<https://medium.com/@dd93/collecting-weather-data-to-boost-data-science-models-with-selenium-390d9db88210> - webscraping

Collecting Weather Data to Boost Data Science Models with Selenium

[image]

Have you ever wonder how weather data would affect your data set or wanted to understand the historical weather patterns in your local city? Do you find it difficult to find weather data for your specific needs? I did.

I was working on a Kaggle competition involving geospatial data and I wanted more relevant data that was compatible with my data set. Unfortunately, I couldn’t find anything. I spent days looking and all I found was a website that would only provide weather condition such as whether or not the skies were cloudy or clear on a given day. That wasn’t enough for me.

TLDR: So instead, I built my own web scrapper, collected my own data, and applied it to my data set. The results led to a model that was more generalizable to new data sets. And now you can improve your results too with this tutorial.

I used Selenium to build a web scraper with Python to collect weather information such as Temperature, Humidity, Wind Speed, and Weather Conditions from [https://www.wunderground.com/](https://www.wunderground.com/" \t "_blank).

[image]

Interface of wunderground.com

What makes this website different from your typical websites is that the information is based on the weather conditions of nearby airports. While this sounds counter-intuitive, it is in fact generalizable to cities close to those airports and more informative than typical sites. This makes it easy to search for the closest airports using zip codes.

If you scroll all the way down to the bottom, you’ll see a gold mine of weather data for you to mine. Beautiful isn’t it? You can mine this for specific days or a sequence of days.

[image]

Here are my steps on how I collect the data from this table for a range of dates. You can skip to the bottom of the page if you want to see the full code snippet.

**Step 1**: Load the packages need

import numpy as np  
import pandas as pd  
import time  
import datetime  
import re  
from selenium import webdriver

Selenium is a tool used to automate tasks and actions with a browser through programming languages such as Python or Java. You can think about it as controlling your web browser with commands that you provide it. We are going to use it to collect data from HTML webpages.

To initialize Selenium, you’ll need to load the webdriver at given path. (Here is the installation guide for Selenium using Python: [http://selenium-python.readthedocs.io/installation.html](http://selenium-python.readthedocs.io/installation.html" \t "_blank))

driver = webdriver.Chrome('/usr/local/bin/chromedriver')  
driver.get('[https://www.wunderground.com/history/airport/KSFO/2018/03/14/DailyHistory.html?req\_city=San%20Francisco&req\_state=CA&reqdb.zip=94128&reqdb.magic=4&reqdb.wmo=99999](https://www.wunderground.com/history/airport/KSFO/2018/03/14/DailyHistory.html?req_city=San%20Francisco&req_state=CA&reqdb.zip=94128&reqdb.magic=4&reqdb.wmo=99999" \t "_blank)')

webdriver.Chrome will load the required software to run Selenium on Chrome. driver.get() will open a new web browser that it can control.

**Step 2**: Get a sequence of dates to search for

def list\_dates(start,end):  
 """ This creates a list of of dates between the 'start' date and the 'end' date """ # create datetime object for the start and end dates  
 start = datetime.datetime.strptime(start, '%Y-%m-%d')  
 end = datetime.datetime.strptime(end, '%Y-%m-%d') # generates list of dates between start and end dates  
 step = datetime.timedelta(days=1)  
 dates = []  
 while start <= end:  
 dates.append(start.date())  
 start += step # return the list of dates in string format  
 return [str(date) for date in dates]

I wrote the list\_dates() function to produce a sequence of dates from a start date to an end date. This is useful because you won’t need to need to manually input dates.

# this dictionary is used to map the months produced in the previous function with the full name of the month   
 month\_dict ={  
 1:'January',  
 2:'February',  
 3:'March',  
 4:'April',  
 5:'May',  
 6:'June',  
 7:'July',  
 8:'August',  
 9:'September',  
 10:'October',  
 11:'November',  
 12:'December'  
}def date\_part(data,f\_mat ='%Y-%m-%d'):  
 """Extracts the date information produced by list\_dates() for Month,Day,Year """ # creates a pandas dataframe of dates  
 dates = pd.DataFrame(data,columns=['date'])  
 date\_time = dates['date']  
 fld = pd.to\_datetime(date\_time, format=f\_mat)  
 for n in ('Month', 'Day','Year'):  
 dates[n] = getattr(fld.dt,n.lower())  
 dates['Month'] = dates['Month'].map(month\_dict)  
 return dates

The date\_part() function is used to break down the dates produced by list\_date() into Month, Day, and Year so that it can easily be sent to wunderground to search for a date to collect data from. For example, both functions will produce the following pandas dataframe:

start = '2014-1-1'  
end = '2014-1-4'  
date = list\_dates(start,end)  
date = date\_part(date,'%Y-%m-%d')

[image]

Date dictionary to be used to scrape specific weather dates

**Step 3**: Search by location

Now that we have the date dictionary, you will need to provide a zip code so that the wunderground can pull up the weather information. You can automate this task with Selenium. If you go to the search bar, right-click it, and click inspect, you will get HTML tag information about that search bar. This is generalizable to any HTML-JavaScript-CSS feature on the web page. For example, here you can see that the unique ‘id’ for the search bar:

[image]

Inspecting source code of wunderground web page

[image]

Inside the developer source with the tag information

To send a zip code to the search bar, the following code will look for the search bar by a specific ‘id’ with ‘find\_element\_by\_id’, clear the search bar, send the zip code, and finally submit it to wunderground to search for the city.

zipcode='91770'  
search=driver.find\_element\_by\_id('history-icao-search')  
search.clear() # clears field  
search.send\_keys(zipcode) # send zipcode to search field  
search.submit() # submits zip code to search

**Step 4**: Search for date

[image]

You can similarly search for dates like Step 3. By looking for HTML tags associated to the ‘Weather History Date’ pull down menus, you’ll see that it contains class attributes such as month, day and year. We can use this to send information to search for those pull down menus. You can iterate through the pandas data frame produced from Step 2 to search the weather of a given date.

Month='March'  
Day='14'  
Year='2018'month = driver.find\_element\_by\_class\_name('month')  
month.send\_keys(Month)  
day = driver.find\_element\_by\_class\_name('day')  
day.send\_keys(Day)  
year = driver.find\_element\_by\_class\_name('year')  
year.send\_keys(Year)  
year.submit()

**Step 5**: Collect weather data

[image]

We’re now ready to collect the weather data. We can see that the weather table is contained in this div tag with unique ‘id’ of ‘observations\_details’. We can use this information to scrape the whole weather table with a simple line of code:

weatherdata = driver.find\_elements\_by\_id('observations\_details')[0].text

**Step 6**: Bring it all together

Finally, we can combine all the previous steps into a compact script. I have also including some preprocessing steps and a function in runs everything with one function:

from selenium import webdriver  
import time  
import numpy as np  
import pandas as pd  
import datetime  
import pickle  
import redef list\_dates(start,end):  
 """ This creates a list of of dates between the 'start' date and the 'end' date """  
 # create datetime object for the start and end dates  
 start = datetime.datetime.strptime(start, '%Y-%m-%d')  
 end = datetime.datetime.strptime(end, '%Y-%m-%d')  
 # generates list of dates between start and end dates  
 step = datetime.timedelta(days=1)  
 dates = []  
 while start <= end:  
 dates.append(start.date())  
 start += step  
 # return the list of dates in string format  
 return [str(date) for date in dates]# this dictionary is used to map the months produced in the previous function with the full name of the month   
month\_dict ={  
 1:'January',  
 2:'February',  
 3:'March',  
 4:'April',  
 5:'May',  
 6:'June',  
 7:'July',  
 8:'August',  
 9:'September',  
 10:'October',  
 11:'November',  
 12:'December'  
}  
def date\_part(data,f\_mat ='%Y-%m-%d'):  
 """Extracts the date information produced by list\_dates() for Month,Day,Year """  
 # creates a pandas dataframe of dates  
 dates = pd.DataFrame(data,columns=['date'])  
 date\_time = dates['date']  
 fld = pd.to\_datetime(date\_time, format=f\_mat)  
 for n in ('Month', 'Day','Year'):  
 dates[n] = getattr(fld.dt,n.lower())  
 dates['Month'] = dates['Month'].map(month\_dict)  
 return datesdef scrapper(dates,zipcode):  
 data=[] # list to append scrapped data  
 # submits the zipcode to find the closest weather center  
 search= driver.find\_element\_by\_xpath('//\*[[@id](http://twitter.com/id" \t "_blank)="history-icao-search"]')  
 search.clear()  
 search.send\_keys(zipcode)  
 search.submit()  
 time.sleep(3) # sleep timer to wait for page to load (not necessary)  
   
 # iterates through provided list of dates to scrap weather for  
 for i,v in dates.iterrows():  
 # inputs month, day, year into website to view information  
 month = driver.find\_element\_by\_class\_name('month')  
 month.send\_keys(v['Month'])  
 day = driver.find\_element\_by\_class\_name('day')  
 day.send\_keys(v['Day'])  
 year = driver.find\_element\_by\_class\_name('year')  
 year.send\_keys(v['Year'])  
 year.submit() # submits to search for month, day, year  
# time.sleep(3) # sleep timer to wait for page to load (not necessary)# scraps table on the bottom for weather information  
 weatherdata = driver.find\_elements\_by\_id('observations\_details') # locates the data  
 x = weatherdata[0].text # scrapes that data  
 x= re.sub(r'[^\x00-\x7F]+',' ', x) # removes unicode  
 x = x.split('\n') # breaks the data into observations per row  
 x = x[1:-1] # removes the last line  
 data.extend([i+' '+v['date'] for i in x]) # appends all scraped data  
 return datadef preprocess\_data(data):  
 """Preprocess the scraped data and load the data into a pandas dataframe"""  
 dt = [i.replace('Calm Calm', 'Calm 0.0 mph') for i in data]  
 dt = [i.replace(' AM', 'AM') for i in dt]  
 dt = [i.replace(' PM', 'PM') for i in dt]  
 dt = [i.replace('%', '') for i in dt]   
 dt = [i.replace(' mi', '') for i in dt]  
 dt = [i.replace(' mph', '') for i in dt]  
 dt = [i.replace(' in', '') for i in dt]  
 dt = [re.sub(' +',' ',i) for i in dt]dt = [i.replace('Mostly ', 'Mostly') for i in dt]  
 dt = [i.replace('Partly ', 'Partly') for i in dt]  
 dt = [i.replace('Scattered ', 'Scattered') for i in dt]  
 dt = [i.replace('Light ', 'Light') for i in dt]  
 dt = [i.replace('Heavy ', 'Heavy') for i in dt]  
   
 dt = [i.replace('Fog , Rain', 'Rain') for i in dt]  
 dt = [i.replace('Fog , Snow', 'FogSnow') for i in dt]  
 dt = [i.replace('Fog', ' ',1) for i in dt]  
 dt = [i.replace('Rain , Thunderstorm', 'RainThunderstorm') for i in dt]  
   
 dt = [i.replace('Thunderstorm', '',1) for i in dt]  
 dt = [i.replace('Thunderstorms and Rain', 'ThunderstormsandRain') for i in dt]dt = [i.replace('Rain', '',1) for i in dt]  
 dt = [i.replace('Snow', '',1) for i in dt]  
# dt = [i.replace('Light Drizzle', 'LightDrizzle') for i in dt]  
   
 dt = [i.replace('F', '',) for i in dt]  
 dt = [i.replace(' og', ' Fog') for i in dt]  
 dt = [i.replace('Patches of Fog', 'PatchesofFog',1) for i in dt]  
 dt = [i.replace('Lightreezing Rain', 'LightFreezingRain') for i in dt]  
   
 dt = [i.split() for i in dt]  
 dt = [ i[:2] +i[-10:] for i in dt]  
   
 dt = pd.DataFrame(dt,columns = ['time','temp(F)','dewpoint(F)','humidity(%)','pressure(in)','visibility(mi)','winddir','windspeed(mph)','gustspeed(mph)','precip(in)','conditions','date'])  
 dt['time'] = [datetime.datetime.strftime(datetime.datetime.strptime(val, "%I:%M%p"), "%H:%M") for val in dt['time']]  
 return dtdef weather\_scrapper(start\_date,end\_date, zipcode):  
 """final webscrapper function"""  
 dates = list\_dates(start\_date,end\_date)  
 dates = date\_part(dates,'%Y-%m-%d')  
 data = scrapper(dates,zipcode)  
 return preprocess\_data(data)####################################################################driver = webdriver.Chrome('/usr/local/bin/chromedriver')  
x = '[https://www.wunderground.com/history/airport/KSFO/2018/2/24/DailyHistory.html?req\_city=San%20Francisco&req\_statename=California'](https://www.wunderground.com/history/airport/KSFO/2018/2/24/DailyHistory.html?req_city=San%20Francisco&req_statename=California%27" \t "_blank)  
driver.get(x)# YYYY-MM-DD  
start = '2014-1-1'  
end = '2014-1-5'  
zipcode = '10001'weather\_scrapper(start,end,zipcode)

Congratulations! Now you can collect your weather data with Selenium and enhance all your Kaggle / work / school / side projects without compromising validation scores! You can also use what you’ve from this and collect data from other sources too!

References:

1. [Weather Underground](https://www.wunderground.com/" \t "_blank)
2. [Selenium Download Page](https://www.seleniumhq.org/download/" \t "_blank)
3. [Personal GitHub Repository](https://github.com/davidkes/Weather-Data-Scaper" \t "_blank)

<https://www.wunderground.com/>

<https://www.seleniumhq.org/download/>

<https://github.com/davidkes/Weather-Data-Scaper>