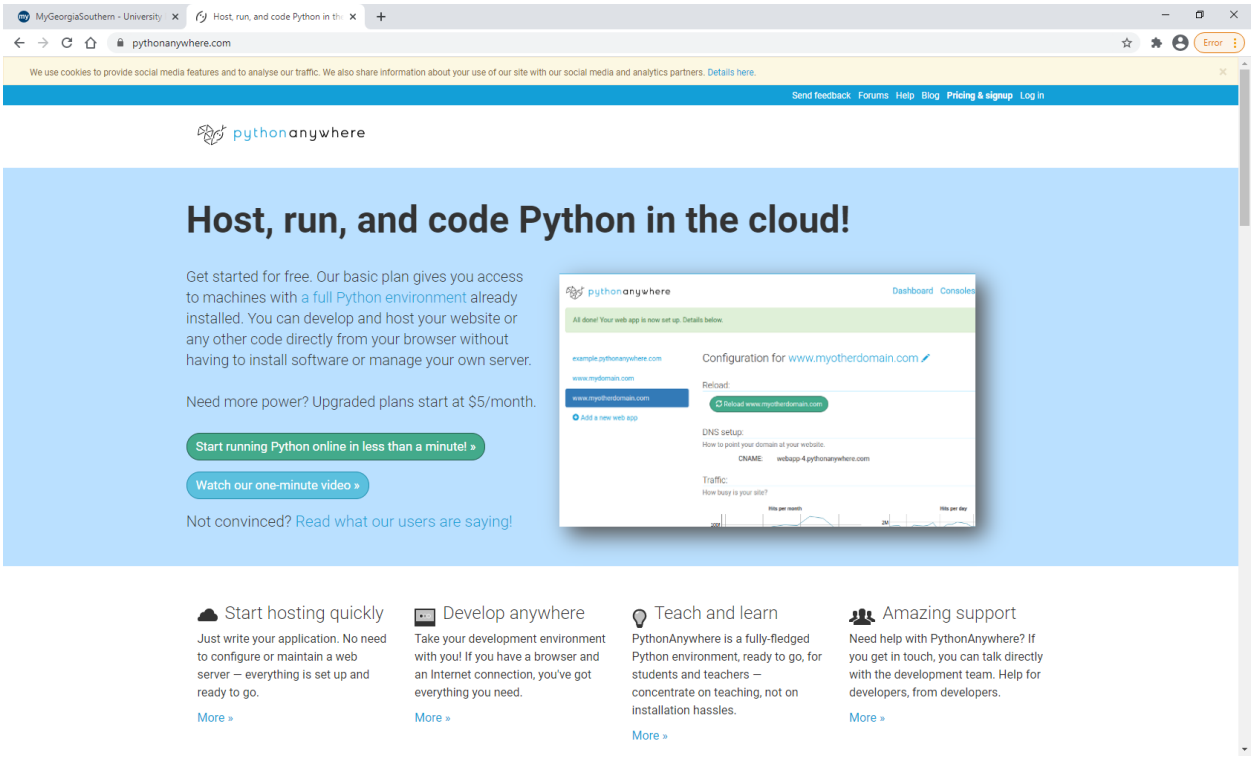


Figure5.2-1 Python anywhere homepage.

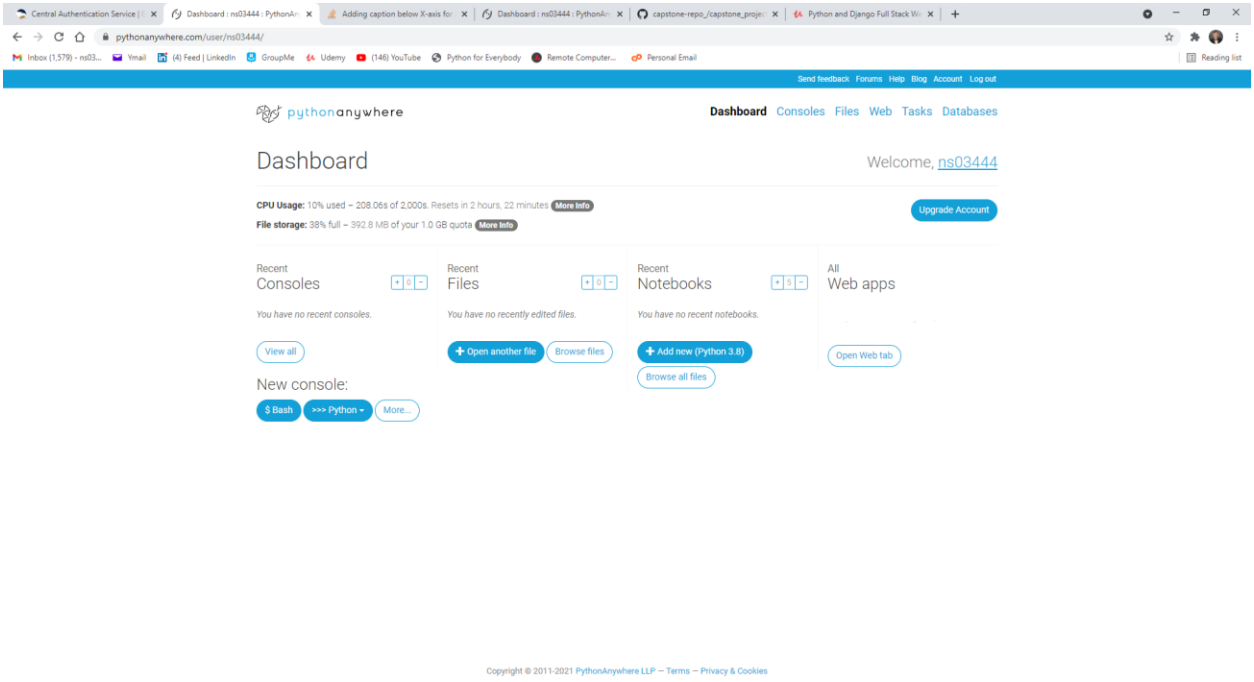


Figure5.2-2 Account dashboard.

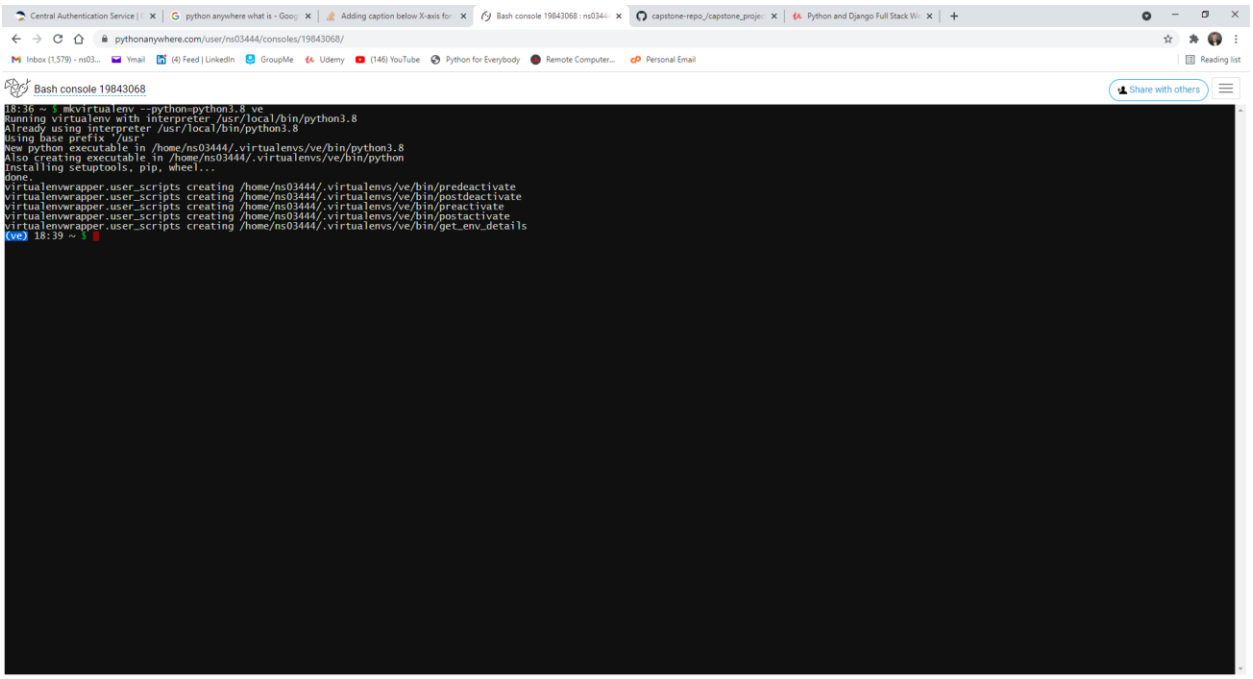


Figure5.2-3 Creating a new virtual environment.

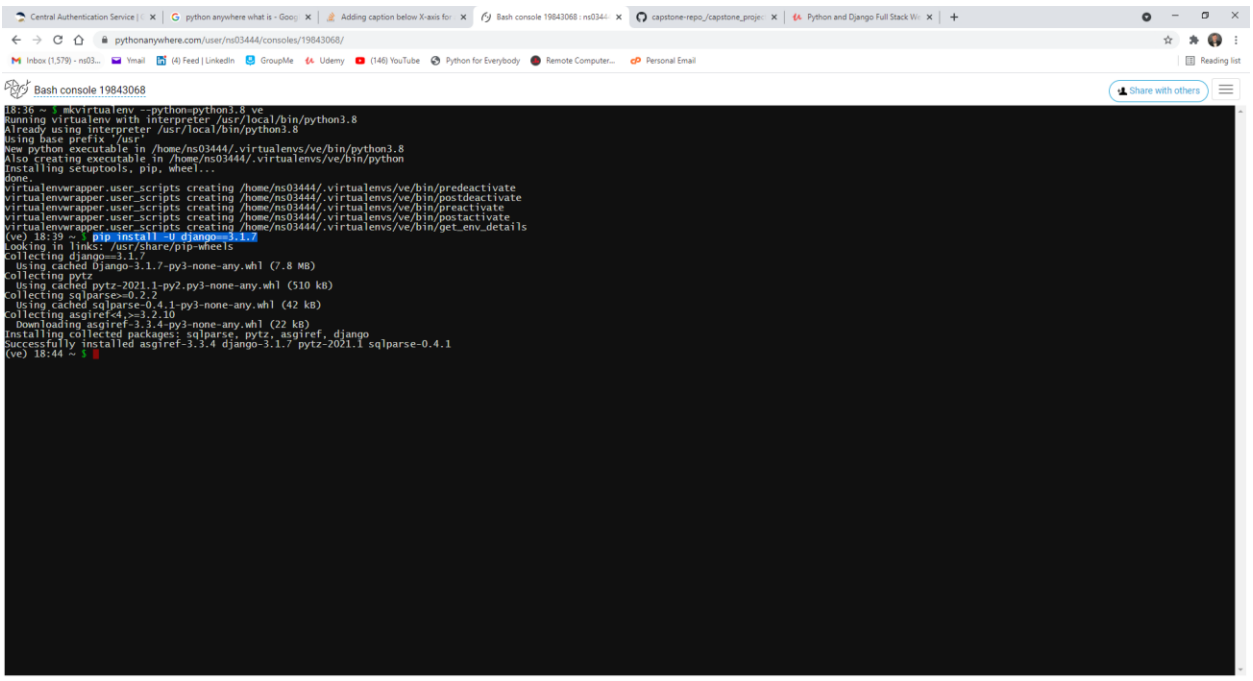


Figure5.2-4 Installing Django & other dependencies.

6.2.1 Connecting to the Repository

Now that Django and all other dependencies have been installed, it is time to get an actual copy of our git repository onto the hosting service. To do this, all I will need is the "clone" command and the address to my repository on my Github account.

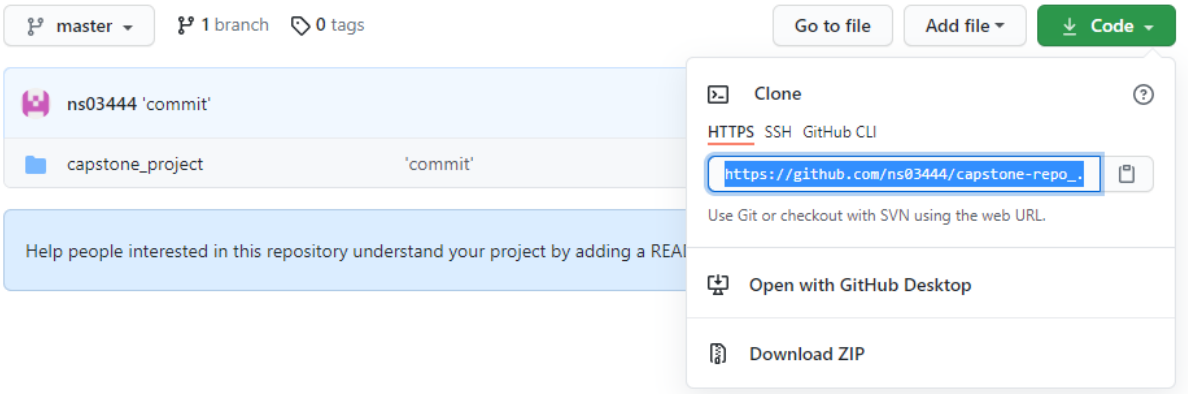


Figure5.2.1-1 Copying address to Github repository.

Bash console 19843068

```
(ve) 18:52 ~ $ git clone https://github.com/ns03444/capstone-repo_.git
Cloning into 'capstone-repo_'...
remote: Enumerating objects: 76, done.
remote: Counting objects: 100% (76/76), done.
remote: Compressing objects: 100% (58/58), done.
remote: Total 76 (delta 16), reused 76 (delta 16), pack-reused 0
Unpacking objects: 100% (76/76), done.
Checking connectivity... done.
(ve) 18:53 ~ $
```

Figure5.2.1-2 Cloning Github repository.

Bash console 19843068

```
(ve) 19:03 ~/capstone-repo_ (master)$ ls
capstone_project
(ve) 19:03 ~/capstone-repo_ (master)$ cd capstone_project
(ve) 19:03 ~/capstone-repo_/capstone_project (master)$ ls
capstone_project  db.sqlite3  manage.py  matplot  news_app  news_scrape.py  stock_app  stock_scrape.py  templates
(ve) 19:03 ~/capstone-repo_/capstone_project (master)$
```

Figure5.2.1-3 Repository successfully cloned (project files listed).

6.2.2 Creating the web app

Now that my repository has been cloned to my virtual environment on Python Anywhere, I will return to the dashboard and use the web tab to create a *new web application.*

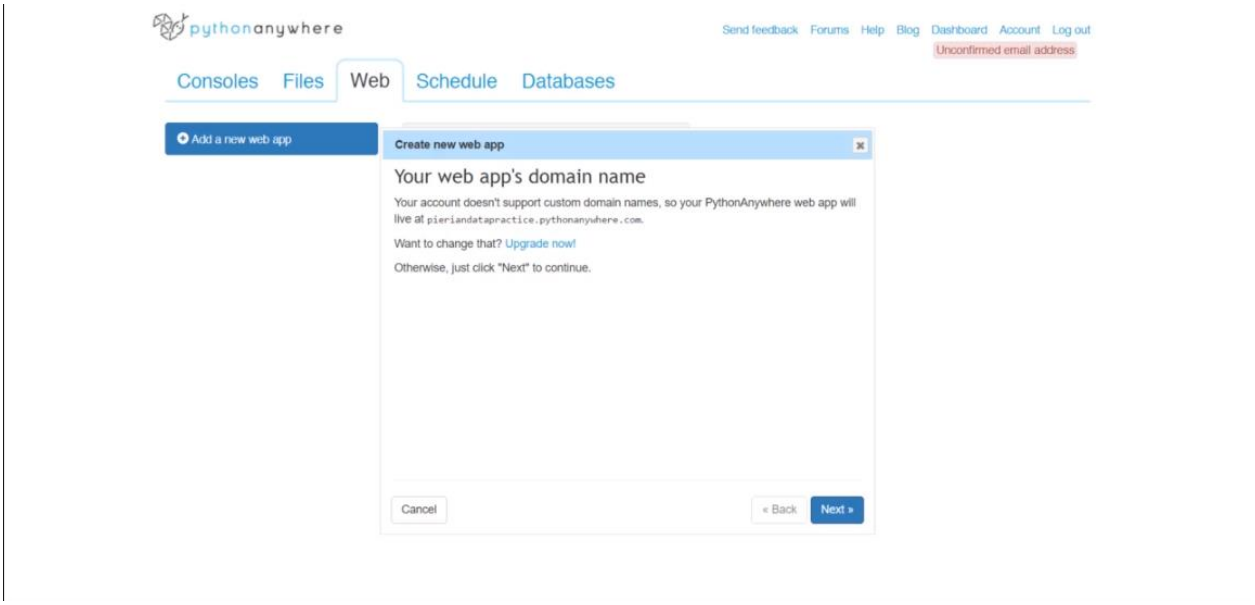


Figure5.2.2-1 Python Anywhere Web tab.

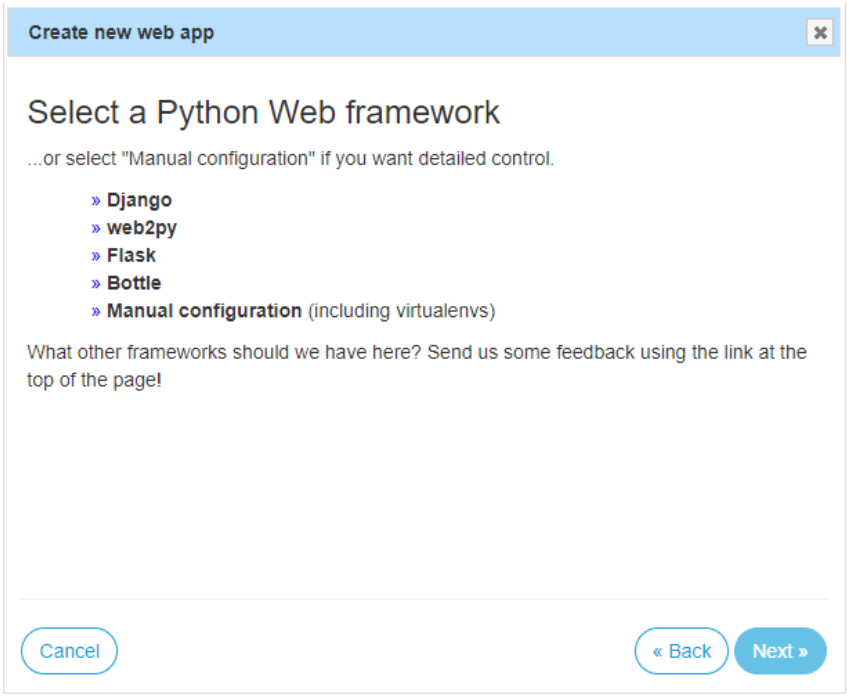


Figure5.2.2-2 Creating a new web app with Python Anywhere.

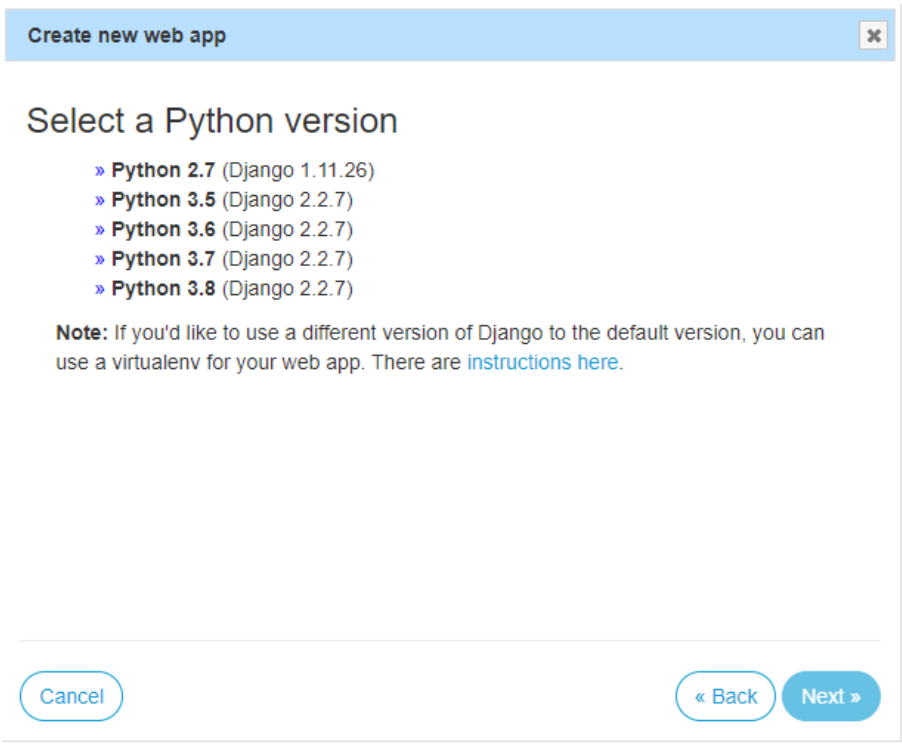


Figure5.2.2-3 Creating a new web app with Python Anywhere – cont'd.

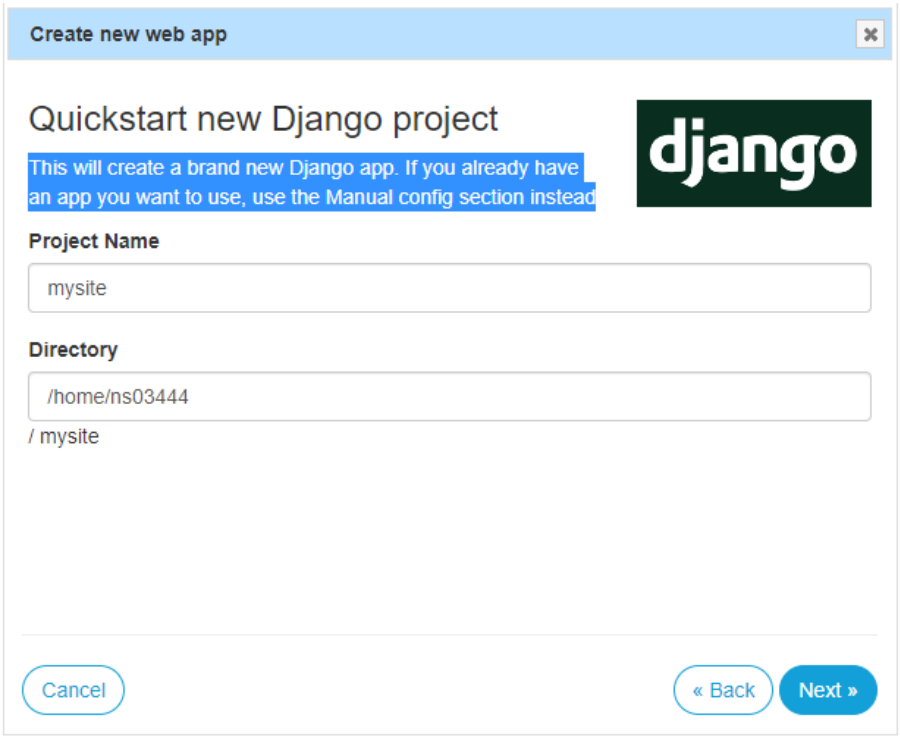


Figure5.2.2-4 Creating a new web app with Python Anywhere – cont'd.

Since we already have an existing Django project, I will be making the application configurations manually.

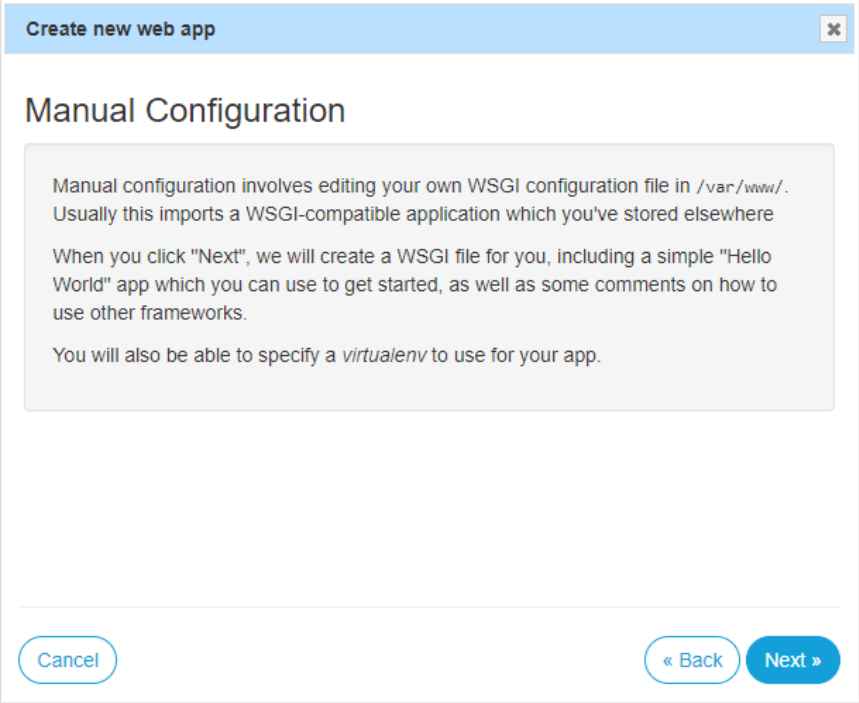


Figure5.2.2-5 Using manual configuration for existing Django projects.

The image above explains the process for manually configuring the file. By default, all free accounts are assigned *one free domain name that starts with the account's username, followed by ".pythonanywhere.com".* My new domain "ns03444.pythonanywhere.com" was created after clicking next in the dialogue box shown above.

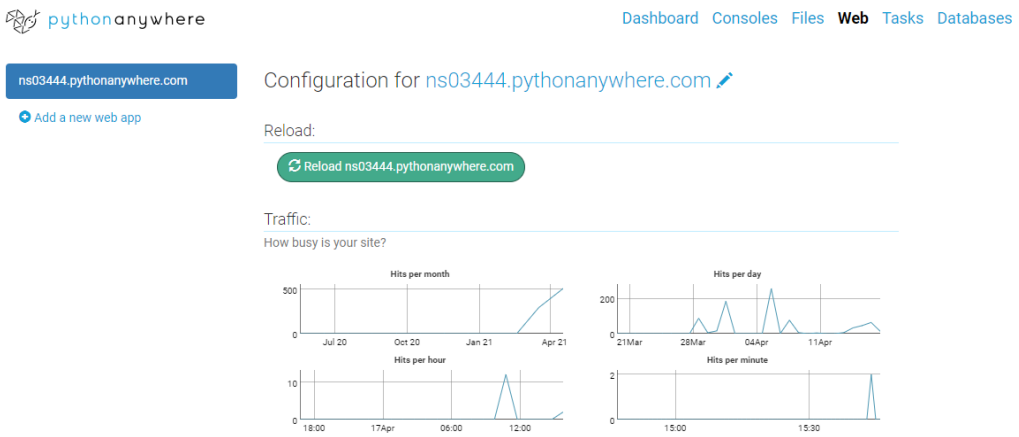


Figure5.2.2-6 Python Anywhere web app created.

The image above displays that my new web app has been created and now I will continue following the manual configuration instructions by following the hyperlink to my domain.

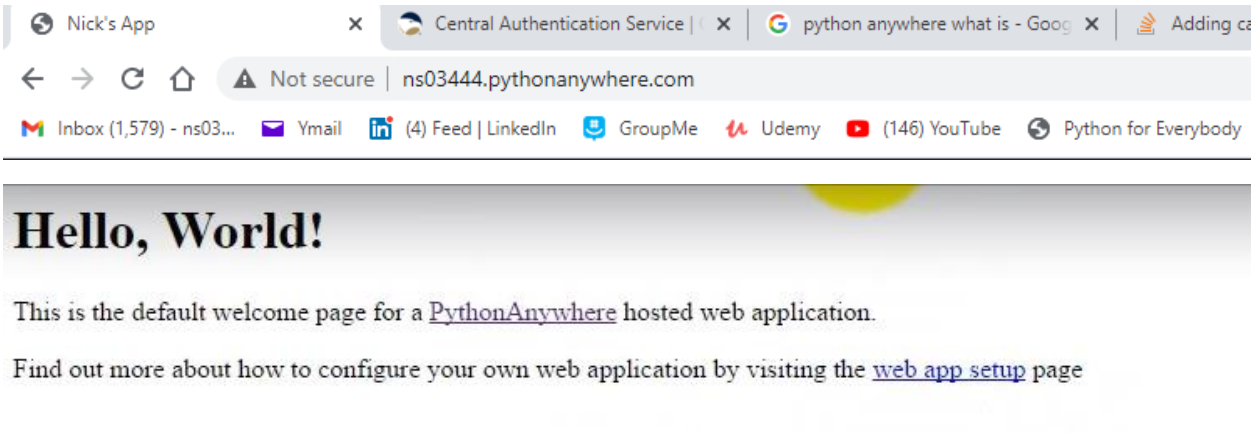


Figure5.2.2-7 Webpage returned when following link to my domain.

By following the "web app setup" page hyperlink displayed above, I am returned back to the web tab in Python Anywhere. Here, I will specify directory routes and configure the "wsgi.py" file for deployment. First, I will define the path to my virtual environment.

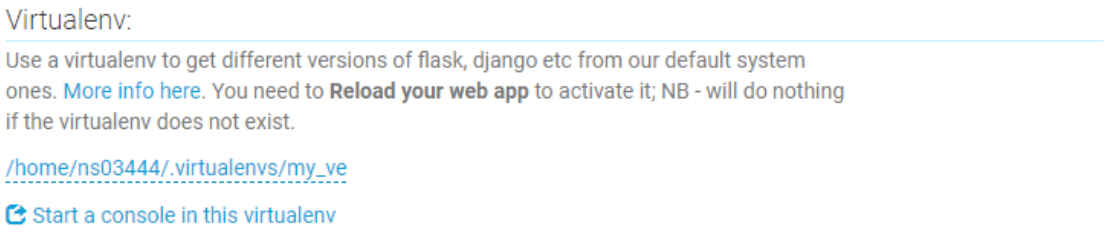


Figure5.2.2-8 Specifying which virtual environment to use for web app.

Next, I will configure the file path to my *source code*, which is simply the name of my Django project (capstone_project).

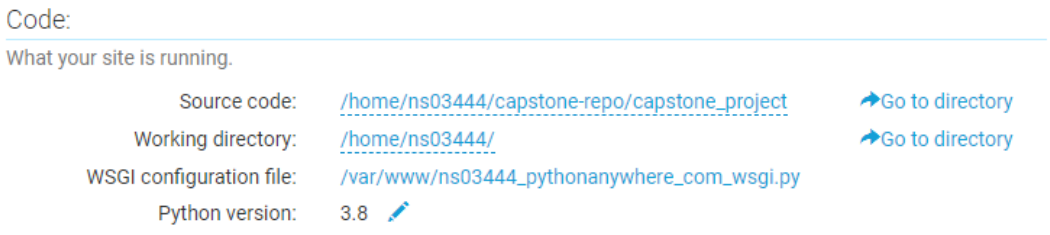


Figure5.2.2-9 Identifying source code for web app.

Next, I will configure the file path to my static files, this way the images embedded in the "image_page.html" will be displayed.

Static files:

Files that aren't dynamically generated by your code, like CSS, JavaScript or uploaded files, can be served much faster straight off the disk if you specify them here. You need to **Reload your web app** to activate any changes you make to the mappings below.

URL	Directory	Delete
/static/admin	/home/ns03444/.virtualenvs/my_ve/lib/python3.8/site-packages/django/contrib/admin/static/admin	
/static/	/home/ns03444/capstone-repo/capstone_project/static	
Enter URL	*Enter path*	

Figure5.2.2-10 Mapping to static files.

Last, I will be opening the web app's "wsgi.py" file to make some edits/additions. Luckily, Python Anywhere provides a text editor as well as a python shell to run code, right from the browser.

WSGI configuration file: /var/www/ns03444_pythonanywhere_com_wsgi.py

```
# To use your own django app use code like this:
import os
import sys
#
## assuming your django settings file is at '/home/ns03444/mysite/mysite/settings.py'
## and your manage.py is is at '/home/ns03444/mysite/manage.py'
path = '/home/ns03444/capstone-repo/capstone_project'
if path not in sys.path:
    sys.path.append(path)
os.chdir(path)
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "capstone_project.settings")
import django
django.setup()
#os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'
#
## then:
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()

```

Figure5.2.2-11 Configuring "wsgi.py" file for Django deployment.

The code shown above represents the changes made to the "wsgi.py" file. Like the code used in some local files, this will allow Python Anywhere to serve the Django application when the server is requested, as well as, allow hosts to the website. The last thing that must be done is adding my new domain name to "ALLOWED HOSTS" in the project settings file.

```
DEBUG =False

ALLOWED_HOSTS = ['ns03444.pythonanywhere.com']

```

Figure5.2.2-12 Configuring project settings for new host name.

After making the additions in the project settings; setting debug mode to false since I am no longer in the development phase of my application; and reloading the web application, my website is now live!

6.3 Final output

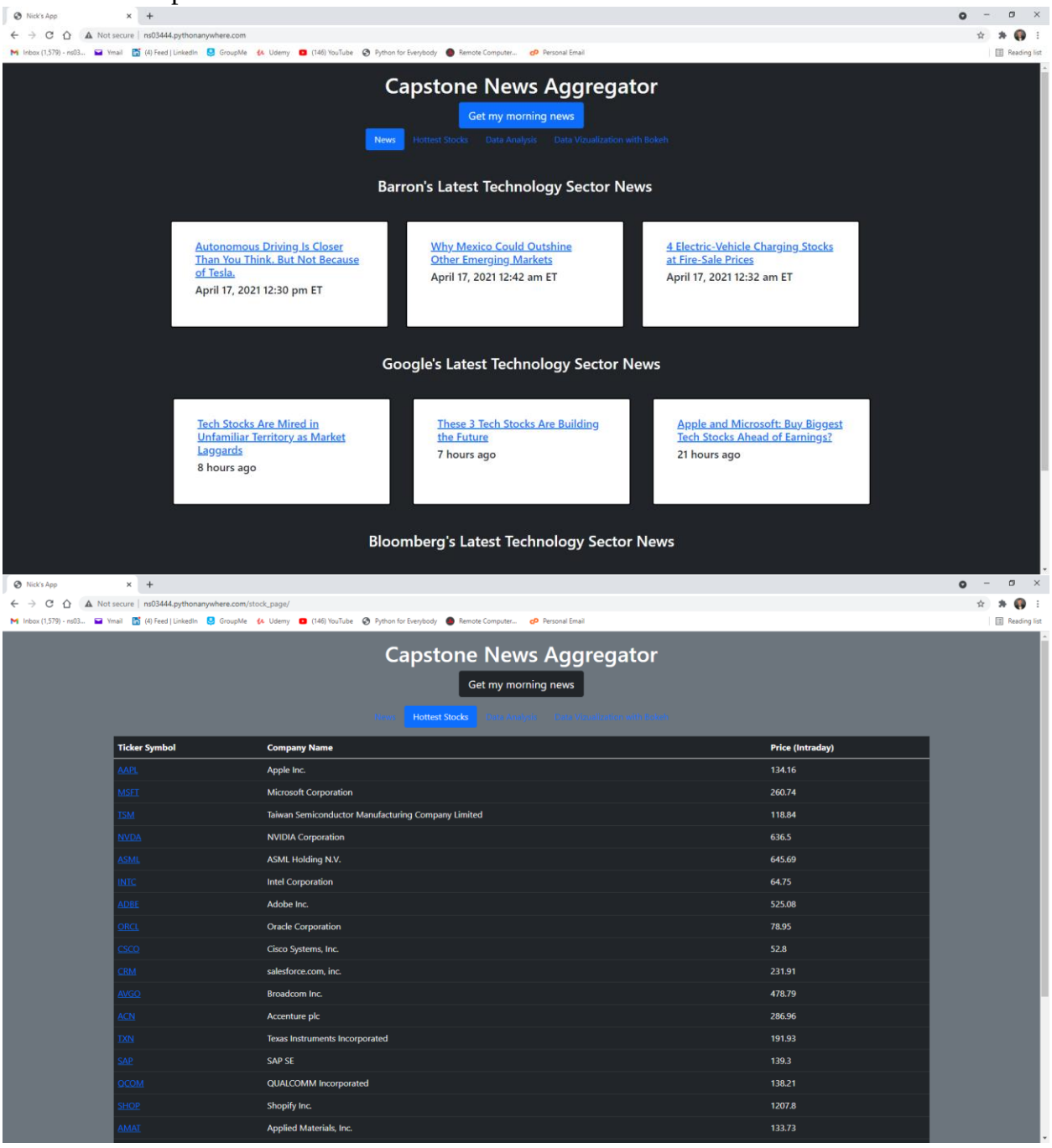
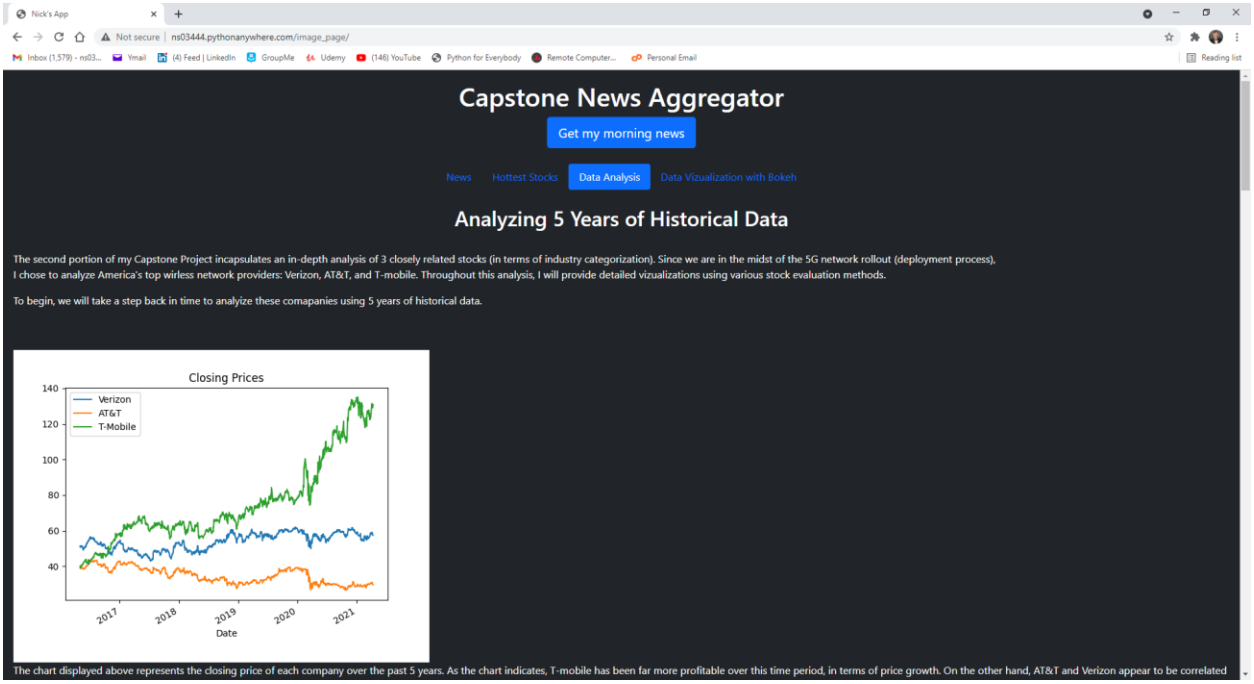


Figure5.3-1 Final Project Output.



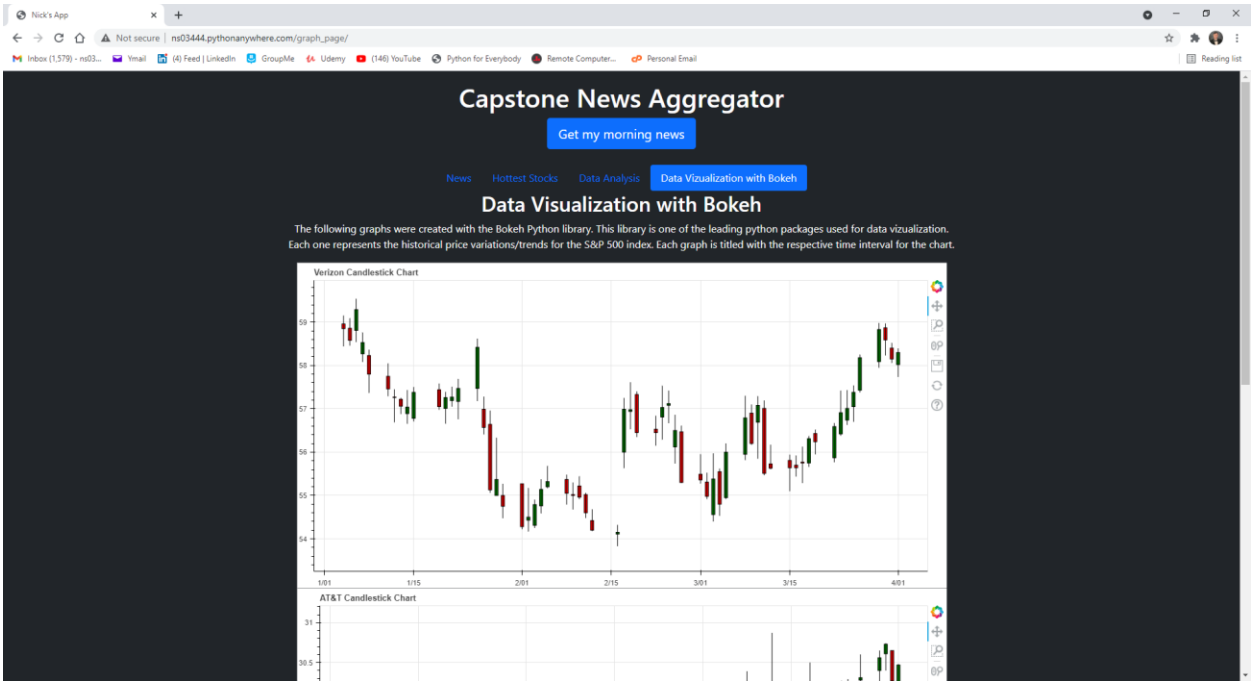


Figure5.3-1 Final Project Output.

6.4 Scheduling Tasks

One of the final, yet most important tasks I have yet to accomplish is, creating a timer/scheduler to dynamically updated the content on the news and stock table pages. Fortunately, the hosting service I chose to use, “Python Anywhere” has task scheduling capabilities. However, at this point, the “news_scrape.py” and “stock_scrape.py” scripts are only being used to populate the database tables when they are running. Therefore, I will need to add a new function in both files to delete the old objects before repopulating them. Then, I will schedule both python scripts to run as often as possible using Python Anywhere.

6.4.1 Modifying the tasks

I began the process by logging into my python anywhere account and navigating the file tab. There, I can open and edit the news/stock scraping files. I will add a function to the top of both files that deletes all objects in the tables, then use the same functions I previously used to repopulate the database.

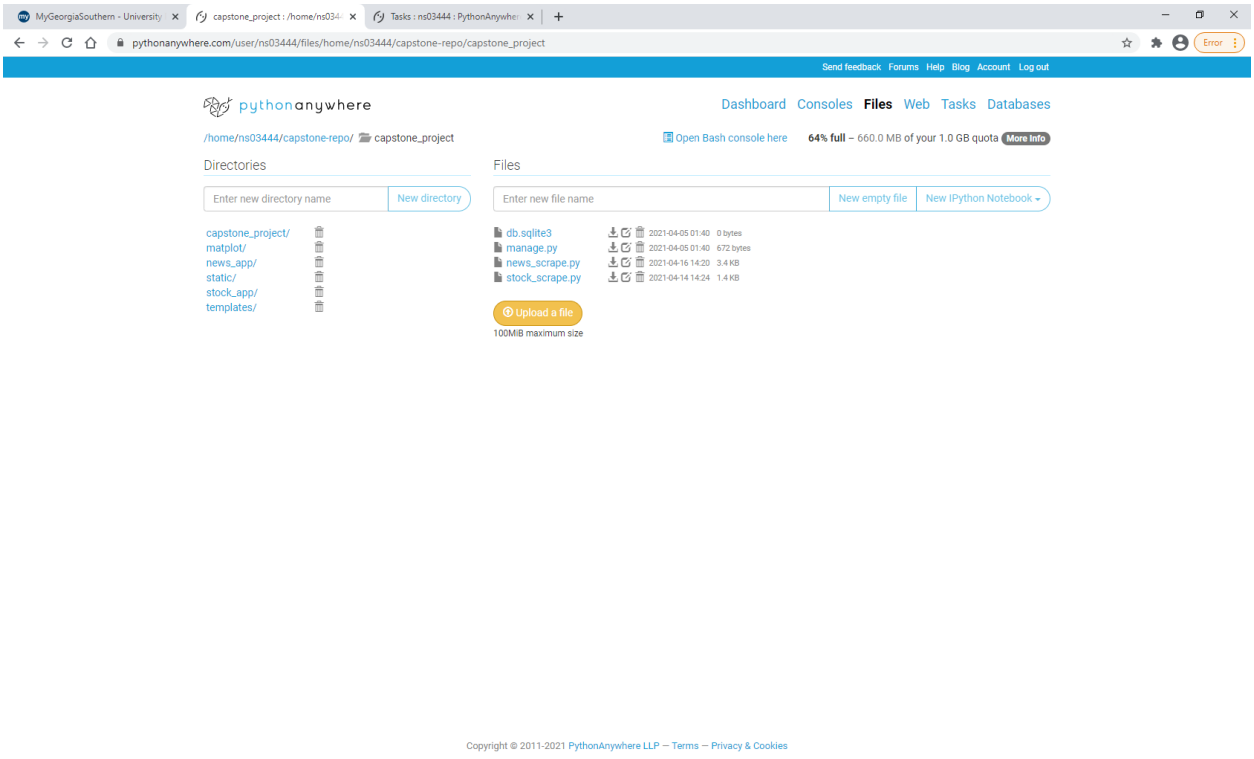


Figure5.4.1-1 Python Anywhere file tab – My project directory.

```
9
10 def news_delete():
11     deleter1 = Google.objects.all().delete()
12     deleter2 = Bloomberg.objects.all().delete()
13     deleter3 = Barron.objects.all().delete()
14     print('deleted.')
15 news_delete()
16
17 def news_scrape():
```

Figure5.4.1-2 Creating function to delete all news article items from each table.

```
12
13 def stock_delete():
14     deleter = Stock.objects.all().delete()
15     print('deleted.')
16 stock_delete()
17
18 def stock_scrape():
```

Figure5.4.1-3 Creating function to delete all stock items from each table.

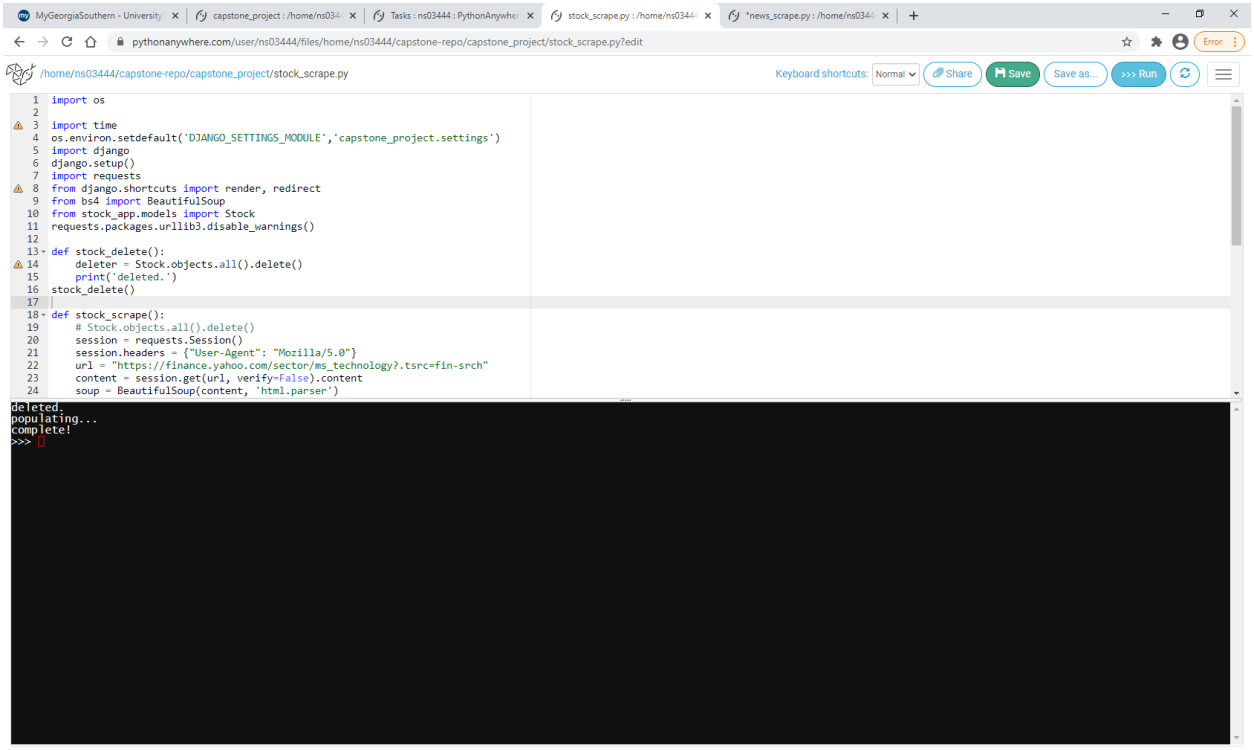


Figure5.4.1-4 Running news & stock scrape files with delete functions added.

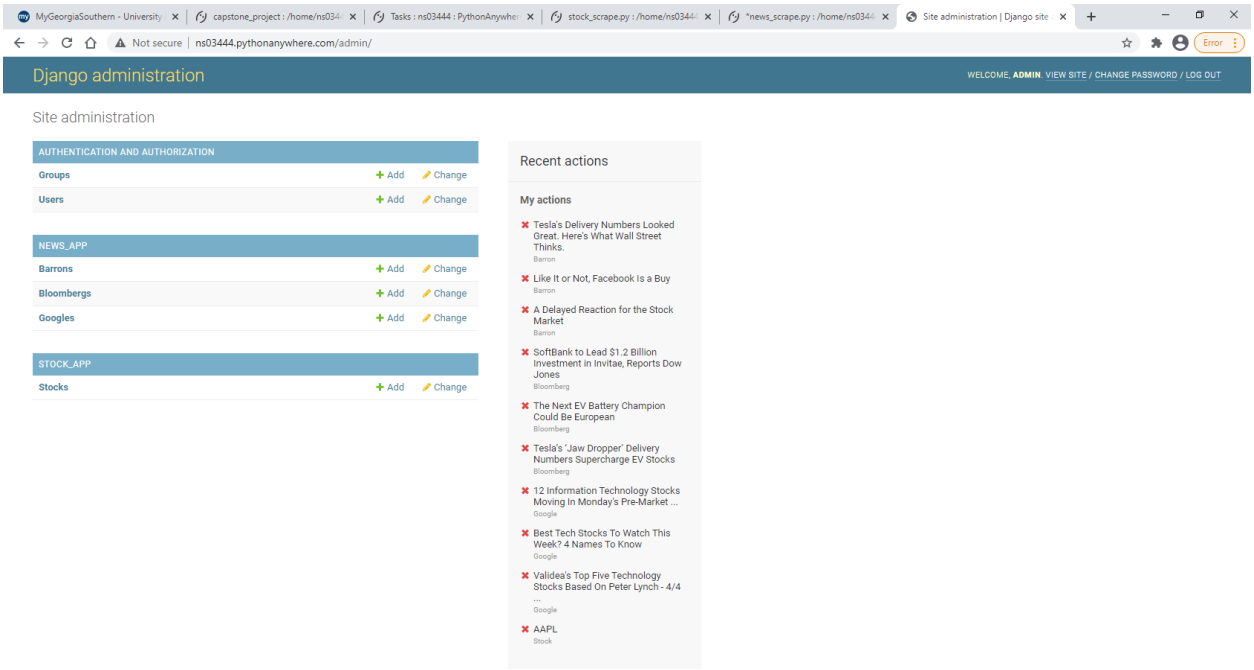


Figure5.4.1-5 Django admin panel indicating that items have recently been removed.

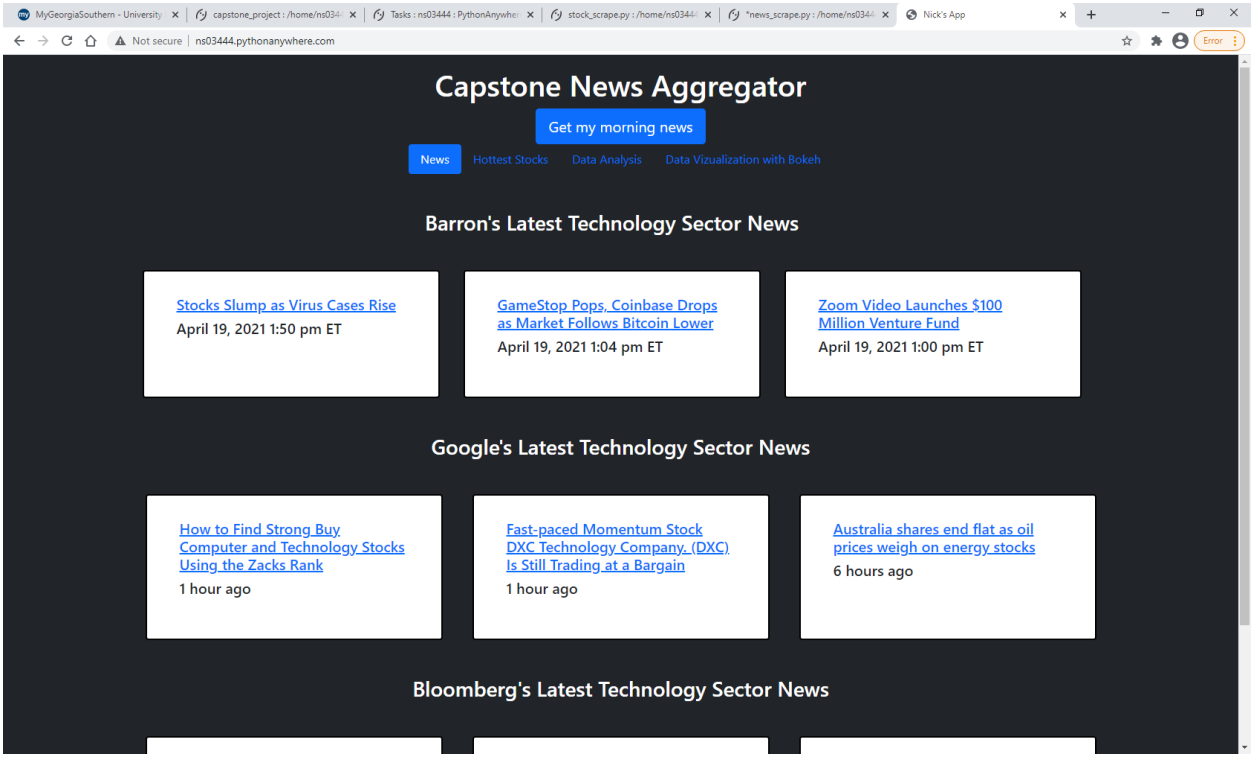


Figure5.4.1-6 News content updated.

After both scripts ran, I checked the webpage content to make sure they worked correctly and found that it did! Now I will just need to schedule them to run.

6.4.2 Scheduling the tasks

Python Anywhere offers “scheduled” tasks, as well as, “always-on” tasks, however, with the free account, I am only able to schedule one task to run once a day. Since I did not feel that this would be sufficient for updating the content as often as I wanted, I upgraded my subscription to the cheapest paid account type for $5.00 USD/month.

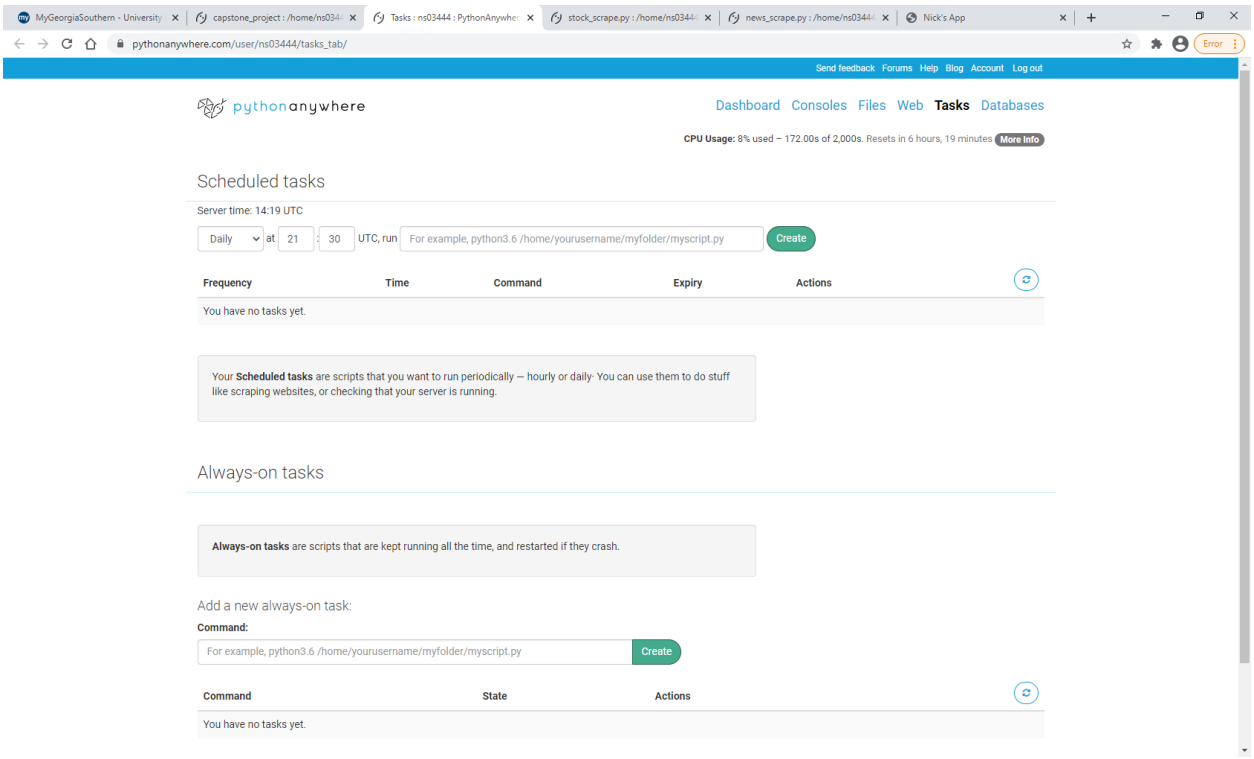


Figure5.4.2-1 Python Anywhere task tab.

Now I can schedule the tasks to run on an hourly basis throughout the day.

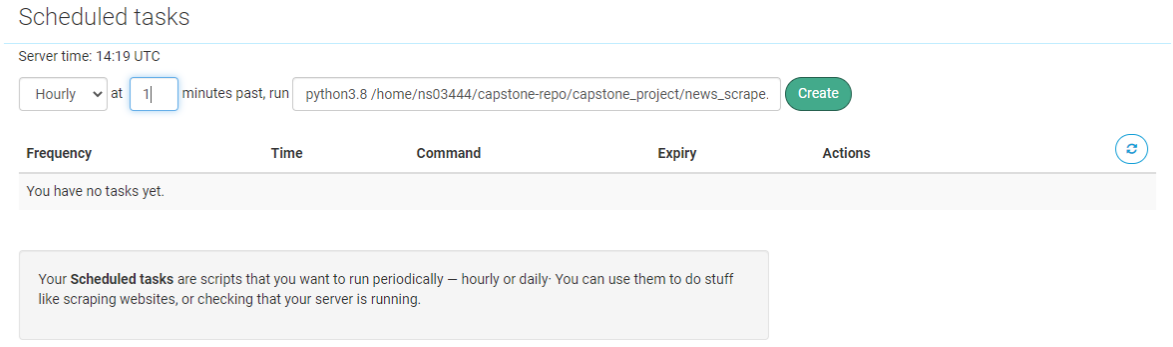


Figure5.4.2-2 Scheduling “news_scape.py” to run hourly.

By clicking the create button in the image above, I will create a scheduled task for running the "news_scrape.py" script, one minute past every new hour (ex. 12:01pm). Next, I will do the same for the "stock_scrape.py' file and it will be time test my final product!

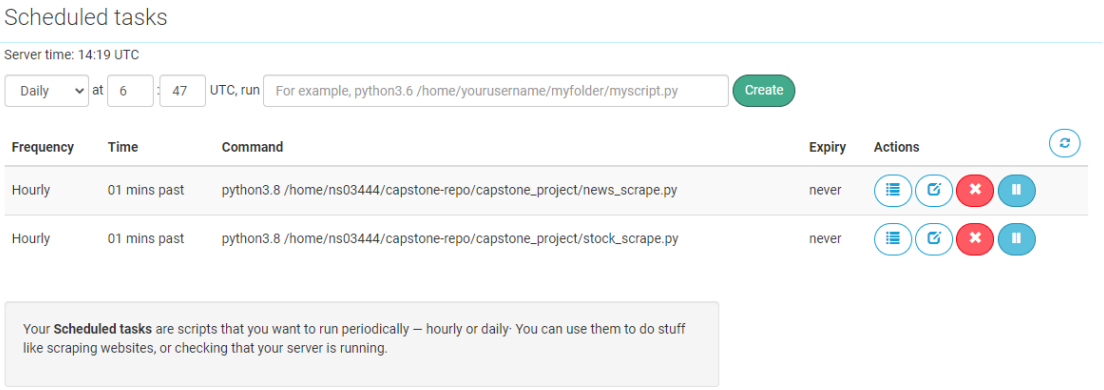


Figure5.4.2-2 Scheduling both tasks to run hourly.

6.4.3 Testing the Scheduler

As previously noted, the tasks are scheduled to run once every hour, I will now take screenshots of the webpages containing content before and after the scheduled tasks run at the beginning of the hour. Notice that some article titles stay the same, while the time changes. This is because the website's I am scraping the information from have not released any new articles, however, the amount of time the article has been posted updates. The prices on the stock performance table are also updated.

***One special note, I ran into time zone complications with the "Barron's" website, therefore, all times are displayed for their articles is in UTC time (four hours ahead).*

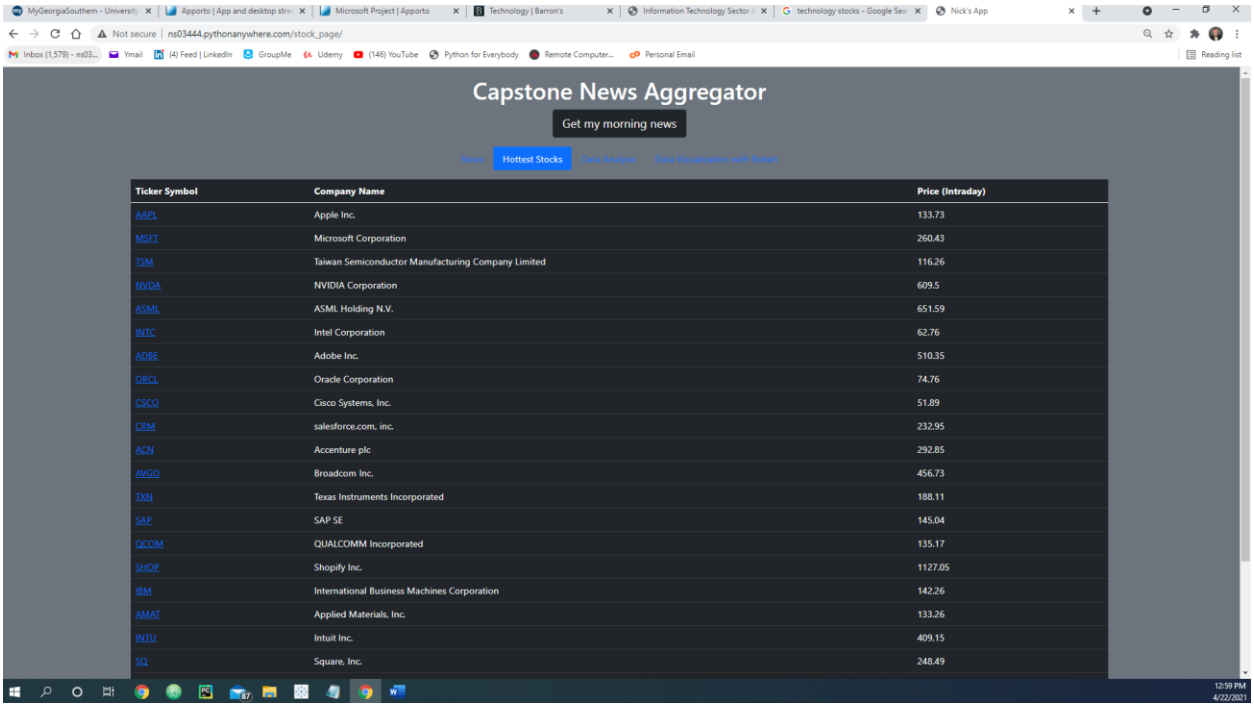


Figure5.4.3-1 Stock table content at 12:59pm 4/22/2021

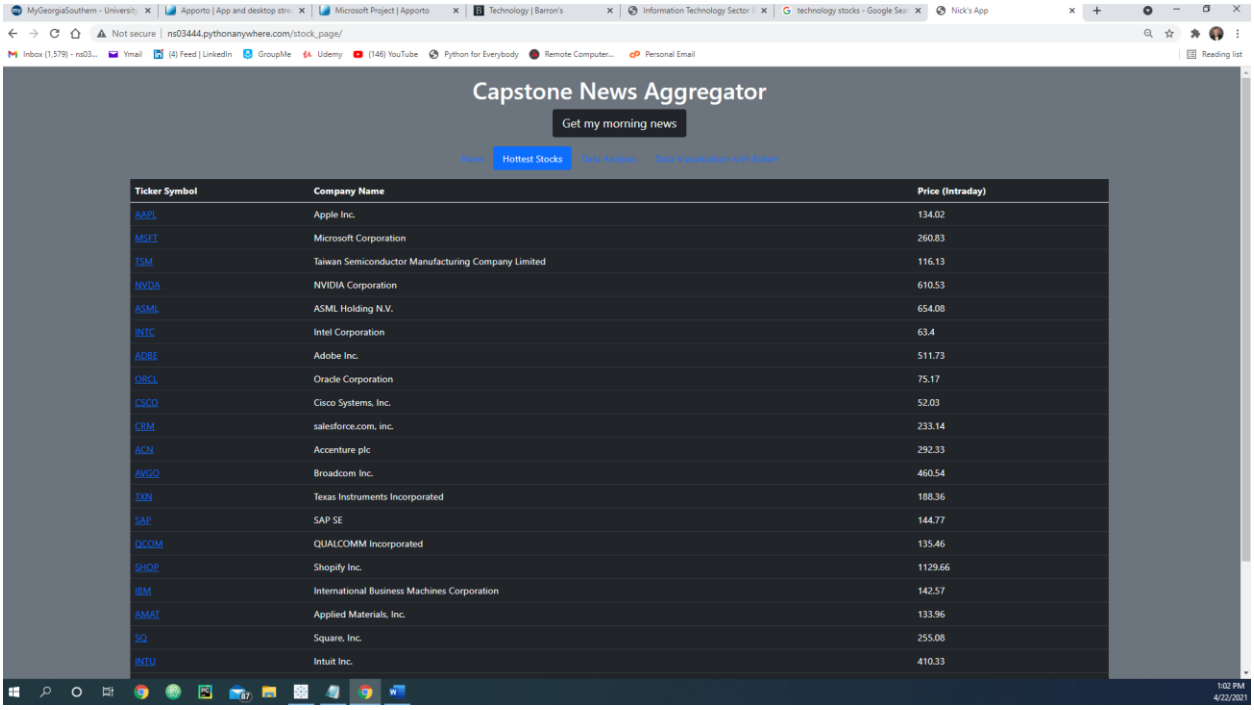


*Figure5.4.3-2 Stock table content at 1:02 4/22/2021- (***Prices have been updated)*

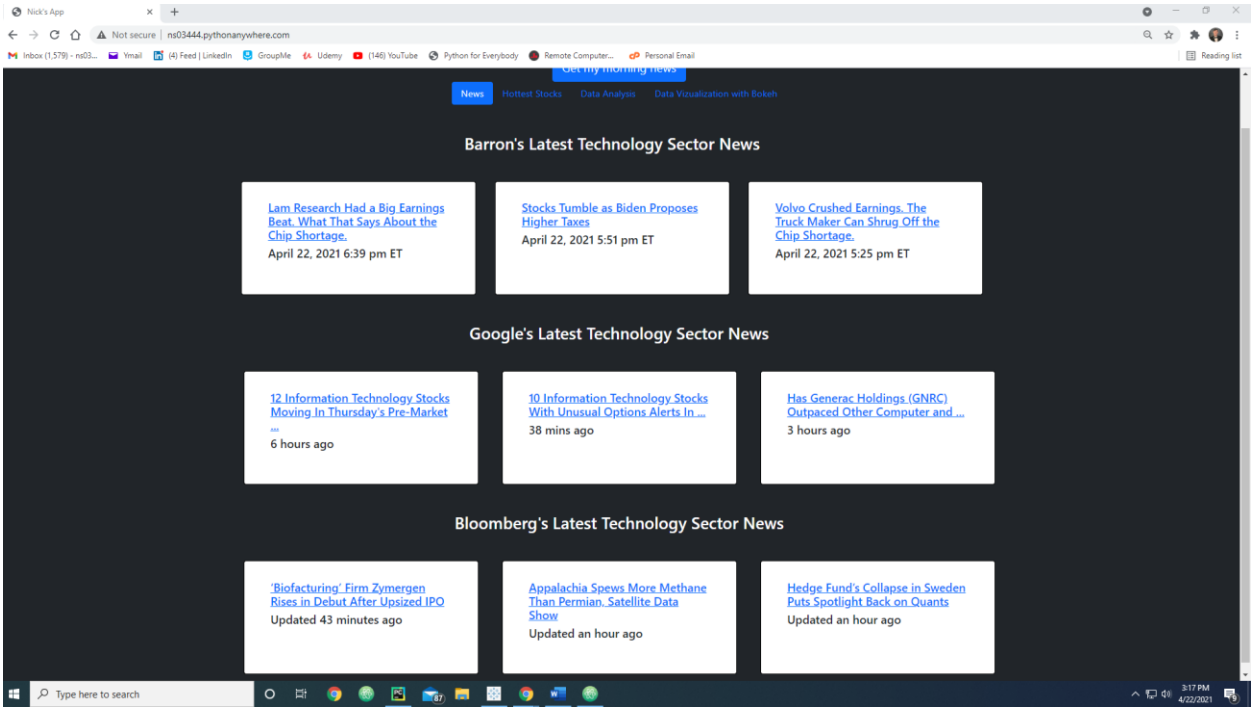


Figure5.4.3-3 News content at 3:17 4/22/2021

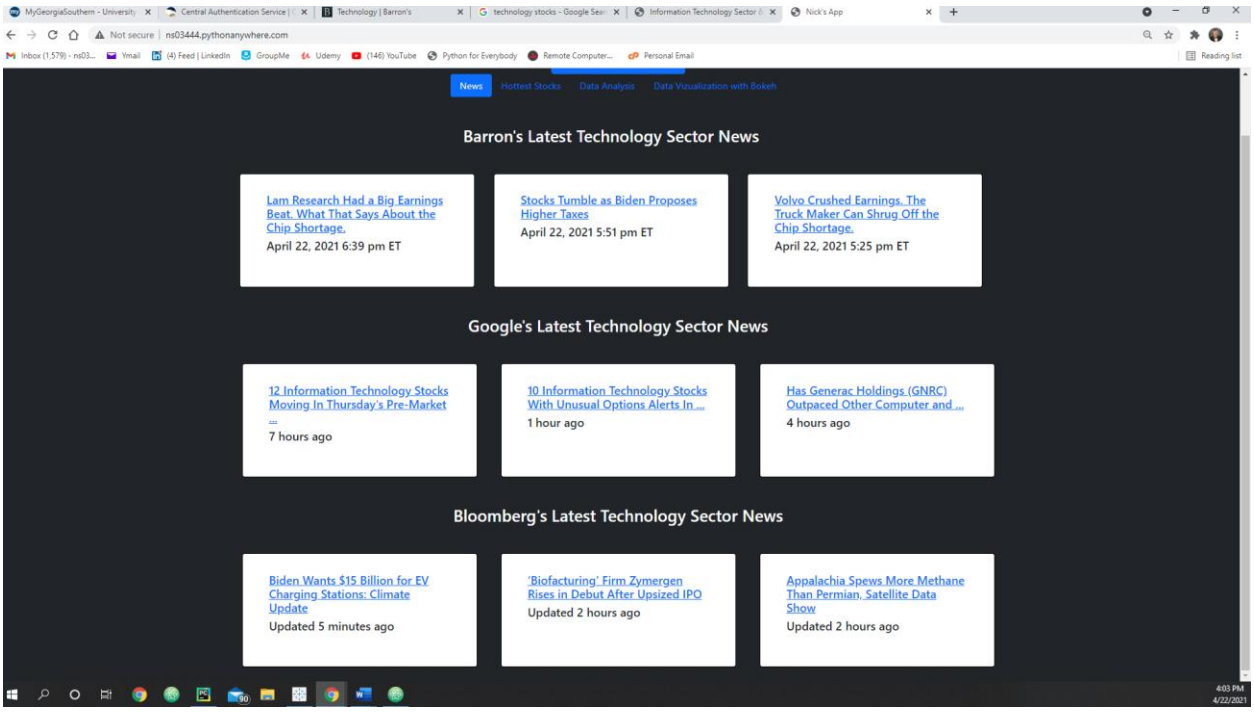
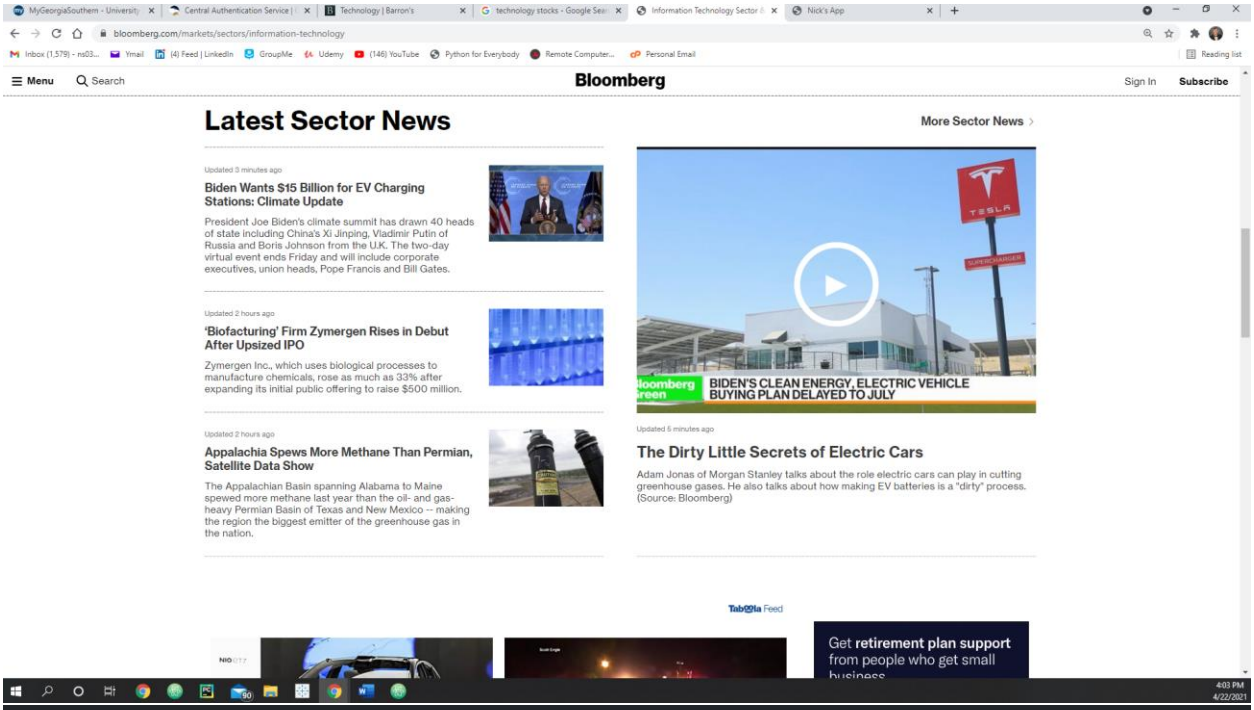


*Figure5.4.3-4 News content at 4:03 4/22/2021- (***Content has been updated)*



Bloomberg's Latest Technology Sector News

[Biden Wants $15 Billion for EV Charging Stations: Climate Update]

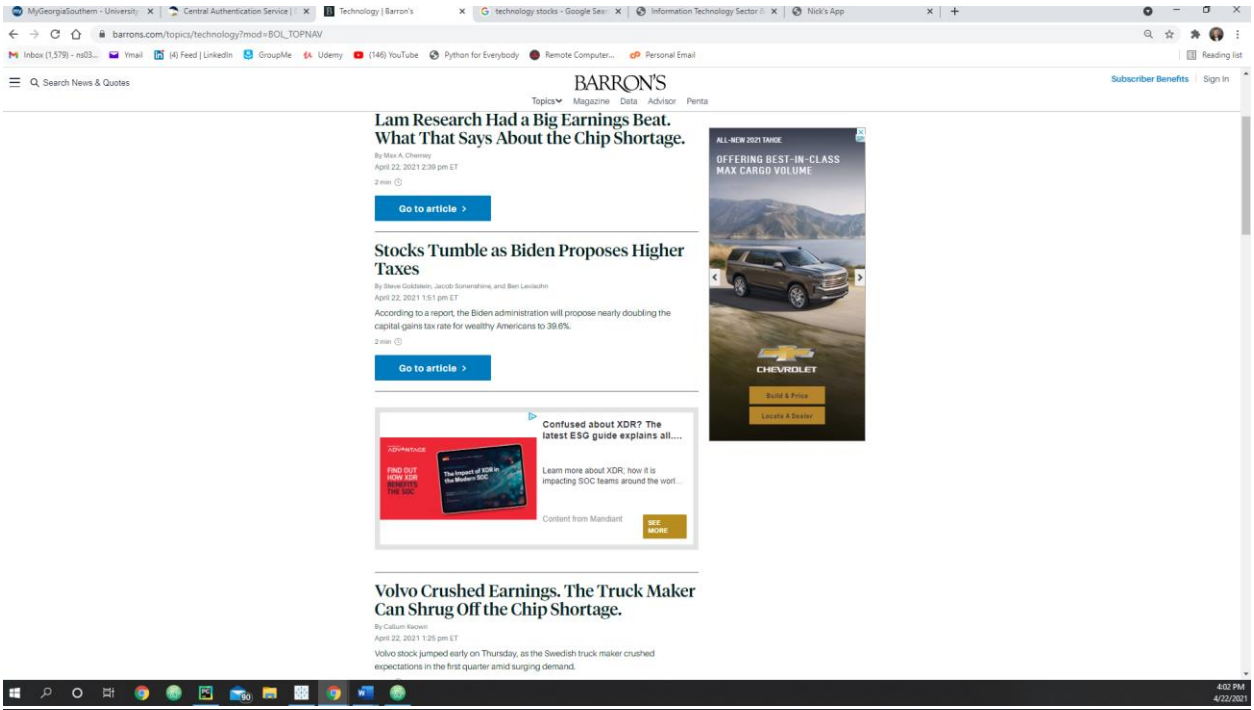
Updated 5 minutes ago

['Biofacturing' Firm Zymergen Rises in Debut After Upsized IPO]

Updated 2 hours ago

[Appalachia Spews More Methane Than Permian, Satellite Data Show]

Updated 2 hours ago

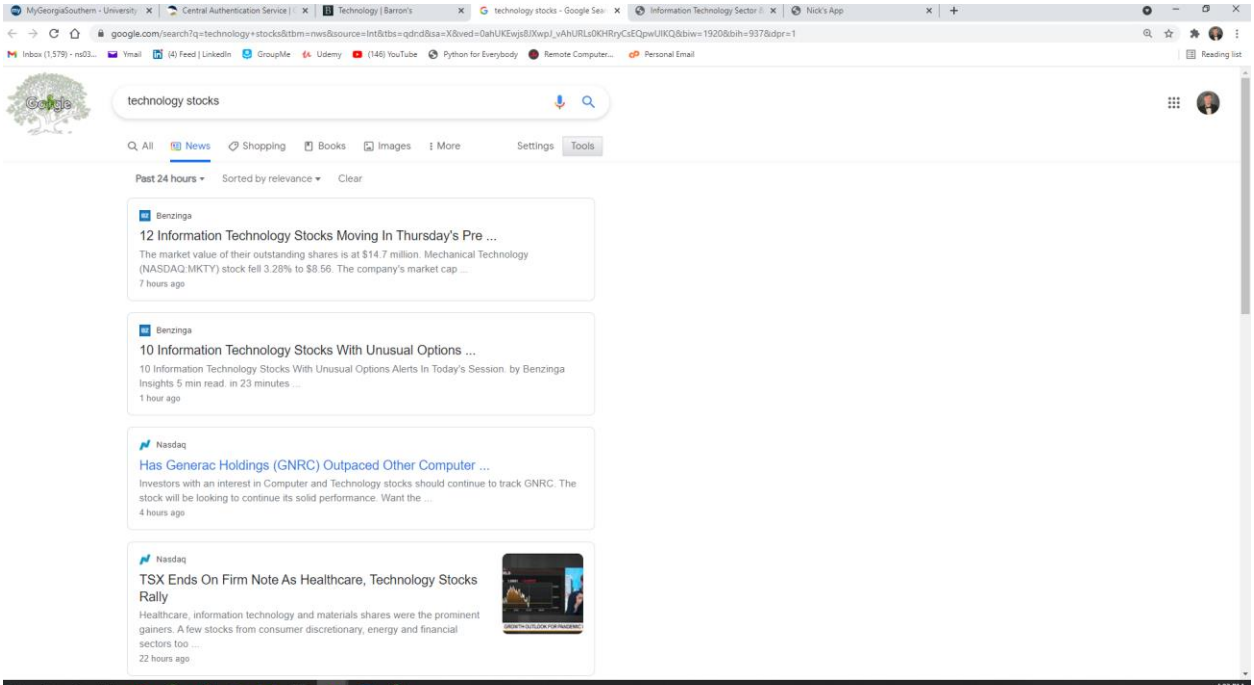


Barron's Latest Technology Sector News

[Lam Research Had a Big Earnings Beat. What That Says About the Chip Shortage.](#)
April 22, 2021 6:39 pm ET

[Stocks Tumble as Biden Proposes Higher Taxes](#)
April 22, 2021 5:51 pm ET

[Volvo Crushed Earnings. The Truck Maker Can Shrug Off the Chip Shortage.](#)
April 22, 2021 5:25 pm ET



Google's Latest Technology Sector News

[12 Information Technology Stocks Moving In Thursday's Pre-Market ...]

7 hours ago

[10 Information Technology Stocks With Unusual Options Alerts In ...]

1 hour ago

[Has Generac Holdings (GNRC) Outpaced Other Computer and ...]

4 hours ago

7 Reflection

Developing this application has provided me an indescribable learning experience that could not have been taught any other way. In my opinion, the project was a huge success for several reasons. One of the most important reasons being that all milestones were met, and the final product is functioning exactly as described in the project proposal. Moving forward, I will use the information I learned from creating this project throughout my professional career.