Web Scraping

- Extract required data (based on problem statement) from any website and store that into structured format like .csv.

life cycle of Datascience:

1) problem statement
2) Data collection based on problem statement
3) Data cleaning
4) Data Analysis
5) Data Visualization
6) Feature Engineering
7) model building
8) model evaluation
9) model Deployment

webScraping Project will cover all this.

Required libraries:

1) requests
2) BeautifulSoup

Requests module

* The requests module allows you to send HTTP requests using Python.
* The HTTP request returns a Response object with all response data (content, encoding, status etc)

Install requests module

```
pip install requests
import requests
```

Syntax

```
requests.methodname(parameters)
```

method	Description
delete(url, args)	Sends a DELETE request to the specified url
get(url, parms, args)	Sends a GET request to the specified url
head(url, args)	Sends a HEAD request to the specified url
patch(url, data, args)	Sends a PATCH request to the specified url
post(url, data, json, args)	Sends a post request to the specified url
put(url, data, args)	Sends a PUT request to the specified url

request(method, url, args)	Sends a request of the specified method to the specified url.

BeautifulSoup module

* BeautifulSoup is a python library that is used for webscraping purposes to pull the data out of HTML and XML files. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner.

Install BeautifulSoup

```
pip install beautifulsoup4
from bs4 import BeautifulSoup
```

HTML

* HyperText Markup Language (HTML) is the language that the webpages are created in.
* HTML isn't a programming language, like Python, though.
* It's a Markup language that tells a browser how to display content.
* HTML has many functions that are similar to what you might find in a word processor like microsoft word - it can make text bold, create paragraphs, and so on.
* HTML is not case sensitive.
* HTML is a collection of tags. (words in between < > brackets)
* most basic tag is <html>. This tag tells the web browser that everything inside of it is HTML.

Simple HTML document

```
<html>
</html>
```

* we haven't added any content to our page yet, so if we viewed our HTML document in a web browser, we wouldn't see anything.
** once a tag is opened in HTML we have to close it.

* Right inside an html tag, we can put two other tags: the head tag and the body tag.
* The main content of the webpage goes in to body tag.
* The head tag contains data about the title of the page, and other information that generally isn't useful in web scraping.

```
<html>
<head>
</head>
<body>
</body>
</html>
```

* we still haven't added any content to our page (that goes inside the body tag), so if we open this HTML file in browser, we still won't see anything.
* The head and body tags were placed inside the <html> tag. In HTML tags are nested and can go inside other tags.
* we'll now add our first content to the page, inside a P tag. The P tag defines a paragraph, and any text inside the tag is shown as a seperate paragraph:

```
<html>
<head>
</head>
<body>
<p>
Here's a paragraph of text!
</p>
<p>
Here's a second paragraph of text!
</p>
</body>
</html>
```

* Rendered in a browser, that HTML file will look like this:

Here's a paragraph of text!

Here's a second paragraph of text!

* Tags have commonly used names that depend on their position in relation to other tags:
  - child - a child is a tag inside another tag. So the two p tags are both children of the body tag.
  - Parent - a parent is the tag where another tag is inside. Above, the html tag is the parent of the body tag.

- Sibling - a sibling is a tag that is nested inside the same parent as another tag, For example, head and body are siblings, since they're both inside html. Both p tags are siblings, since they're both inside the body.

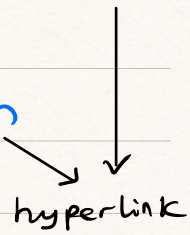
* we can also add properties to HTML tags that change their behaviour. Below, we'll add some extra text and hyperlinks using the a tag.

```
<html>
<head>
</head>
<body>
<p>
Here's a paragraph of text!
<a href="https:....">Learn Data science online</a>
</p>
<p>
Here's a second paragraph of text!
<a href="https:....">Python
</a>
</p>
</body>
</html>
```

* Here's how it will look in browser

Here's a paragraph of text! Learn Datascience online

Here's a second paragraph of text! Python



* In the above example, we added two a tags. a tags are links, and tell the browser to render a link to another webpage.
* The href property of the tag determines where the link goes.
* a and p are extremely common HTML tags. Here are a few others:
  - div - indicates a division, or area, of the page.
  - b - bolds any text inside
  - i - italicizes any text inside
  - table - Creates a table
  - form - creates an input form
* Before we move in to actual webscraping, let's learn about the class and id properties. These special properties give HTML element names, and make them easier to interact with when we're scraping.
* One element can have multiple classes, and a class can be shared between elements.

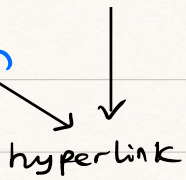
* Each element can only have one id, and an id can only be used once on a page.
* classes and ids are optional, and not all elements will have them.
* we can add classes and ids to our example

```
<html>
<head>
</head>
<body>
<p class="bold-paragraph">
Here's a paragraph of text!
<a href="https:.." id="learn-link">Learn Datascience online
</a>
</p>
<p class="bold-paragraph extra-large"
Here's a second paragraph of text!
<a href="https:.." class="extra-large">Python
</a>
</p>
</body>
</html>
```

* Here's how it will look in browser

Here's a paragraph of text! Learn Datascience online

Here's a second paragraph of text! Python



* As we can see, adding classes and ids doesn't change how the tags are rendered at all.
* HTML code can be written in a Notepad and saved with .html extension.
* we can open in default browser by double clicking the .html file.
* Inside the .html file if we right click and select inspect we can see the html code.
* Code inside the inside can be edited to reflect changes but it is only temporary until the browser is refreshed.

The fundamentals of webscraping

what is webscraping in python?

* some websites offer data sets that are downloadable in csv format, or accessible via an Application-Programming interface (API). But many websites with useful data don't offer these convienient options.
* If we want to view data on these websites it has to be done from the website.
* If we wanted to analyze this data, or download it for use in some other app, we wouldn't want to painstakingly copy-paste everything.
* webscraping is a technique that lets us to use the programming to do the heavy lifting.
* we can write some code that looks at the website, grabs just the data we want to work with, and outputs it in the format we need.
* we can perform webscraping using python requests and BeautifulSoup library and then analyzing them using pandas and visualization libraries.
* There are other programming languages to do webscraping like R but here we learn using Python.

How does webscraping work?

* When we scrape the web, we write code that sends a request to the server that's hosting the page we specified.
* The server will return the source code - HTML, mostly - for the page (or pages) we requested.
* So far, we're essentially doing the same thing a web browser does - sending a server request with a specific url, and asking the server to return the code for that page.
* But unlike a web browser, our webscraping code won't interpret the page's source code and display the page visually.
* Instead, we'll write some custom code that filters through the page's source code looking for specific elements we've specified, and extracting whatever content we've instructed it to extract.
* For example, if we want to get all of the data from inside a table that was displayed on a webpage, our code would be written to go through these steps in sequence:
    1. Request the content (source code) of a specific from the server
    2. Download the content that is returned.

3. Identify the elements of the page that are part of the table we want.
4. Extract and (if necessary) reformat those elements into a dataset we can analyze or use in whatever way we require.

* Python and BeautifulSoup have built-in features designed to make this relatively straight forward.
* webscraping consumes lot of resources from the website owner's server.
* Python has lots of tutorials, how to videos and bits of example code out there to help once we have mastered the BeautifulSoup basics.

webscraping best practices:-

- Always check if webscraping particular website is legal or not.
- Never scrape more frequently than you need to.
- Consider caching the content we scrape so that it's only downloaded once.
- Build pauses in to our code using functions like time.sleep() to keep from overlapping servers with too many requests too quickly.

Sample python code for webscraping

```
import requests
from bs4 import BeautifulSoup

# we need to collect the website url we want to
  scrape from
      url = "https: ------"

# we request access we need to get a result of
  200      variable          request method
                ↓              ↓
      page = requests.get(url)

      page
        ↳ Response [200]  ✓ Proceed

      if Response is
          ↳ [400, 404, 500, 505, 403...]  X don't proceed

# we need to view the text
      page.text              # requests attribute
        ↳ '...'

# this format is not easily readable.
# we can use BeautifulSoup to get html format
  readable text
      BeautifulSoup(page.text)
```

we create a soup object and can use methods on it

```
soup = BeautifulSoup(page.text)
```

Depending on what we want to collect we will go to that place and right click on it and select inspect to see the html code.

from here we will copy the class and the tag then we can use find_all to get all the elements with that tag and class.

```
soup.find_all("tagname", class_="classname")
```

this has a datatype of list so we use indexing to get what we want

ex:- `soup.find_all("div", class_="....")`

so `soup.find_all("....")[0].text`

↳ returns text from first element in this tag & class.

After many manipulations we will collect information on all features and store it in to lists.

using these lists we can create DataFrame with the help of DataFrame creation with Dictionary.

Once DataFrame is created we will do Data Analysis on it
    - Data cleaning based on problem statement
    - Data Analysis based on problem statement

DataFrame once created needs to be exported to .csv file.

During Data cleaning and Data Analysis we will import .csv file and use it.

web scraping and EDA project

This project helps us to apply python, Numpy, Pandas, matplotlib, Regular Expression, webscraping and EDA skill sets. This project will help us in business understanding, data cleaning and data visualization in a life cycle of Data Science projects.

Steps :-

1) case study selection
2) search for Relevant websites
3) Define the problem statement
4) Extract the Data
    - min 400 rows x 8 columns
5) Create a DataFrame.
6) Export in to .csv format
7) Read csv file
    - How many features (columns) do we have?
    - How many observations (rows) do we have?
    - what is the datatype of each feature.
    - How many missing values are there.
8) Clean the Data
    - Removing the special characters
    - Incorrect headers
    - Incorrect format of data (Invalid values, columns)
    - Converting the data types

- Imputing the missing values
- Identifying and testing the outliers

Note: Fixing Rows and columns

we can merge different columns if it helps in better understand the data. Similarly we can also split one column in to multiple columns based on our requirements or understanding. Add column names, it is very important to have appropriate column names in your DataFrame.

9) Data Analysis and visualization (EDA)

As we can see these two data types in statistics.

1. Categorical
2. Numerical data (continous data & Discrete data)

Uni-Variant analysis

* Numerical data - Boxplot, Histogram/PDF, violin plot, and others
* Categorical data - Count plot, bar plot, pie chart etc

Bi-Variant/multi-Variant Analysis: Scatter plot and hue added to other plots using seaborn.

Groupby, Pivot, Crosstab

- Continous and categorical variables (groupby, pivot table)
- Continous to continous variables (correlation plot)
- Categorical to categorical variables (cross tab)

Plots can be drawn based on the data features/columns:

[Box-plot, Bar-plot, count plot, Pie chart, scatter plot, Violin-plot, distribution-plot, heatmap, histogram and kde-plot etc, use all plots for individual variables]

10) Interpretations (must)

write the interpretations / comments of each stage in the above steps in the jupyter file for better understanding of all.

11) Conclusion

At last give the interpretations for what we infer about our problem statement.

12) Presentation and project VIVA :

Based on the project completion we have to give the presentations about what we infer from our business problem.