AbhijitMandal_DSC540_Week9-10Ex

May 22, 2021

0.0.1 DSC 540 Week 9-10

Abhijit Mandal

0.0.2 Activity 9: Extracting the Top 100 eBooks from Gutenberg

Head over to the supplied Jupyter notebook (in the GitHub repository) to work on this activity. These are the steps that will help you solve this activity:

- Import the necessary libraries, including regex and beautifulsoup.
- Check the SSL certificate.
- Read the HTML from the URL. Write a small function to check the status of the web request.
- Decode the response and pass this on to BeautifulSoup for HTML parsing.
- Find all the href tags and store them in the list of links.
- Check what the list looks like – print the first 30 elements.
- 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.
- 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).
- 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).
- Create a starting index. It should point at the text Top 100 Ebooks yesterday.
- Use the splitlines method of soup.text. It splits the lines of text of the soup object. Loop 1-100 to add the strings of the next 100 lines to this temporary list. Hint: use the splitlines method. Use a regular expression to extract only text from the name strings and append it to an empty list.
- Use match and span to find the indices and use them.

0.0.3 Load the necessary libraries.

[161]:
```python
import urllib.request, urllib.parse, urllib.error
import requests
from bs4 import BeautifulSoup
import ssl
import re
```

[162]:
```python
#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
```

[163]:
```python
# 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 1
    else:
        #print("Failed!")
        return -1
```

[164]:
```python
# Read the HTML from the URL and check the response
gutenburgurl = 'https://www.gutenberg.org/browse/scores/top'
response = requests.get(gutenburgurl)
status_check(response)
```

[164]:
```
1
```

[165]:
```python
# Decode the response and pass on to BeautifulSoup for HTML parsing
urlContent = response.content.decode(response.encoding)
soup = BeautifulSoup(urlContent, 'html.parser')
```

[166]:
```python
# 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]
```

[166]:
```
['/',
 '/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/',
 '#books-last1',
 '#authors-last1',
 '#books-last7']
```

[167]:
```
#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
# Number 19 to 119 in the original list of links have the Top 100 ebooks'␣
↪number.
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 filenumber 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, 84, 1342, 64317, 11, 98, 2701, 844, 1661, 174, 46, 2542,
74, 5740, 76, 43, 1260, 1250, 2852, 5200, 1952, 1080, 65395, 345, 2591, 65392,
65394, 25344, 1400, 55, 16, 205, 65393, 219, 65398, 35899, 2600, 120, 215,
58585, 57775, 158, 514, 1232, 4300, 730, 6130, 1727, 768, 2500, 36, 236, 45,
1184, 3825, 902, 2554, 43453, 27827, 203, 829, 244, 996, 1497, 135, 30254,
65401, 5739, 26184, 16328, 20228, 160, 408, 2814, 103, 3600, 65384, 1399, 19033,
65399, 766, 19942, 100, 1998, 113, 23, 65396]
```

[168]:
```
# 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
```

Donation

Frequently Viewed or Downloaded
These listings are based on the number of times each eBook gets downloaded.
Multiple downloads from the same Internet address on the same day count as one download, and addresses that download more than 100 eBooks in a day are considered robots and are not counted.

Downloaded Books
2021-05-21149192
last 7 days1076641
last 30 days4858592

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

Frankenstein; Or, The Modern Prometheus by Mary Wollstonecraft Shelley (1791)
Pride and Prejudice by Jane Austen (1465)
The Great Gatsby by F. Scott Fitzgerald (900)
Alice's Adventures in Wonderland by Lewis Carroll (835)
A Tale of Two Cities by Charles Dickens (783)
Moby Dick; Or, The Whale by Herman Melville (619)
The Importance of Being Earnest: A Trivial Comedy for Serious People by Oscar Wilde (612)
The Adventures of Sherlock Holmes by Arthur Conan Doyle (579)
The Picture of Dorian Gray by Oscar Wilde (578)

```
A Christmas Carol in Prose; Being a Ghost Story of Christmas by Charles Dickens
(574)
Et dukkehjem. English by Henrik Ibsen (559)
The Adventures of Tom Sawyer, Complete by Mark Twain (534)
Tractatus Logico-Philosophicus by Ludwig Wittgenstein (502)
Adventures of Huckleberry Finn by Mark Twain (490)
The Strange
```

[169]:
```
# 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])
```

[170]:
```
# 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)
```

```
Frankenstein
Pride and Prejudice by Jane Austen
The Great Gatsby by F
Alice
A Tale of Two Cities by Charles Dickens
Moby Dick
The Importance of Being Earnest
The Adventures of Sherlock Holmes by Arthur Conan Doyle
The Picture of Dorian Gray by Oscar Wilde
A Christmas Carol in Prose
Et dukkehjem
The Adventures of Tom Sawyer
```

Tractatus Logico
Adventures of Huckleberry Finn by Mark Twain
The Strange Case of Dr
Jane Eyre
Anthem by Ayn Rand
The Hound of the Baskervilles by Arthur Conan Doyle
Metamorphosis by Franz Kafka
The Yellow Wallpaper by Charlotte Perkins Gilman
A Modest Proposal by Jonathan Swift
You Can
Dracula by Bram Stoker
Grimms
Roman Public Life by A
Battle Out of Time by Dwight V
The Scarlet Letter by Nathaniel Hawthorne
Great Expectations by Charles Dickens
The Wonderful Wizard of Oz by L
Peter Pan by J
Walden
The Phantom Regiment by James Grant
Heart of Darkness by Joseph Conrad
The Calumet Book of Oven Triumphs
The Philippines a Century Hence by Jos
War and Peace by graf Leo Tolstoy
Treasure Island by Robert Louis Stevenson
The Call of the Wild by Jack London
The Prophet by Kahlil Gibran
Le jardin des supplices by Octave Mirbeau
Emma by Jane Austen
Little Women by Louisa May Alcott
Il Principe
Ulysses by James Joyce
Oliver Twist by Charles Dickens
The Iliad by Homer
The Odyssey by Homer
Wuthering Heights by Emily Bront
Siddhartha by Hermann Hesse
The War of the Worlds by H
The Jungle Book by Rudyard Kipling
Anne of Green Gables by L
The Count of Monte Cristo
Pygmalion by Bernard Shaw
The Happy Prince
Prestuplenie i nakazanie
A Pickle for the Knowing Ones by Timothy Dexter
The Kama Sutra of Vatsyayana by Vatsyayana
Uncle Tom
Gulliver

```
A Study in Scarlet by Arthur Conan Doyle
Don Quixote by Miguel de Cervantes Saavedra
The Republic by Plato
Les Mis
The Romance of Lust
A Perfect Fool by Florence Warden
Korean
Simple Sabotage Field Manual by United States
Beowulf
Noli Me Tangere by Jos
The Awakening
The Souls of Black Folk by W
Dubliners by James Joyce
Around the World in Eighty Days by Jules Verne
Essays of Michel de Montaigne
And Five Were Foolish by Dornford Yates
Anna Karenina by graf Leo Tolstoy
Alice
Plank Frame Barn Construction by John L
David Copperfield by Charles Dickens
Candide by Voltaire
The Complete Works of William Shakespeare by William Shakespeare
Also sprach Zarathustra
The Secret Garden by Frances Hodgson Burnett
Narrative of the Life of Frederick Douglass
The Chronicles of Enguerrand de Monstrelet
Sense and Sensibility by Jane Austen
As Other Men Are by Dornford Yates
The Prince and the Pauper by Mark Twain
The Mysterious Affair at Styles by Agatha Christie
Beyond Good and Evil by Friedrich Wilhelm Nietzsche
The Time Machine by H
The Legend of Sleepy Hollow by Washington Irving
Chambers
Hans Andersen
The Devil
A Japanese Boy by Shigemi Shiukichi
Leviathan by Thomas Hobbes
The Life and Adventures of Robinson Crusoe by Daniel Defoe
```

0.0.4 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).

[171]:
```
# Loading Libraries
import urllib.request, urllib.parse, urllib.error
import json
from PIL import Image
import requests
import pandas as pd
```

[172]:
```
# read the api key file to get the key
# Opening JSON file
f = open('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])
```

[173]:
```
# Assing the portal value to variable
omdbBaseURL = "http://www.omdbapi.com/?"

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

[174]:
```
# 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 an show the image
            im = Image.open(requests.get(valData, stream=True).raw)
            im.show()
        else:
            print(keyData + " : " + valData + "\n")
```

[175]:
```
# method to get movie details

def search_movie(moviename, serviceurl, apikey):
    try:
        apiURl = serviceurl + "t=" + moviename + apikey

        # calling the api to get response
        with urllib.request.urlopen(apiURl) as url:
            data = json.loads(url.read().decode())

        # check the status of the response
        status = False
        for item in data.items():
            if (str(item[0]).lower() == 'response' and str(item[1]).lower() ==␣
↪'true'):
                status = True


        # checking the status of response
        if(status == False):
            print('Api returned failure response')
        else:
            # if response is successful then print the data
            print_json(data)

    except:
        # print an error message if connection not sucessful
        print("Error occured while processing your request")
```

[176]:
```
# Test the search_movie function by entering Titanic.
search_movie('Titanic', omdbBaseURL, apikey)
```

```
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, Kathy Bates

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 : USA, Mexico, Australia, Canada

Awards : Won 11 Oscars. Another 115 wins & 83 nominations.

Ratings : [{'Source': 'Internet Movie Database', 'Value': '7.8/10'}, {'Source':
'Rotten Tomatoes', 'Value': '89%'}, {'Source': 'Metacritic', 'Value': '75/100'}]

Metascore : 75

imdbRating : 7.8

imdbVotes : 1,067,012

imdbID : tt0120338
```

```
Type : movie

DVD : 01 Jun 2014

BoxOffice : $659,363,944

Production : 20th Century Fox, Lightstorm Entertainment, Paramount Pictures

Website : N/A

Response : True
```

[177]:
```python
# 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, apikey)
```

```
Api returned failure response
```

0.0.5 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 (you can delete the account after this assignment)
- Sign in to 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)

[178]:
```python
# Load Libraries

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

[179]:
```python
# 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("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',
        }
```

[180]:
```
# we can use our Twython object to call the search method,
# which returns a dictionary of search_metadata and statuses - the queried␣
 ↪results.
# We'll only look at the statuses part, and save a portion of all information␣
 ↪in a pandas dataframe,
# to present it in a table.


# Search tweets
dict_ = {'user': [], 'date': [], 'text': [], 'favorite_count': []}
for status in python_tweets.search(**query)['statuses']:
    dict_['user'].append(status['user']['screen_name'])
    dict_['date'].append(status['created_at'])
    dict_['text'].append(status['text'])
    dict_['favorite_count'].append(status['favorite_count'])

# Structure data in a pandas DataFrame for easier manipulation
df = pd.DataFrame(dict_)
df.sort_values(by='favorite_count', inplace=True, ascending=False)
df.head(5)
```

[180]:
```
       user                            date  \
3    ThePSF  Mon May 17 18:00:20 +0000 2021
4    ThePSF  Wed May 19 18:23:02 +0000 2021
1    ThePSF  Fri May 21 21:59:02 +0000 2021
2  nostarch  Mon May 17 18:01:51 +0000 2021
0     Azure  Fri May 21 16:00:01 +0000 2021
```

```
                                                text  favorite_count
3  The PSF is featured in a Humble Book Bundle al…             215
4  Proceeds from this month's Humble Book Bundle …             121
1  Humble Book Bundle is donating proceeds to the…             112
2  Our latest @Humble Bundle deal is live! Pay wh…             111
0  Learn how to:\n Use Python for ETL, data expl…              63
```

0.0.6 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

[181]:
```
# 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("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()
```

[181]:
```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

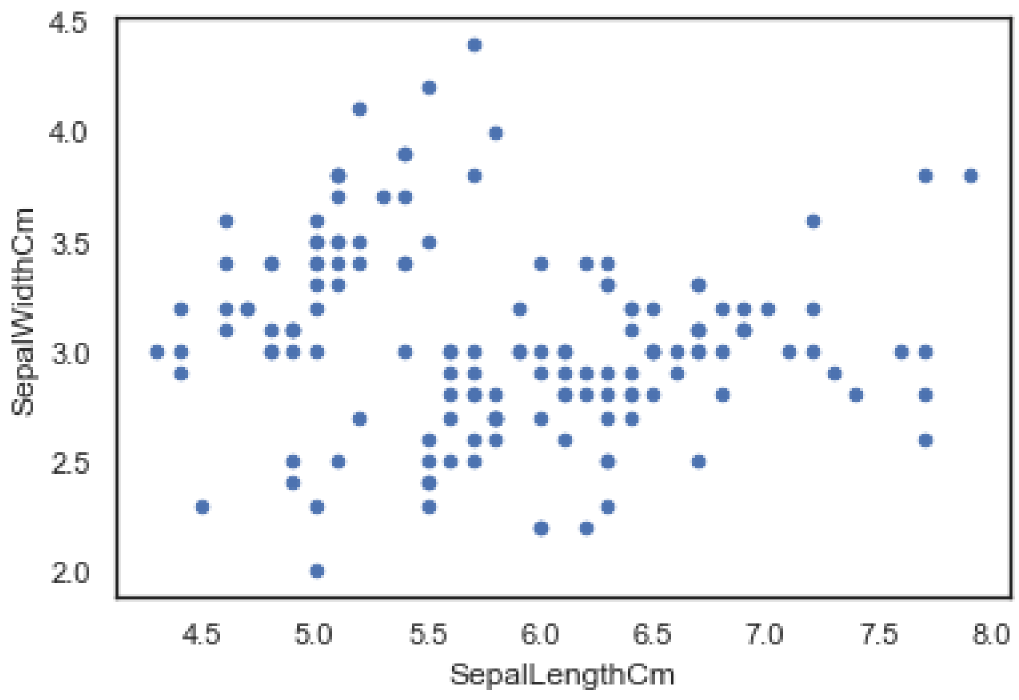
[182]:
```
# Let's see how many examples we have of each species
iris["Species"].value_counts()
```

[182]:
```
Iris-versicolor    50
Iris-setosa        50
Iris-virginica     50
Name: Species, dtype: int64
```

[183]:
```
# 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="SepalLengthCm", y="SepalWidthCm")
```

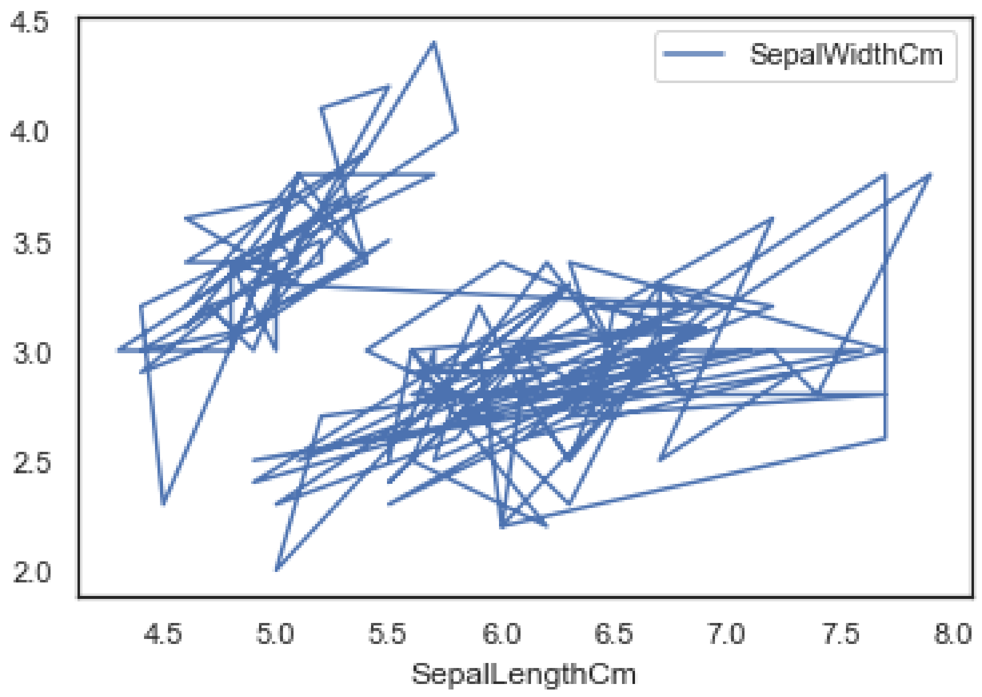
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```

[183]:
```
<AxesSubplot:xlabel='SepalLengthCm', ylabel='SepalWidthCm'>
```



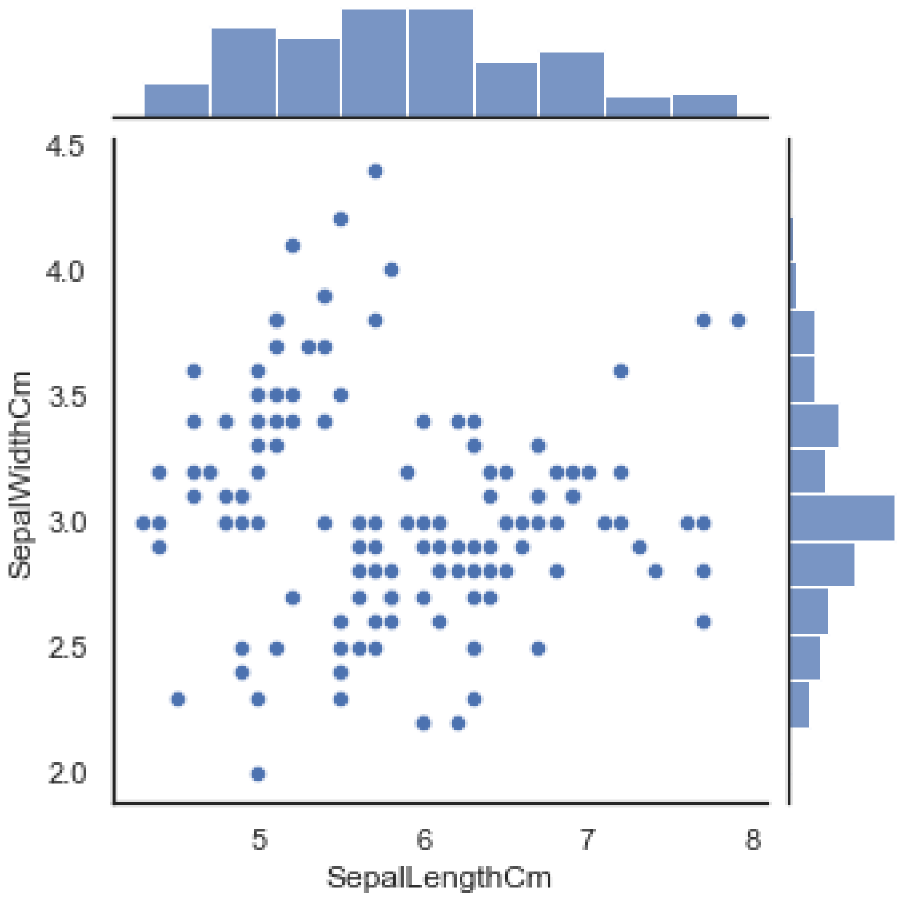
[184]:
```
iris.plot(kind="line", x="SepalLengthCm", y="SepalWidthCm")
```

[184]:
```
<AxesSubplot:xlabel='SepalLengthCm'>
```



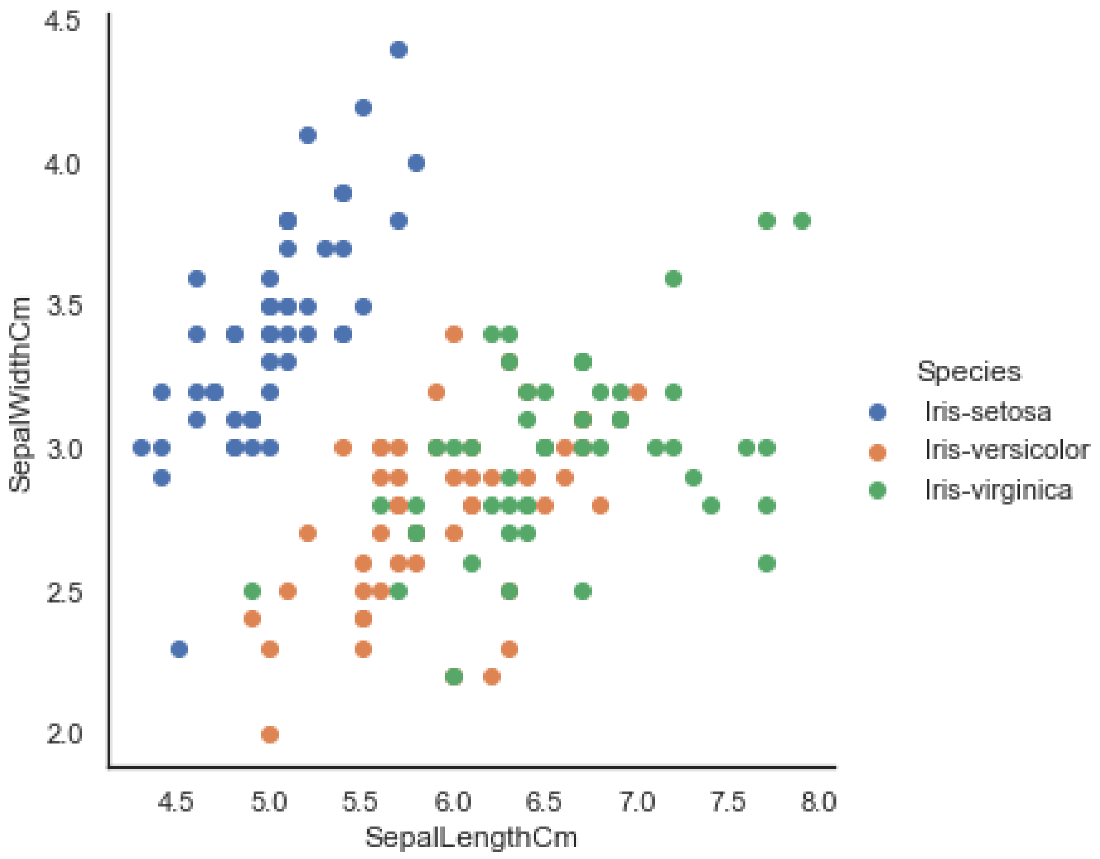
```
[185]: # We can also use the seaborn library to make a similar plot
       # A seaborn jointplot shows bivariate scatterplots and univariate histograms in␣
        ↪the same figure
       sns.jointplot(x="SepalLengthCm", y="SepalWidthCm", data=iris, size=5)
```

```
[185]: <seaborn.axisgrid.JointGrid at 0x7f9b685c52e0>
```



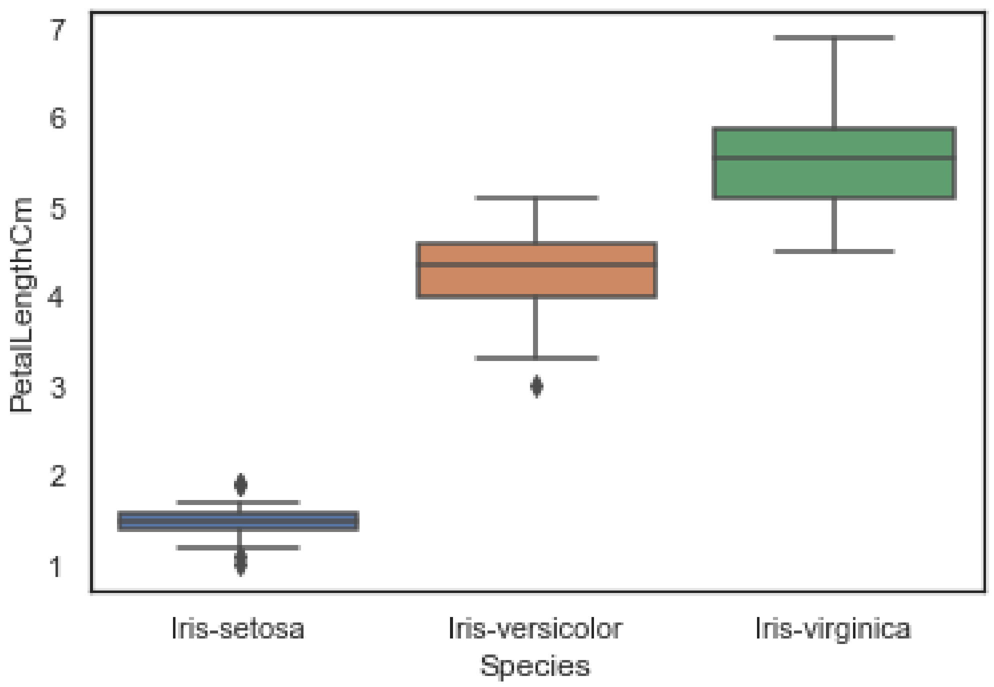
[186]:
```
# One piece of information missing in the plots above is what species each␣
 ↪plant is
# We'll use seaborn's FacetGrid to color the scatterplot by species
sns.FacetGrid(iris, hue="Species", size=5) \
   .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
   .add_legend()
```

[186]:
```
<seaborn.axisgrid.FacetGrid at 0x7f9b3a36e2e0>
```

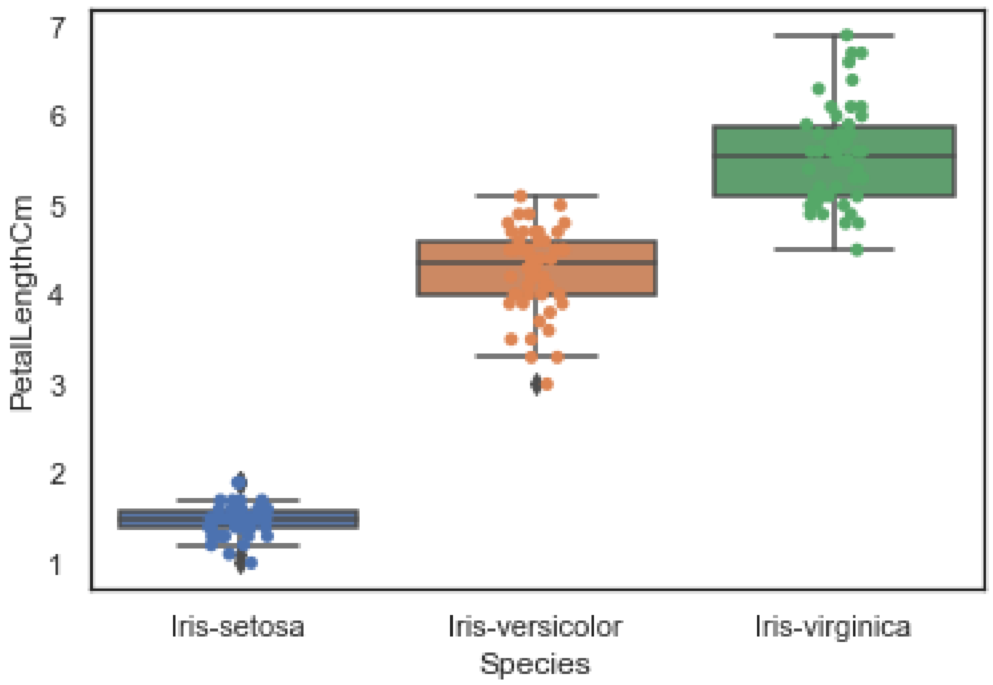


```
[187]: # We can look at an individual feature in Seaborn through a boxplot
       sns.boxplot(x="Species", y="PetalLengthCm", data=iris)
```

```
[187]: <AxesSubplot:xlabel='Species', ylabel='PetalLengthCm'>
```

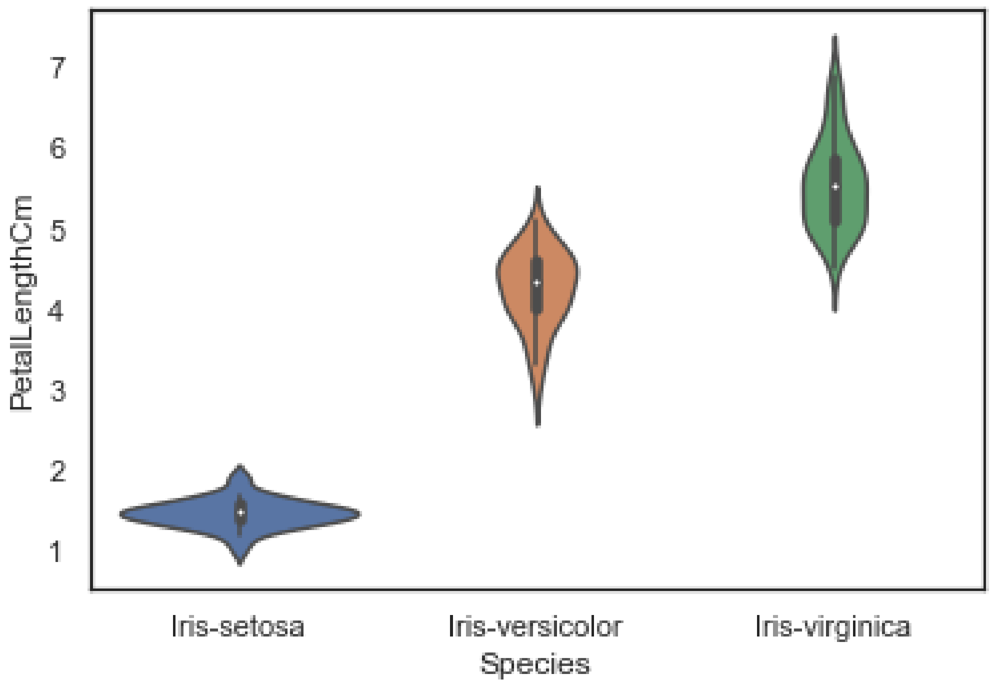


[188]:
```
# One way we can extend this plot is adding a layer of individual points on top␣
↪of
# it through Seaborn's striplot
#
# We'll use jitter=True so that all the points don't fall in single vertical␣
↪lines
# above the species
#
# Saving the resulting axes as ax each time causes the resulting plot to be␣
↪shown
# on top of the previous axes
ax = sns.boxplot(x="Species", y="PetalLengthCm", data=iris)
ax = sns.stripplot(x="Species", y="PetalLengthCm", data=iris, jitter=True,␣
↪edgecolor="gray")
```



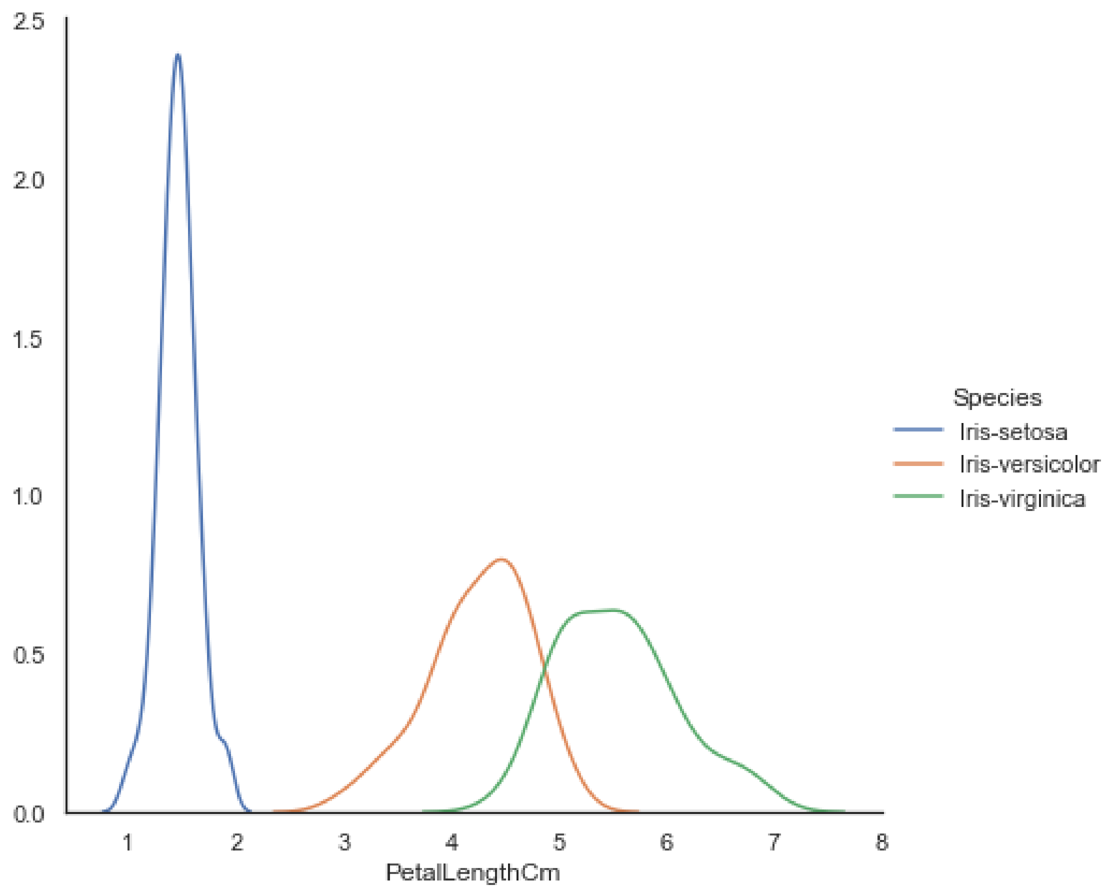
```
[189]: # A violin plot combines the benefits of the previous two plots and simplifies␣
        ↪them
       # Denser regions of the data are fatter, and sparser thiner in a violin plot
       sns.violinplot(x="Species", y="PetalLengthCm", data=iris, size=6)
```

```
[189]: <AxesSubplot:xlabel='Species', ylabel='PetalLengthCm'>
```



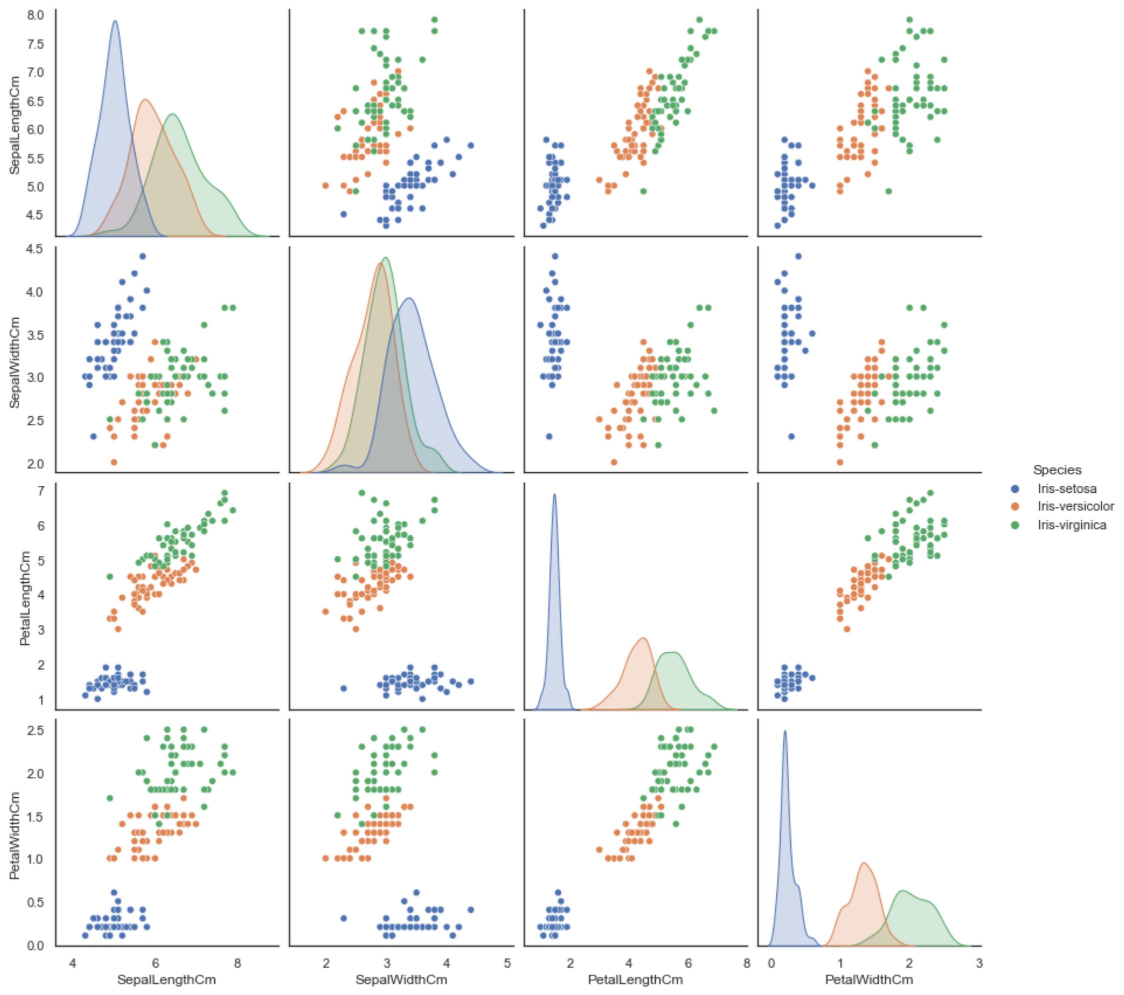
[190]:
```
# A final seaborn plot useful for looking at univariate relations is the␣
↪kdeplot,
# which creates and visualizes a kernel density estimate of the underlying␣
↪feature
sns.FacetGrid(iris, hue="Species", size=6) \
   .map(sns.kdeplot, "PetalLengthCm") \
   .add_legend()
```

[190]:
```
<seaborn.axisgrid.FacetGrid at 0x7f9b687cd070>
```



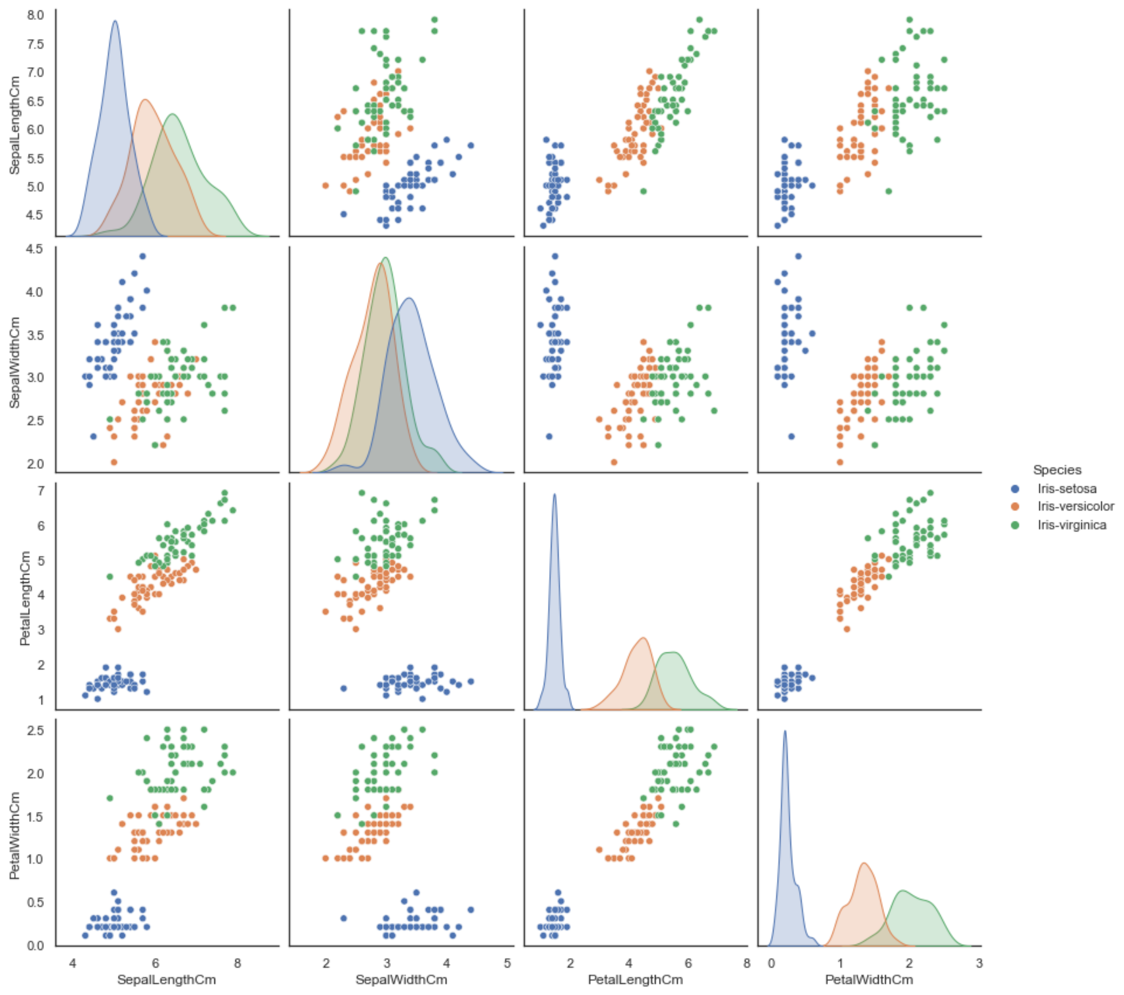
[191]:
```
# Another useful seaborn plot is the pairplot, which shows the bivariate␣
↪relation
# between each pair of features
#
# From the pairplot, we'll see that the Iris-setosa species is separataed from␣
↪the other
# two across all feature combinations
sns.pairplot(iris.drop("Id", axis=1), hue="Species", size=3)
```

[191]:
```
<seaborn.axisgrid.PairGrid at 0x7f9b3a3919d0>
```



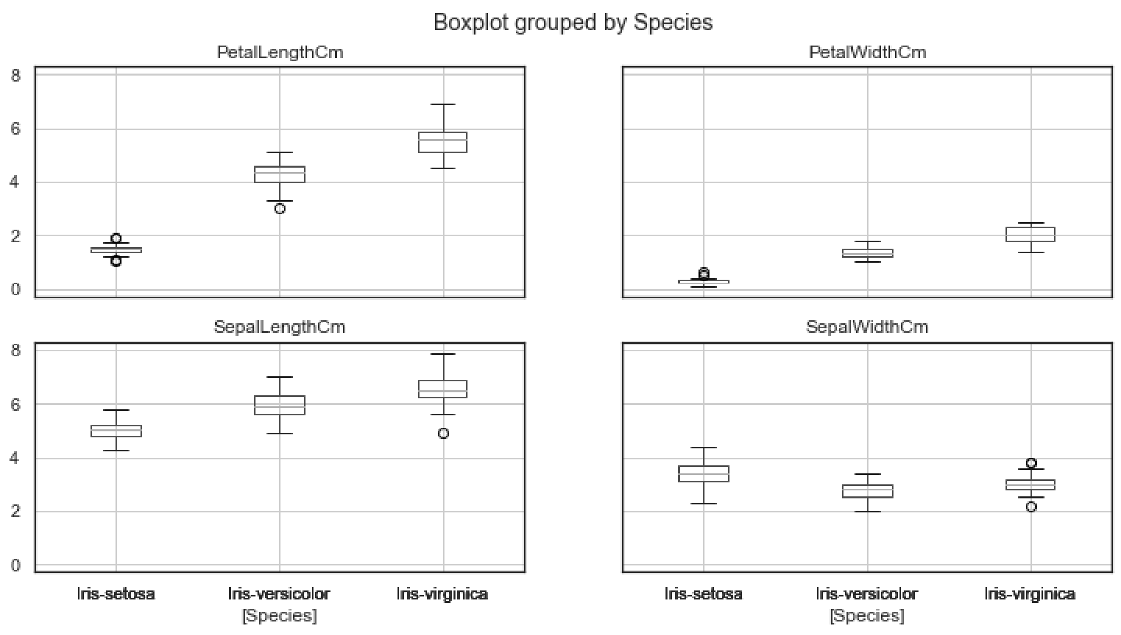
```
[192]: # The diagonal elements in a pairplot show the histogram by default
       # We can update these elements to show other things, such as a kde
       sns.pairplot(iris.drop("Id", axis=1), hue="Species", size=3, diag_kind="kde")
```

```
[192]: <seaborn.axisgrid.PairGrid at 0x7f9b78cd11f0>
```



```
[193]: # 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.drop("Id", axis=1).boxplot(by="Species", figsize=(12, 6))
```

```
[193]: array([[<AxesSubplot:title={'center':'PetalLengthCm'}, xlabel='[Species]'>,
               <AxesSubplot:title={'center':'PetalWidthCm'}, xlabel='[Species]'>],
              [<AxesSubplot:title={'center':'SepalLengthCm'}, xlabel='[Species]'>,
               <AxesSubplot:title={'center':'SepalWidthCm'}, xlabel='[Species]'>]],
             dtype=object)
```


Boxplot grouped by Species
PetalLengthCm
PetalWidthCm
SepalLengthCm
SepalWidthCm
0
2
4
6
8
Iris-setosa
Iris-versicolor
Iris-virginica
[Species]