

# DSC 540 Week 10

Kannur, Gyan

## Activity 9: Extracting the Top 100 eBooks from Gutenberg

All steps details below

Import the necessary libraries, including regex and BeautifulSoup.

```
# Load the necessary libraries.
import urllib.request, urllib.parse, urllib.error
import requests
import tweepy
from bs4 import BeautifulSoup
import ssl
import re
import warnings
warnings.filterwarnings("ignore")
```

- Check the SSL certificate.

```
#There may be certificate error due to invalid certificates, Ignore this error
certx = ssl.create_default_context()
certx.check_hostname = False
certx.verify_mode = ssl.CERT_NONE

# function to check the response status code, 200 means OK, any other status means the request failed

def status_check(r):
    if r.status_code==200:
        print("Success!")
        return 0
    else:
        print("Failed!")
        return 1
```

- Read the HTML from the URL. Write a small function to check the status of the web request.

```
# Read the HTML from the URL and check the response
gutenbergurl = 'https://www.gutenberg.org/browse/scores/top'
response = requests.get(gutenbergurl)
status_check(response)

Success!
```

- Decode the response and pass this on to BeautifulSoup for HTML parsing.

*# Decode the response and pass on to BeautifulSoup for HTML parsing*

```
urlContent = response.content.decode(response.encoding)
soup = BeautifulSoup(urlContent, 'html.parser')
```

- Find all the href tags and store them in the list of links.
- Check what the list looks like – print the first 30 elements

*# Find all the href tags and store them in the list of links.*

*# Empty list to hold all the http links in the HTML page*

```
href_list=[]
```

```
for item in soup.find_all('a'):
    href_list.append(item.get('href'))
```

*# Check what the list looks like – print the first 30 elements.*

```
href_list[:30]
```

```
['/',
 '/about/',
 '/about/',
 '/policy/collection_development.html',
 '/about/contact_information.html',
 '/about/background/',
 '/policy/permission.html',
 '/policy/privacy_policy.html',
 '/policy/terms_of_use.html',
 '/ebooks/',
 '/ebooks/',
 '/ebooks/bookshelf/',
 '/browse/scores/top',
 '/ebooks/offline_catalogs.html',
 '/help/',
 '/help/',
 '/help/copyright.html',
 '/help/errata.html',
 '/help/file_formats.html',
 '/help/faq.html',
 '/policy/',
 '/help/public_domain_ebook_submission.html',
 '/help/submitting_your_own_work.html',
 '/help/mobile.html',
 '/attic/']
```

```

'/donate/',
'/donate/',
'pretty-pictures',
'#books-last1',
'#authors-last1']

```

- Use a regular expression to find the numeric digits in these links. These are the file numbers for the top 100 eBooks.
- Initialize the empty list to hold the file numbers over an appropriate range and use regex to find the numeric digits in the link href string.

```

#Initialize the empty list to hold the file numbers over an
appropriate range and use regex to
# find the numeric digits in the link href string
# Number 19 to 119 in the original list of links have the Top 100
ebooksnumber.

```

```

filenum =[]

for i in range(19,119):
    link=href_list[i]
    link=link.strip()
    # Use the findall method. What does the soup object's text look
like?

    num=re.findall('[0-9]+',link)
    if len(num)==1:
        # Append the filename casted as integer
        filenum.append(int(num[0]))

```

```

# Print the file numbers
print("file numbers for the top 100 ebooks on Gutenberg are\n")
print(filenum)

```

file numbers for the top 100 ebooks on Gutenberg are

```

[1, 1, 7, 7, 30, 30, 25558, 84, 2701, 1513, 2641, 46, 37106, 100, 145,
11, 1342, 67979, 16389, 394, 6761, 6593, 2160, 5197, 4085, 1259,
64317, 25344, 2554, 75076, 2542, 43, 174, 75074, 345, 76, 5200, 844,
1400, 1080, 98, 50150, 1260, 2000, 2650, 1184, 1661, 28054, 6130,
75075, 1998, 1952, 1727, 2600, 1232, 3207, 2591, 16119, 55, 31552,
4300, 74, 36034, 219, 75077, 4363, 30254, 7370, 16328, 768, 75070,
408, 45, 23, 75071, 37423, 3296, 5740, 1497, 27827, 2814, 10007, 8800,
996, 205, 730, 31284, 24518, 41, 41445, 67098]

```

- Use the findall method. What does the soup object's text look like? Use the .text method and print only the first 2,000 characters (do not print the whole thing, as it is too long).

```
# Use the .text method and print only the first 2,000 characters  
# (do not print the whole thing, as it is too long).
```

```
print(soup.text[:2000])
```

Top 100 | Project Gutenberg

Menu▼

About



About Project Gutenberg  
Collection Development  
Contact Us  
History & Philosophy  
Permissions & License  
Privacy Policy

Terms of Use

Search and Browse



Book Search

Bookshelves

Frequently Downloaded

Offline Catalogs

Help



All help topics →

Copyright How-To

Errata, Fixes and Bug Reports

File Formats

Frequently Asked Questions

Policies →

Public Domain eBook Submission

Submitting Your Own Work

Tablets, Phones and eReaders

The Attic →

Donate

Ways to donate

To determine the ranking we count the times each file gets downloaded. Both HTTP and FTP transfers are counted. Only transfers from ibiblio.org are counted as we have no access to our mirrors log files. Multiple downloads from the same IP address on the same day count as one download. IP addresses that download more than 100 files a day are considered robots and are not considered. Books made out of multiple files like most audio books are counted if any file is downloaded.

#### Downloaded Books

2024-12-14404564

last 7 days3319543

last 30 days13680338

#### Pretty Pictures

Top 100 EBooks yesterday –

Top 100 Authors yesterday –

Top 100 EBooks last 7 days –

Top 100 Authors last 7 days –

Top 100 EBooks last 30 days –

Top 100 Authors last 30 days

#### Top 100 EBooks yesterday

呻吟語 by Kun Lü (20918)

歸蓮夢 by active 18th century Su'anzhuren (4999)

Frankenstein; Or, The Modern Prometheus by Mary Wollstonecraft Shelley (2359)

Moby Dick; Or, The Whale by Herman Melville (2220)

Romeo and Juliet by William Shakespeare (2082)

A Christmas Carol in Prose; Being a Ghost Story of Christmas by Charles Dickens (1975)

Middlemarch by George Eliot (1651)

The Complete Works of William Shakespeare by William Shakespeare (1627)

A Room with a View by E. M. Forster (1626)

Little Women; Or, Meg, Jo, Beth, and Amy by Louisa May Alcott (1549)

The Blue Castle: a novel by L. M. Montgomery (1437)

The Enchanted A

- Search in the extracted text (using a regular expression) from the soup object to find the names of the top 100 eBooks (yesterday's ranking).

```
# Initialize a temp list to hold titles
```

```

title_list_temp=[]

# Creating a start index pointing at the text "Top 100 Ebooks
yesterday"

index_start = soup.text.splitlines().index('Top 100 EBooks yesterday')

# Loop 1-100 to add the strings of next 100 lines to this temporary
list.

for item in range(107):
    title_list_temp.append(soup.text.splitlines()[index_start+2+item])

#Print 1 record
print(title_list_temp[1])

Frankenstein; Or, The Modern Prometheus by Mary Wollstonecraft Shelley
(3920)

```

- Use match and span to find the indices and use them.
- Print the list of titles

```

# Use regular expression to extract only text from the name strings
and append to an empty list

title_list=[]
for i in range(7,107):
    id1,id2=re.match('[a-zA-Z ]*',title_list_temp[i]).span()
    title_list.append(title_list_temp[i][id1:id2])

# Printing List of Titles
for item in title_list:
    print(item)

The Complete Works of William Shakespeare by William Shakespeare
Middlemarch by George Eliot
Alice
Pride and Prejudice by Jane Austen
The Blue Castle
The Enchanted April by Elizabeth Von Arnim
Cranford by Elizabeth Cleghorn Gaskell
The Adventures of Ferdinand Count Fathom
History of Tom Jones
The Expedition of Humphry Clinker by T
My Life
The Adventures of Roderick Random by T
Twenty years after by Alexandre Dumas and Auguste Maquet
The Great Gatsby by F
The Scarlet Letter by Nathaniel Hawthorne

```

Crime and Punishment by Fyodor Dostoyevsky  
An interrupted night by Pansy  
A Doll  
The Strange Case of Dr  
The Picture of Dorian Gray by Oscar Wilde  
Elementary cryptanalysis by Helen Fouch  
Dracula by Bram Stoker  
Adventures of Huckleberry Finn by Mark Twain  
Metamorphosis by Franz Kafka  
The Importance of Being Earnest  
Great Expectations by Charles Dickens  
A Modest Proposal by Jonathan Swift  
A Tale of Two Cities by Charles Dickens  
The Devil is an Ass by Ben Jonson  
Jane Eyre  
Don Quijote by Miguel de Cervantes Saavedra  
Du c  
The Count of Monte Cristo by Alexandre Dumas and Auguste Maquet  
The Adventures of Sherlock Holmes by Arthur Conan Doyle  
The Brothers Karamazov by Fyodor Dostoyevsky  
The Iliad by Homer  
Salvage  
Thus Spake Zarathustra  
The Yellow Wallpaper by Charlotte Perkins Gilman  
The Odyssey by Homer  
War and Peace by graf Leo Tolstoy  
The Prince by Niccol  
Leviathan by Thomas Hobbes  
Grimms  
Doctrina Christiana  
The Wonderful Wizard of Oz by L  
Novo dicion  
Ulysses by James Joyce  
The Adventures of Tom Sawyer  
White Nights and Other Stories by Fyodor Dostoyevsky  
Heart of Darkness by Joseph Conrad  
The structure of the English sentence by Lillian Kimball Stewart  
Beyond Good and Evil by Friedrich Wilhelm Nietzsche  
The Romance of Lust  
Second Treatise of Government by John Locke  
Beowulf  
Wuthering Heights by Emily Bront  
Larry Dexter and the bank mystery  
The Souls of Black Folk by W  
Anne of Green Gables by L  
Narrative of the Life of Frederick Douglass  
Bad and mad by W  
How We Think by John Dewey  
The Confessions of St



Tractatus Logico  
The Republic by Plato  
The Kama Sutra of Vatsyayana by Vatsyayana  
Dubliners by James Joyce  
Carmilla by Joseph Sheridan Le Fanu  
The divine comedy by Dante Alighieri  
Don Quixote by Miguel de Cervantes Saavedra  
Walden  
Oliver Twist by Charles Dickens  
Josefine Mutzenbacher by Felix Salten  
Memoirs of Extraordinary Popular Delusions and the Madness of Crowds  
by Charles Mackay  
The Legend of Sleepy Hollow by Washington Irving  
Frankenstein  
Winnie  
The Story of the Great War  
Meditations by Emperor of Rome Marcus Aurelius  
The War of the Worlds by H  
A Study in Scarlet by Arthur Conan Doyle  
Les Mis  
Little Women by Louisa May Alcott  
Aesop  
Peter Pan by J  
The Hound of the Baskervilles by Arthur Conan Doyle  
Les mis  
Anna Karenina by graf Leo Tolstoy  
Le Morte d  
The King in Yellow by Robert W  
Ethan Frome by Edith Wharton  
Hamlet

Top

Top

Top

Top

Top

## Activity 10: Building Your Own Movie Database by Reading an API

The aims of this activity are as follows: To retrieve and print basic data about a movie (the title is entered by the user) from the web (OMDb database) If a poster of the movie can be found, it downloads the file and saves it at a user-specified location These are the steps that will help you solve this activity:

- Import urllib.request, urllib.parse, urllib.error, and json.
- Load the secret API key (you have to get one from the OMDb website and use that; it has a daily limit of 1,000) from a JSON file stored in the same folder in a variable, by using json.loads.

- Obtain a key and store it in JSON as APIkeys.json.
- Open the APIkeys.json file.
- Assign the OMDb portal (<http://www.omdbapi.com/?>) as a string to a variable.
- Create a variable called apikey with the last portion of the URL (&apikey=secretapikey), where secretapikey is your own API key.
- Write a utility function called print\_json to print the movie data from a JSON file (which we will get from the portal).
- Write a utility function to download a poster of the movie based on the information from the JSON dataset and save it in your local folder. Use the os module. The poster data is stored in the JSON key Poster. Use the Python command to open a file and write the poster data. Close the file after you're done. This function will save the poster data as an image file.
- Write a utility function called search\_movie to search for a movie by its name, print the downloaded JSON data, and save the movie poster in the local folder. Use a try-except loop for this. Use the previously created serviceurl and apikey variables. You have to pass on a dictionary with a key, t, and the movie name as the corresponding value to the urllib.parse.urlencode() function and then add the serviceurl and apikey to the output of the function to construct the full URL. This URL will be used to access the data. The JSON data has a key called Response. If it is True, that means the read was successful. Check this before processing the data. If it's not successful, then print the JSON key Error, which will contain the appropriate error message returned by the movie database.
- Test the search\_movie function by entering Titanic.
- Test the search\_movie function by entering "Random\_error" (obviously, this will not be found, and you should be able to check whether your error catching code is working properly).
- Import urllib.request, urllib.parse, urllib.error, and json.

```
# Loading Libraries
import urllib.request, urllib.parse, urllib.error
import json
from PIL import Image
import requests
```

- Load the secret API key (you have to get one from the OMDb website and use that; it has a daily limit of 1,000) from a JSON file stored in the same folder in a variable, by using json.loads.
- Obtain a key and store it in JSON as APIkeys.json.
- Open the APIkeys.json file.

```
#read the secret key
f = open('./datasets/APIKeys.json')

# returns JSON object as
# a dictionary
```

```
# getting the API Key from file
apiKey = ""
apiKeyDict = json.load(f)

for item in apiKeyDict.items():
    apiKey = str(item[1])
```

- Create a variable called apikey with the last portion of the URL (&apikey=secretapikey), where secretapikey is your own API key.

```
# assessing the portal value to variable
omdbBaseURL = "http://www.omdbapi.com/"

# variable api key to hold the key value
apikey = "&apikey=" + apiKey
```

- Write a utility function called print\_json to print the movie data from a JSON file (which we will get from the portal).

```
def print_json(data):
    for item in data.items():
        keyData = str(item[0])
        valData = str(item[1])
        if keyData == "Poster":
            # poster contains image, we would download and show the
            image
            im = Image.open(requests.get(valData, stream=True).raw)
            im.show()
        else:
            print(keyData + " : " + valData + "\n")
```

- Write a utility function to download a poster of the movie based on the information from the JSON dataset and save it in your local folder. Use the os module. The poster data is stored in the JSON key Poster. Use the Python command to open a file and write the poster data. Close the file after you're done. This function will save the poster data as an image file.

```
from urllib.request import urlopen
import os
import urllib.request, urllib.parse, urllib.error
#Function to download the poster
def download_poster(json_file):
    #Pulling the movie title and poster link from the JSON file
    title = str(json_file['Title'])
    poster_link = json_file['Poster']

    #Reading the poster image data by passing the poster link through
    the urllib.request library
    poster = urlopen(poster_link).read()
```

```

    #Defining the image file's save location in the current working
    directory
    save_location = os.getcwd() + '\\

    #Getting the file extension from the poster URL by grabbing the
    last value from the URL split by its dots
    #Usually images will end in '.JPEG' or '.PNG' so the last set of
    characters after the last dot should be the extension.
    poster_file_extension = poster_link.split('.')[ -1]

    #If the file extension is one of the most common image extensions,
    the file will be downloaded
    if poster_file_extension.upper() == 'JPEG' or
    poster_file_extension.upper() == 'JPG' or
    poster_file_extension.upper() == 'PNG':
        #Defining what the saved file will be named
        filename = '{}{}{}'.format(save_location, title, '.',
    poster_file_extension)

    #Opening a brand new file, writing the image data to it, then
    closing it
    with open(filename, 'wb') as f:
        f.write(poster)
        f.close()

## utility function for searching
def search_movie(title, serviceurl):
    try:
        url = serviceurl + urllib.parse.urlencode({'t': str(title)})
+apikey
        print(f'Retrieving the data of {title} now... ')
        print(url)
        uh = urllib.request.urlopen(url)
        data = uh.read()
        json_data=json.loads(data)
        if json_data['Response']=='True':
            print_json(json_data)
            # Asks user whether to download the poster of the movie

            if 'Poster' in json_data and json_data['Poster']!='N/A':
                download_poster(json_data)
            else:
                print('Error encountered: ', json_data['Error'])

        except urllib.error.URLError as e:
            print(f'ERROR: {e.reason}')

def show_image(movie_name):
    try:

```

```
from IPython.display import Image, display
display(Image(filename = '{}.jpg'.format(movie_name)))
except:
    print('ERROR!')
```

#Testing the functions on the movie Titanic

*#Testing the functions on the movie Titanic*

search\_movie('Titanic', omdbBaseURL)

show\_image('Titanic')

print(' ')

Retrieving the data of Titanic now...

<http://www.omdbapi.com/?t=Titanic&apikey=a8298fb6>

Title : Titanic

Year : 1997

Rated : PG-13

Released : 19 Dec 1997

Runtime : 194 min

Genre : Drama, Romance

Director : James Cameron

Writer : James Cameron

Actors : Leonardo DiCaprio, Kate Winslet, Billy Zane

Plot : A seventeen-year-old aristocrat falls in love with a kind but poor artist aboard the luxurious, ill-fated R.M.S. Titanic.

Language : English, Swedish, Italian, French

Country : United States, Mexico

Awards : Won 11 Oscars. 126 wins & 83 nominations total

Ratings : [{'Source': 'Internet Movie Database', 'Value': '7.9/10'}, {'Source': 'Rotten Tomatoes', 'Value': '88%'}, {'Source': 'Metacritic', 'Value': '75/100'}]

Metascore : 75

imdbRating : 7.9

imdbVotes : 1,317,259

imdbID : tt0120338

Type : movie

DVD : N/A

BoxOffice : \$674,292,608

Production : N/A

Website : N/A

Response : True

A movie poster for the film "Titanic". The top half features a close-up of Leonardo DiCaprio and Kate Winslet looking down at each other. Below them is a large, low-angle shot of the bow of the RMS Titanic against a cloudy sky. The title "TITANIC" is prominently displayed in large, white, serif letters across the middle of the ship's hull. Above the title, the names "LEONARDO DICAPRIO KATE WINSLET" are written in smaller white capital letters. At the bottom, there is a dense block of small text listing cast members, production credits, and logos for Fox Searchlight Pictures and Twentieth Century Fox. A quote from Entertainment Weekly is visible at the very bottom: "Nothing on Earth could come between them."

A movie poster for the film "Titanic". The top half features a close-up of Leonardo DiCaprio and Kate Winslet looking down at each other. Below them is a large, low-angle shot of the bow of the RMS Titanic against a cloudy sky. The title "TITANIC" is prominently displayed in large, white, serif letters across the middle of the ship's hull. Above the title, the names "LEONARDO DICAPRIO KATE WINSLET" are written in smaller white capital letters. At the bottom, there is a dense block of small text listing cast members, production credits, and logos for Fox Searchlight Pictures and Twentieth Century Fox. A quote from Entertainment Weekly is visible at the very bottom: "Nothing On Earth Could Come Between Them."

## A movie poster for the film "Titanic". The top half features a close-up of Leonardo DiCaprio and Kate Winslet looking down at each other. Below them is a large, low-angle shot of the bow of the RMS Titanic against a cloudy sky. The title "TITANIC" is prominently displayed in large, white, serif letters across the middle of the ship's hull. Above the title, the names "LEONARDO DICAPRIO KATE WINSLET" are written in smaller white capital letters. At the bottom, there is a dense block of small text listing cast members, production credits, and logos for Fox Searchlight Pictures and Twentieth Century Fox. A quote from Entertainment Weekly is visible at the very bottom: "Nothing on Earth could come between them."

A movie poster for the film "Titanic". The top half features a close-up of Leonardo DiCaprio and Kate Winslet looking down at each other. Below them is a large, low-angle shot of the bow of the RMS Titanic against a cloudy sky. The title "TITANIC" is prominently displayed in large, white, serif letters across the middle of the ship's hull. Above the title, the names "LEONARDO DICAPRIO KATE WINSLET" are written in smaller white capital letters. At the bottom, there is a dense block of small text listing cast members, production credits, and logos for Fox Searchlight Pictures and Twentieth Century Fox. A quote from Entertainment Weekly is visible at the very bottom: "Nothing on Earth could come between them."

A movie poster for the film "Titanic". The top half features a close-up of Leonardo DiCaprio and Kate Winslet looking down at each other. Below them is a large, low-angle shot of the bow of the RMS Titanic against a cloudy sky. The title "TITANIC" is prominently displayed in large, white, serif letters across the middle of the ship's hull. Above the title, the names "LEONARDO DICAPRIO KATE WINSLET" are written in smaller white capital letters. At the bottom, there is a dense block of small text listing cast members, production credits, and logos for Fox Searchlight Pictures and Twentieth Century Fox. A quote from Entertainment Weekly is visible at the very bottom: "Nothing on Earth could come between them."

Test the `search_movie` function by entering "Random\_error" (obviously, this will not be found, and you should be able to check

whether your error catching code is working properly).

```
# Test the search_movie function by entering "Random_error"  
(obviously, this will not be found, and you should be able to check  
# whether your error catching code is working properly).
```

```
search_movie('Random_error', omdbBaseURL)
```

```
Retrieving the data of Random_error now...
```

```
http://www.omdbapi.com/?t=Random\_error&apikey=a8298fb6
```

```
Error encountered:  Movie not found!
```

### Connect to the Twitter API and do a simple data pull

- If you don't have a twitter account – create one at [twitter.com/signup](https://twitter.com/signup) (you can delete the account after this assignment)
- Sign in to [apps.twitter.com](https://apps.twitter.com)
- Click "Create New App"
- Give your app a name and description
- Agree to the developer agreement – you will want to make sure to indicate this is for a class project, and this step can take several days to get through, so don't wait until last minute to complete this portion of the assignment
- Create an access token
- You should receive a consumer key and a token
- Using either the instructions from the book on connecting to an API or for help look here
  - pull back data searching for "Bellevue University" and "Data Science" (or something else you are interested in)
  - How to Create a Twitter App and API Interface via Python. (Grogan, 2016)
  - Welcome Python-Twitter's Documentation! (The Python-Twitter Developers, 2016)



```

# Load Libraries

# Import the Twython class
from twython import Twython
import json
import pandas as pd
import numpy as np

# Load credentials from json file
# import the Twython class, instantiate an object of it, and create
our search query.

# We'll use only four arguments in the query: q, result_type, count
and lang respectively
# for the search keyword, type, count, and language of results

with open("./datasets/TwitterAPIKeys.json", "r") as file:
    creds = json.load(file)

# Instantiate an object
python_tweets = Twython(creds['CONSUMER_KEY'],
creds['CONSUMER_SECRET'])

# Create our query
query = {'q': 'learn python',
        'result_type': 'popular',
        'count': 10,
        'lang': 'en',
        }

import tweepy
auth = tweepy.OAuthHandler(creds['CONSUMER_KEY'],
creds['CONSUMER_SECRET'])
auth.set_access_token(creds['ACCESS_TOKEN'], creds['ACCESS_SECRET'])

api = tweepy.API(auth)

# Set up Tweepy API client
client = tweepy.Client(creds['BEARER'], creds['CONSUMER_KEY'],
creds['CONSUMER_SECRET'],
creds['ACCESS_TOKEN'], creds['ACCESS_SECRET'],
wait_on_rate_limit=True)

# searched_tweets = [tweepy.Paginator(client.search_recent_tweets,
#
# max_results=30).flatten(limit=10)]
searched_tweets = client.get_users_tweets("44196397")

client.search_recent_tweets(query=query)

```

```

-----
-----
Unauthorized                                Traceback (most recent call
last)
Cell In[26], line 13
      8 client = tweepy.Client(creds['BEARER'], creds['CONSUMER_KEY'],
creds['CONSUMER_SECRET'],
      9                      creds['ACCESS_TOKEN'],
creds['ACCESS_SECRET'], wait_on_rate_limit=True)
     11 # searched_tweets =
[tweepy.Paginator(client.search_recent_tweets,
     12 #
max_results=30).flatten(limit=10)]
--> 13 searched_tweets = client.get_users_tweets("44196397")
     15 client.search_recent_tweets(query=query)

File ~\gyan-python-workspace\jup-workspace\venv\Lib\site-packages\
tweepy\client.py:1583, in Client.get_users_tweets(self, id, user_auth,
**params)
     1483 def get_users_tweets(self, id, *, user_auth=False, **params):
     1484     """get_users_tweets( \
     1485         id, *, end_time=None, exclude=None, expansions=None, \
     1486         max_results=None, media_fields=None,
pagination_token=None, \
     (... )
     1581     .. _here: https://developer.twitter.com/en/docs/twitter-
ids
     1582     """
-> 1583     return self._make_request(
     1584         "GET", f"/2/users/{id}/tweets", params=params,
     1585         endpoint_parameters=(
     1586             "end_time", "exclude", "expansions",
"max_results",
     1587             "media.fields", "pagination_token",
"place.fields",
     1588             "poll.fields", "since_id", "start_time",
"tweet.fields",
     1589             "until_id", "user.fields"
     1590         ), data_type=Tweet, user_auth=user_auth
     1591     )

File ~\gyan-python-workspace\jup-workspace\venv\Lib\site-packages\
tweepy\client.py:129, in BaseClient._make_request(self, method, route,
params, endpoint_parameters, json, data_type, user_auth)
     123 def _make_request(
     124     self, method, route, params={}, endpoint_parameters=(),
json=None,
     125     data_type=None, user_auth=False
     126 ):
     127     request_params = self._process_params(params,

```

```

endpoint_parameters)
--> 129     response = self.request(method, route,
params=request_params,
    130                                     json=json, user_auth=user_auth)
    132     if self.return_type is requests.Response:
    133         return response

```

```

File ~\gyan-python-workspace\jup-workspace\venv\Lib\site-packages\
tweepy\client.py:98, in BaseClient.request(self, method, route,
params, json, user_auth)

```

```

    96     raise BadRequest(response)
    97 if response.status_code == 401:
--> 98     raise Unauthorized(response)
    99 if response.status_code == 403:
    100     raise Forbidden(response)

```

```

Unauthorized: 401 Unauthorized
Unauthorized

```

```

for tweet in searched_tweets:
    print(client.search_recent_tweets(query=query))

```

```

-----
-----
Unauthorized                                Traceback (most recent call
last)

```

```

Cell In[23], line 2
      1 for tweet in searched_tweets:
----> 2     print(client.search_recent_tweets(query=query))

```

```

File ~\gyan-python-workspace\jup-workspace\venv\Lib\site-packages\
tweepy\client.py:1266, in Client.search_recent_tweets(self, query,
user_auth, **params)

```

```

    1174 """search_recent_tweets( \
    1175     query, *, end_time=None, expansions=None,
max_results=None, \
    1176     media_fields=None, next_token=None, place_fields=None, \
    (...)
    1263 .. _Academic Research Project:
https://developer.twitter.com/en/docs/projects
    1264 """
    1265 params["query"] = query
-> 1266 return self._make_request(
    1267     "GET", "/2/tweets/search/recent", params=params,
    1268     endpoint_parameters=(
    1269         "end_time", "expansions", "max_results",
"media.fields",
    1270         "next_token", "place.fields", "poll.fields", "query",
    1271         "since_id", "sort_order", "start_time",
"tweet.fields",

```

```

1272         "until_id", "user.fields"
1273     ), data_type=Tweet, user_auth=user_auth
1274 )

```

File ~\gyan-python-workspace\jup-workspace\venv\Lib\site-packages\tweepy\client.py:129, in BaseClient.\_make\_request(self, method, route, params, endpoint\_parameters, json, data\_type, user\_auth)

```

123 def _make_request(
124     self, method, route, params={}, endpoint_parameters=(),
json=None,
125     data_type=None, user_auth=False
126 ):
127     request_params = self._process_params(params,
endpoint_parameters)
--> 129     response = self.request(method, route,
params=request_params,
130                               json=json, user_auth=user_auth)
132     if self.return_type is requests.Response:
133         return response

```

File ~\gyan-python-workspace\jup-workspace\venv\Lib\site-packages\tweepy\client.py:98, in BaseClient.request(self, method, route, params, json, user\_auth)

```

96     raise BadRequest(response)
97 if response.status_code == 401:
--> 98     raise Unauthorized(response)
99 if response.status_code == 403:
100     raise Forbidden(response)

```

Unauthorized: 401 Unauthorized  
Unauthorized

**Using one of the datasets provided in Weeks 7 & 8, or a dataset of your own, choose 3 of the following visualizations to complete. You must submit via PDF along with your code. You are free to use Matplotlib, Seaborn or another package if you prefer.**

- Line
- Scatter
- Bar
- Histogram
- Density Plot
- Pie Chart

```

# I am using Iris dataset to demonstrate Python data visualizations
# First, we'll import pandas, a data processing and CSV file I/O
library
import pandas as pd

# We'll also import seaborn, a Python graphing library

import warnings # current version of seaborn generates a bunch of
warnings that we'll ignore
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", color_codes=True)

# Next, we'll load the Iris flower dataset, which is in the
"./input/" directory
iris = pd.read_csv("./datasets/iris.csv") # the iris dataset is now a
Pandas DataFrame

# Let's see what's in the iris data - Jupyter notebooks print the
result of the last thing you do
iris.head()

```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```

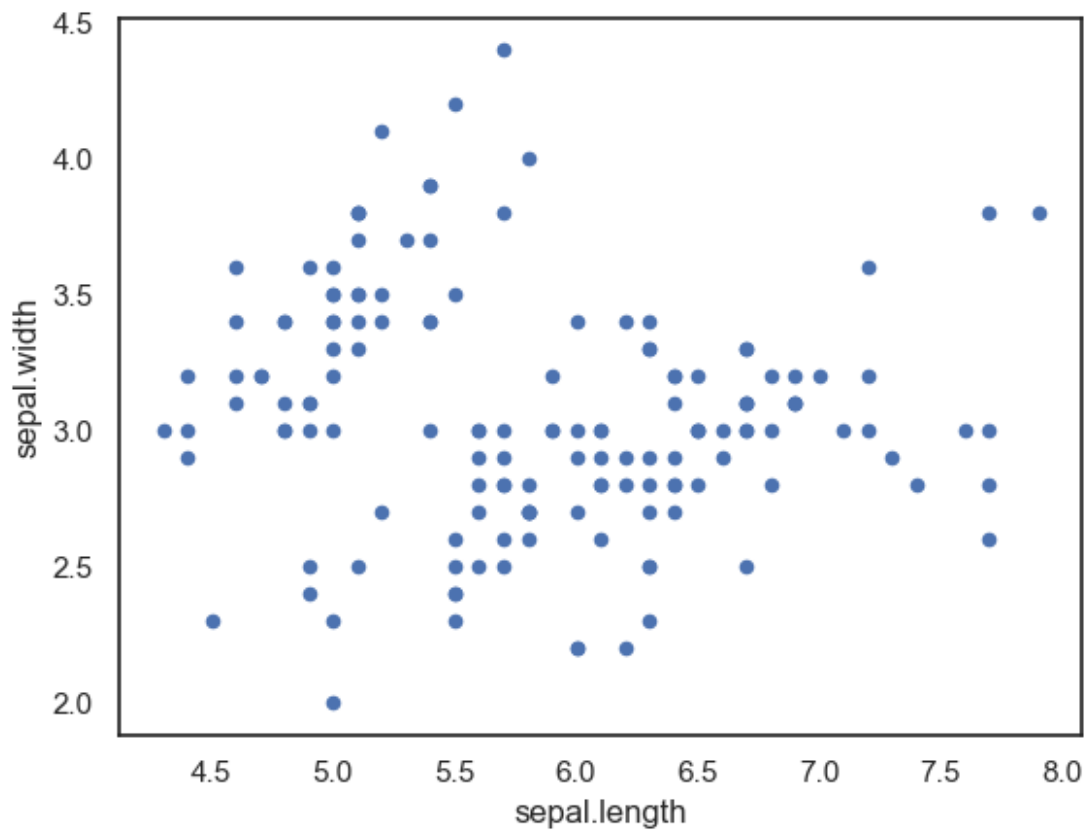
# Let's see how many examples we have of each species
iris["variety"].value_counts()

variety
Setosa      50
Versicolor  50
Virginica   50
Name: count, dtype: int64

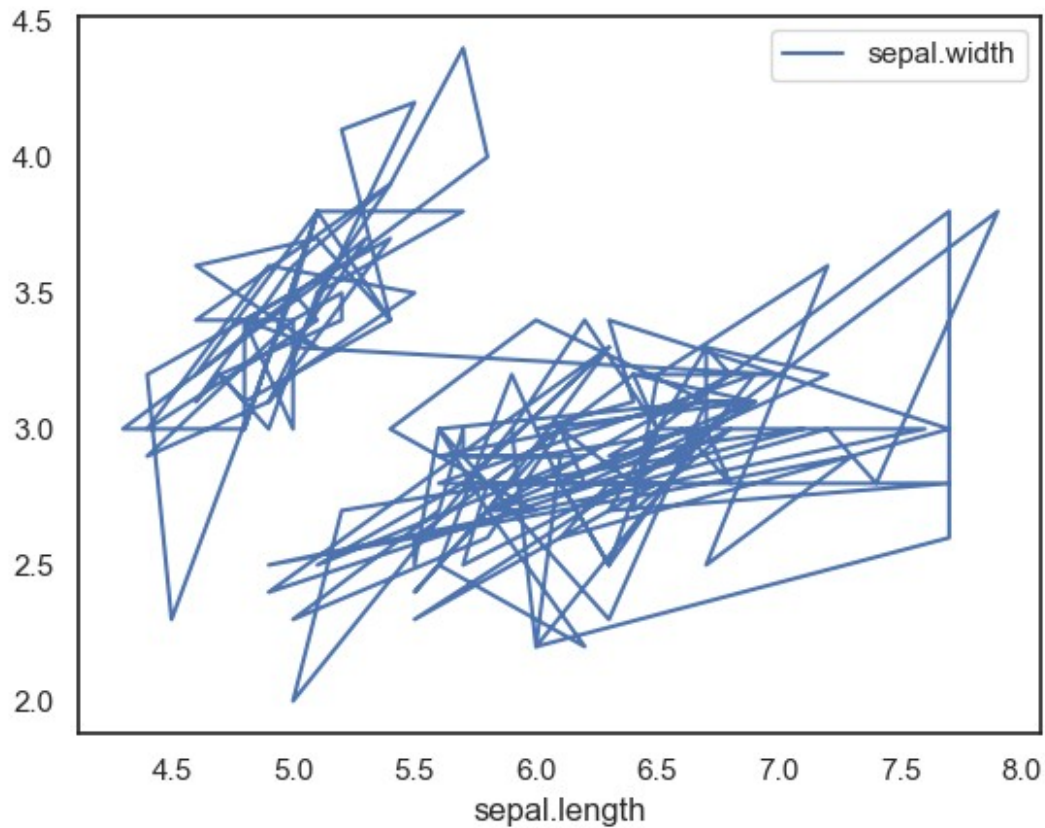
# The first way we can plot things is using the .plot extension from
Pandas dataframes
# We'll use this to make a scatterplot of the Iris features.
iris.plot(kind="scatter", x="sepal.length", y="sepal.width");

<Axes: xlabel='sepal.length', ylabel='sepal.width'>

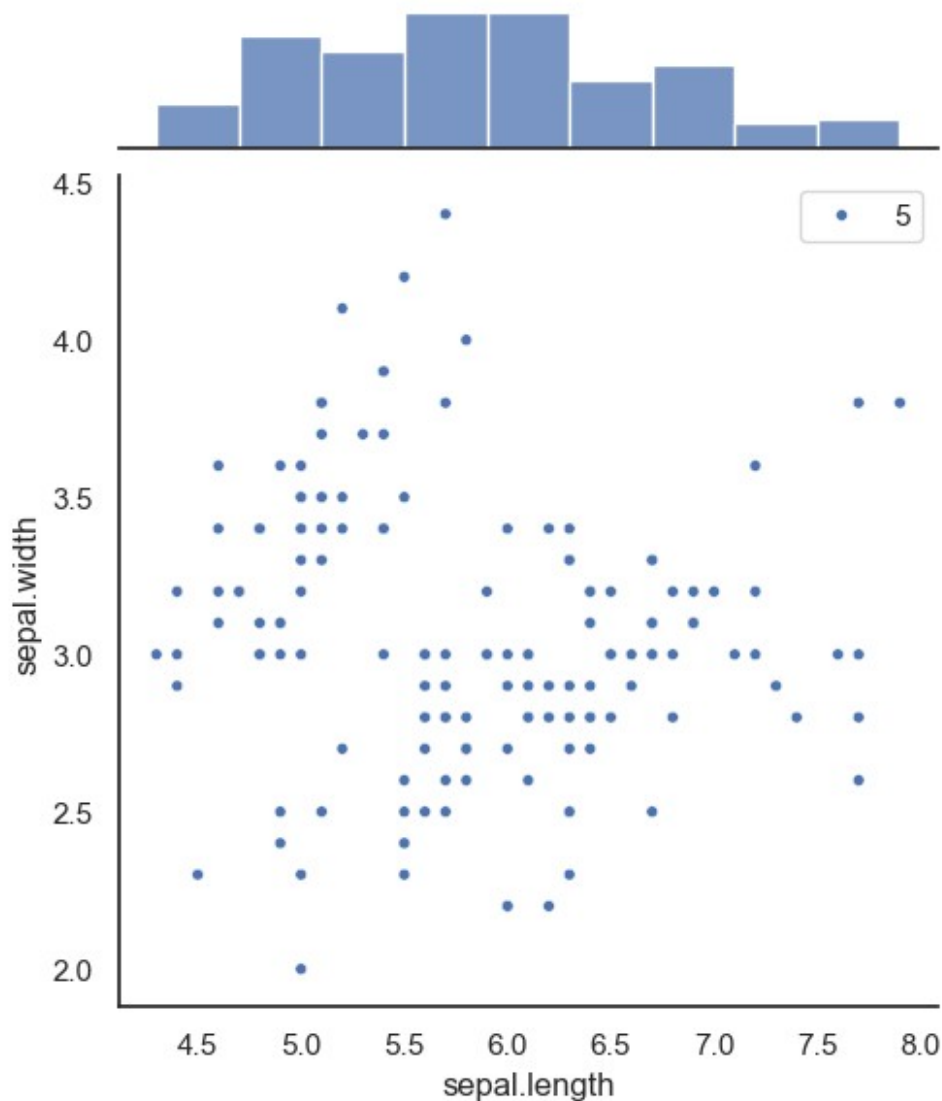
```



```
iris.plot(kind="line", x="sepal.length", y="sepal.width");  
<Axes: xlabel='sepal.length'>
```



```
# We can also use the seaborn library to make a similar plot  
# A seaborn jointplot shows bivariate scatterplots and univariate  
histograms in the same figure  
sns.jointplot(x="sepal.length", y="sepal.width", data=iris, size=5);
```



*# One piece of information missing in the plots above is what species each plant is*

*# We'll use seaborn's FacetGrid to color the scatterplot by species*

```
sns.FacetGrid(iris, hue="variety", size=5) \
.map(plt.scatter, "sepal.length", "sepal.width") \
.add_legend()
```

-----  
 -----  
 TypeError Traceback (most recent call last)

Cell In[35], line 4

```
1 # One piece of information missing in the plots above is what
species each plant is
```

```
2 # We'll use seaborn's FacetGrid to color the scatterplot by
```

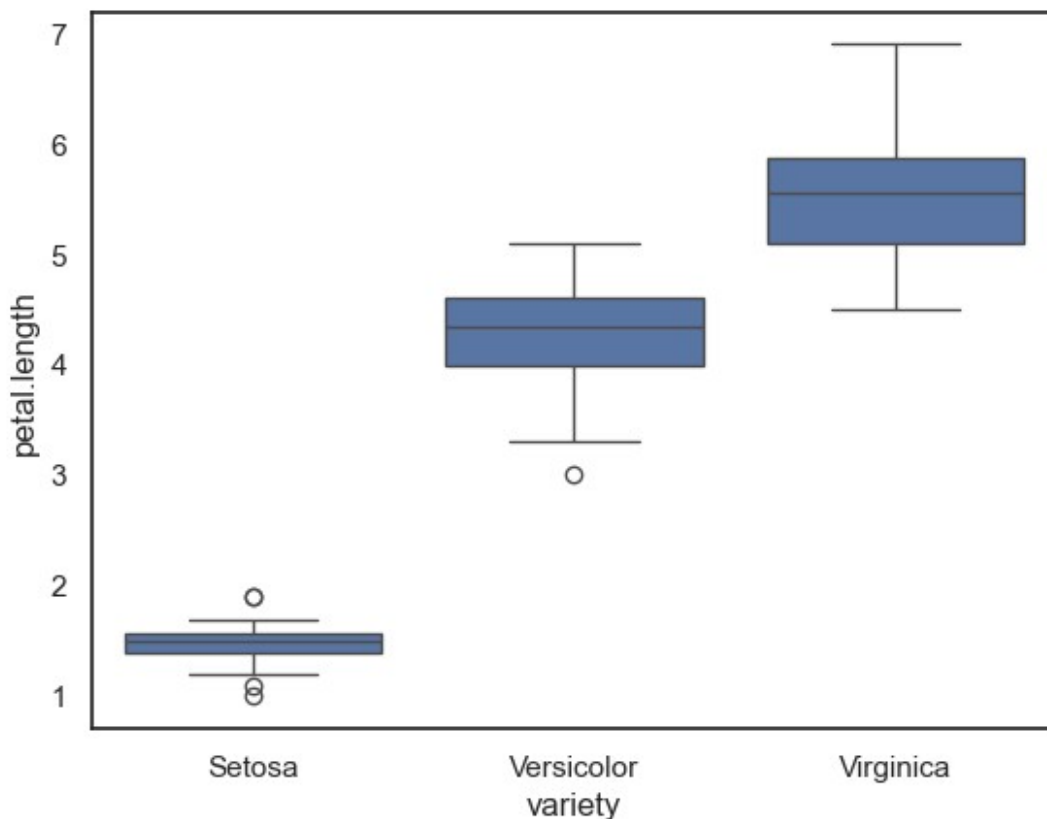


```
species
----> 4 sns.FacetGrid(iris, hue="variety", size=5) \
      5 .map(plt.scatter, "sepal.length", "sepal.width") \
      6 .add_legend()

TypeError: FacetGrid.__init__() got an unexpected keyword argument
'size'

# We can look at an individual feature in Seaborn through a boxplot
sns.boxplot(x="variety", y="petal.length", data=iris)

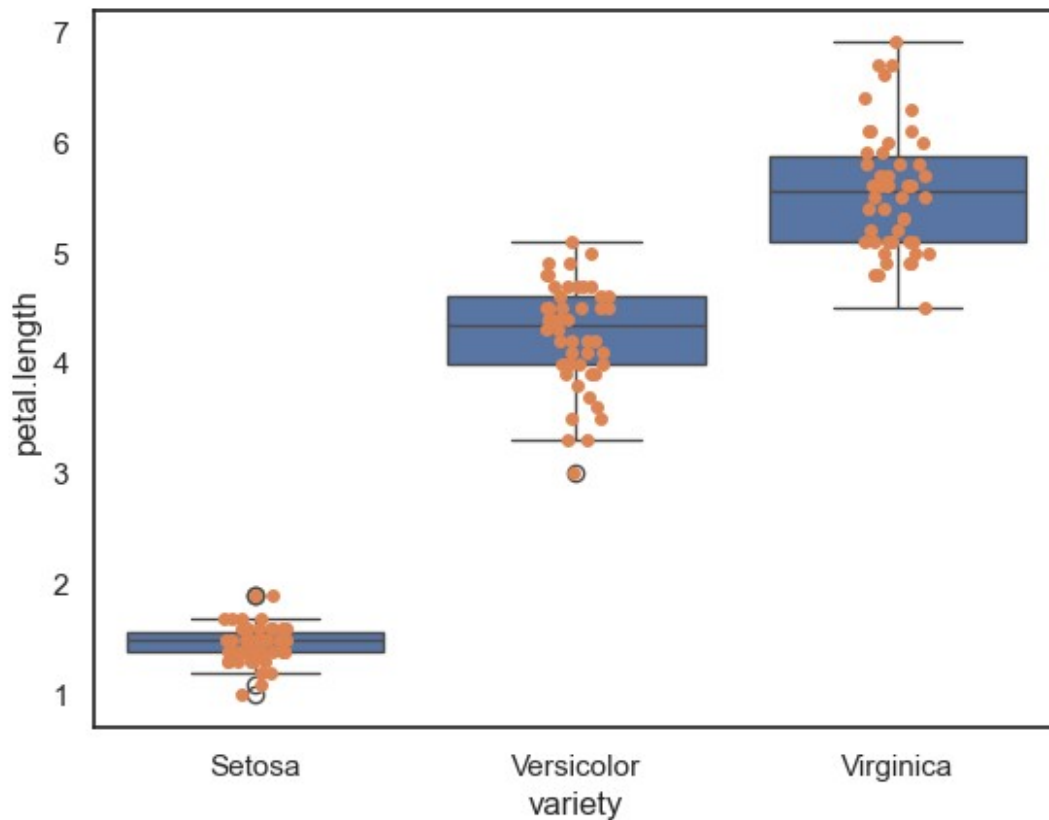
<Axes: xlabel='variety', ylabel='petal.length'>
```



```
# One way we can extend this plot is adding a layer of individual
points on top
# it through Seaborn's stripplot
# # We'll use jitter=True so that all the points don't fall in single
vertical lines
# above the species
# # Saving the resulting axes as ax each time causes the resulting
plot to be shown
# on top of the previous axes

ax = sns.boxplot(x="variety", y="petal.length", data=iris)
```

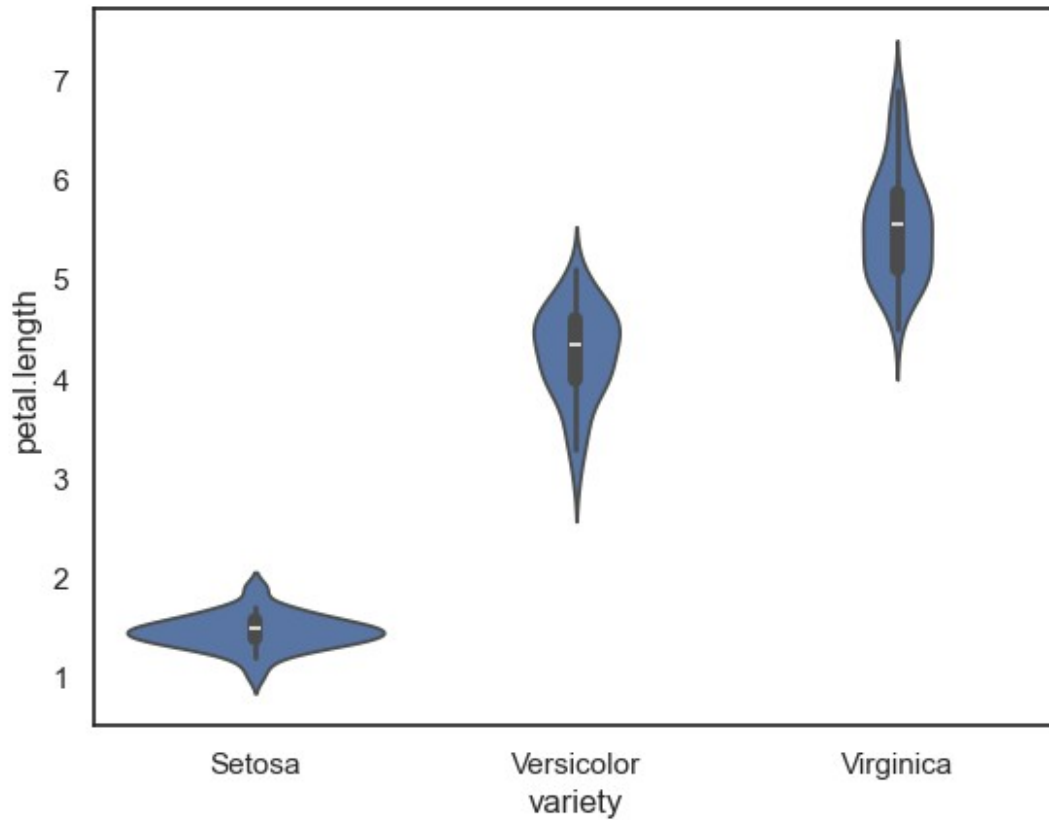
```
ax = sns.stripplot(x="variety", y="petal.length", data=iris,  
jitter=True,edgecolor="gray")
```



*# A violin plot combines the benefits of the previous two plots and simplifies them  
# Denser regions of the data are fatter, and sparser thinner in a violin plot.*

```
sns.violinplot(x="variety", y="petal.length", data=iris)
```

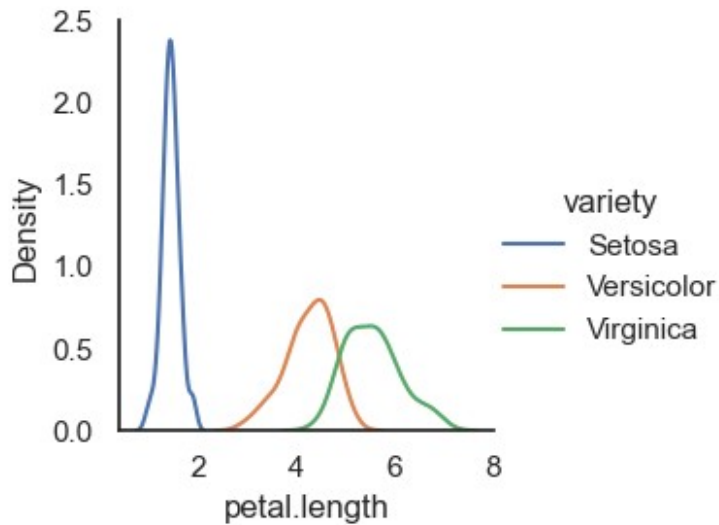
```
<Axes: xlabel='variety', ylabel='petal.length'>
```



*# A final seaborn plot useful for looking at univariate relations is the kdeplot,  
# which creates and visualizes a kernel density estimate of the underlying feature*

```
sns.FacetGrid(iris, hue="variety").map(sns.kdeplot,  
"petal.length").add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x16647a02a20>
```



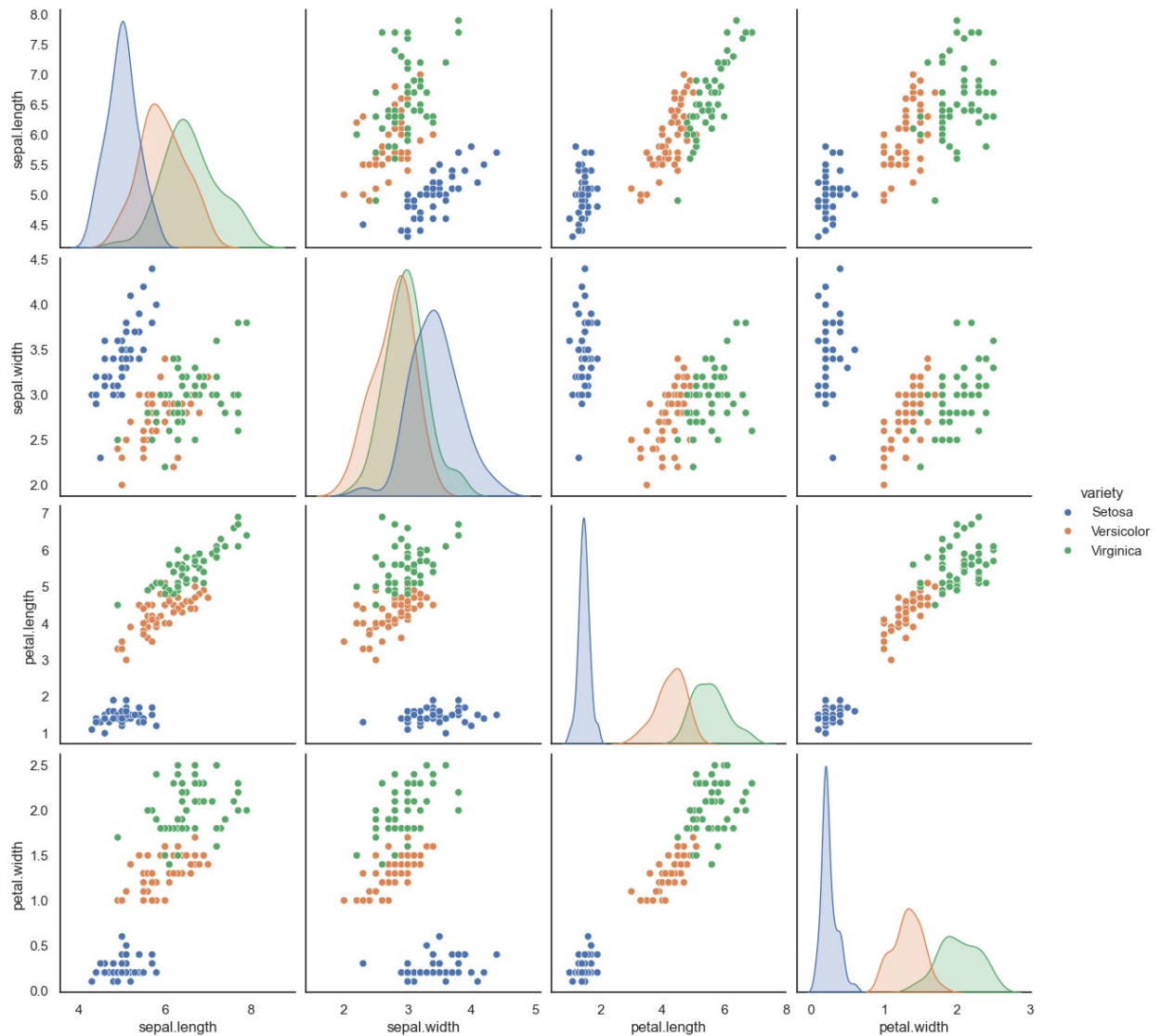
```
iris.reset_index().head()
```

	index	sepal.length	sepal.width	petal.length	petal.width	variety
0	0	5.1	3.5	1.4	0.2	Setosa
1	1	4.9	3.0	1.4	0.2	Setosa
2	2	4.7	3.2	1.3	0.2	Setosa
3	3	4.6	3.1	1.5	0.2	Setosa
4	4	5.0	3.6	1.4	0.2	Setosa

```
# Another useful seaborn plot is the pairplot, which shows the
# bivariate relation
# between each pair of features
# # From the pairplot, we'll see that the Iris-setosa species is
# separataed from the other
# two across all feature combinations
```

```
sns.pairplot(iris, hue="variety", size=3)
```

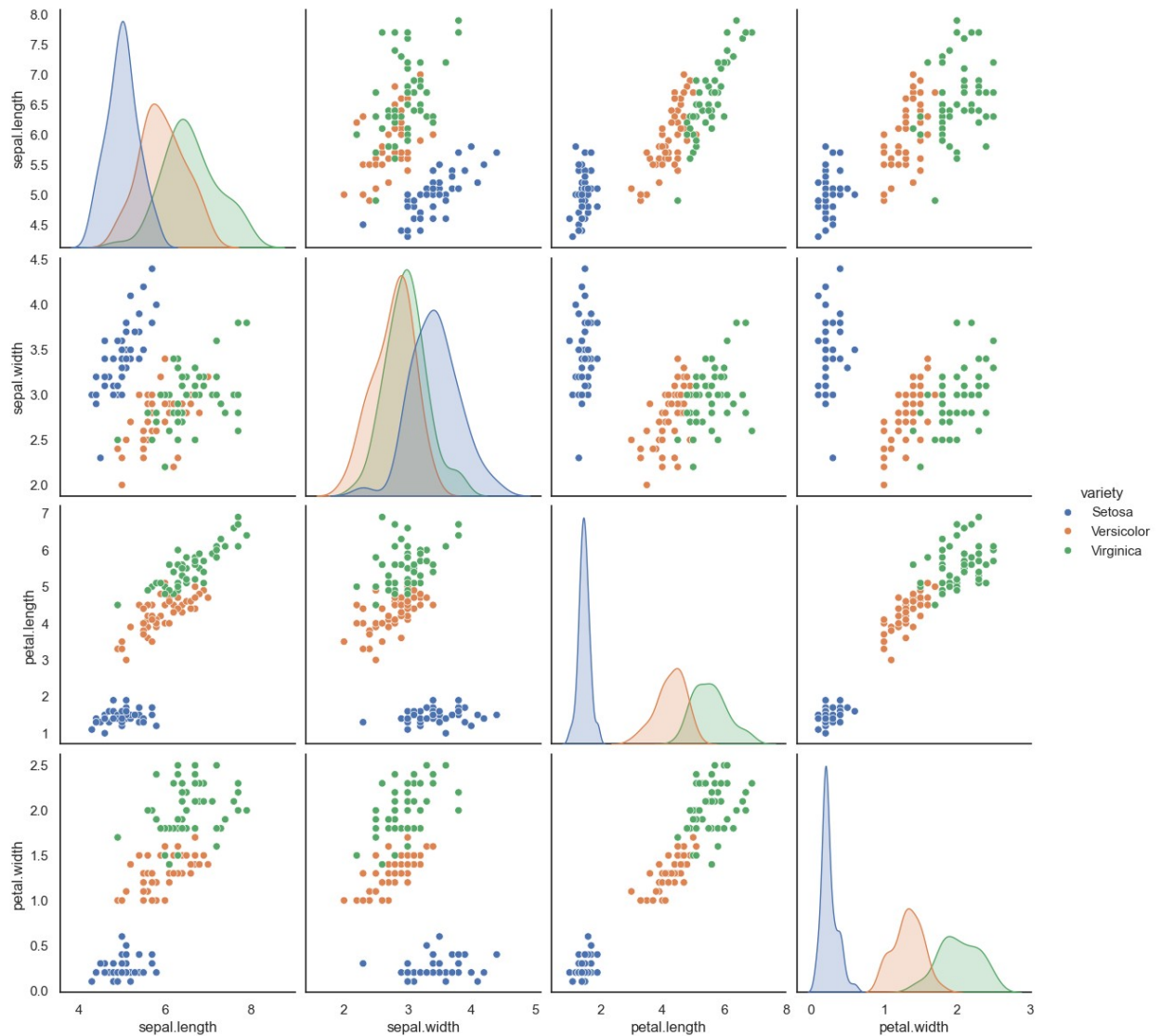
```
<seaborn.axisgrid.PairGrid at 0x166451b44a0>
```



# The diagonal elements in a pairplot show the histogram by default  
 # We can update these elements to show other things, such as a kde

```
sns.pairplot(iris, hue="variety", size=3, diag_kind="kde")
```

```
<seaborn.axisgrid.PairGrid at 0x1664a15cfe0>
```



*# Now that we've covered seaborn, let's go back to some of the ones we can make with Pandas*  
*# We can quickly make a boxplot with Pandas on each feature split out by species*

```
iris.boxplot(by="variety", figsize=(12, 6));
```

Boxplot grouped by variety

