**Simple APIs (Part 1)**

In this video we will discuss Application Program Interfaces, or APIs for short. Specifically, we will discuss: What an API is, API Libraries, and REST APIs, including: Request and Response and an example with PyCoinGecko. An API lets two pieces of software talk to each other. For example, you have your program, you have some data, and you have other software components. You use the API to communicate with other software via inputs and outputs. Just like a function, you don’t have to know how the API works, just its inputs and outputs. Pandas is actually a set of software components, much of which are not even written in Python. You have some data. You have a set of software components. We use the pandas API to process the data by communicating with the other Software Components. Let’s clean up the diagram. When you create a dictionary, and then create a pandas object with the Dataframe constructor, in API lingo, this is an “instance.” The data in the dictionary is passed along to the pandas API. You then use the dataframe to communicate with the API. When you call the method head, the dataframe communicates with the API displaying the first few rows of the dataframe. When you call the method mean the API will calculate the mean and return the values.

REST APIs are another popular type of API; they allow you to communicate through the internet letting you to take advantage of resources like storage, access more data, artificial intelligent algorithms, and much more. The RE stands for Representational, the S stands for State, the T stand for Transfer. In rest APIs your program is called the client. The API communicates with a web service you call through the internet. There is a set of rules regarding Communication, Input or Request, and Output or Response. Here are some common terms. You or your code can be thought of as a client. The web service is referred to as a resource. The client finds the service via an endpoint. We will review this more in the next section. The client sends requests to the resource and the response to the client. HTTP methods are a way of transmitting data over the internet We tell the Rest APIs what to do by sending a request. The request is usually communicated via an HTTP message. The HTTP message usually contains a JSON file. This contains instructions for what operation we would like the service to perform. This operation is transmitted to the webservice via the internet. And the service performs the operation. In a similar manner, the webservice returns a response via an HTTP message, where the information is usually returned via a JSON file. This information is transmitted back to the client. Crypto currency data is excellent to use in an API because it is constantly updated and is vital to Crypto currency trading. We will use the PyCoinGecko Python client or wrapper for the Coin Gecko API, updated every minute by CoinGecko We use the wrapper, or client, because it is easy to use so you can focus on the task of collecting data, we will also introduce pandas time series functions for dealing with time series data. Using PyCoinGecko to collect data is simple. All we need is to install and import the library, then create a client object, and finally, use a function to request our data. In this function we are getting data on bitcoin, in US Dollars, for the past 30 days. In this case our response is a JSON expressed as a Python dictionary of nested lists including price, market cap, and total volumes which contain the Unix timestamp and the price at that time. We are only interested in price so that is what we will select using the key price. To make things simple, we can convert our nested list to a DataFrame. With the columns time stamp and price, it is difficult to understand the column time stamp. We will convert it to a more readable format using the pandas function to\_datetime.

Using this to\_datetime function, we create readable time data, the input is the time stamp column, unit of time is set to milliseconds. We append the output to the new column date.

Now we want to create a candle stick plot. To get the data for the daily candlesticks, we will group by the date to find the minimum, maximum, first, and last price of each day. Finally, we will use plotly to create the candlestick chart and plot it. Now we can view the candlestick chart by opening the HTML file and clicking trust HTML in the top left of the tab. It should look something like this.

**Simple APIs (Part 2)**

In this video, we will discuss Application Program Interfaces that use some kind of artificial intelligence. We will transcribe an audio file using the Watson Text to Speech API. We will then translate the text to a new language using the Watson Language Translator API. In the API call, you will send a copy of the audio file to the API. This is sometimes called a POST request. Then the API will send the text transcription of what the individual is saying. Under the hood, the API is making a GET request. We then send the text we would like to translate into a second language to a second API. The API will translate the text and send the translation back to you. In this case, we translate English to Spanish. We then provide an overview of API keys and endpoints, Watson Speech to Text, and Watson Translate. First, we will review API keys and endpoints. They will give you access to the API. An API key as a way to access the API. It's a unique set of characters that the API uses to identify you and authorize you. Usually, your first call to the API includes the API key. This will allow you access to the API. In many APIs, you may get charged for each call. So like your password, you should keep your API key a secret. An endpoint is simply the location of the service. It's used to find the API on the Internet just like a web address. Now, we will transcribe an audio file using the Watson Text to Speech API. Before you start the lab, you should sign up for an API key. We will download an audio file into your directory. First, we import SpeechToTextV1 from IBM Watson. The service endpoint is based on the location of the service instance. We store the information in the variable url\_s2t. To find out which URL to use, view the service credentials. You will do the same for your API key. You create a speech-to-text adapter object. The parameters are the endpoint and API key. You will use this object to communicate with the Watson Speech to Text service. We have the path of the wav file we would like to convert to text. We create the file object wav with the wav file using open. We set the mode to rb, which means to read the file in binary format. The file object allows us access to the wav file that contains the audio. We use the method recognize from the speech to text adapter object. This basically sends the audio file to Watson Speech to Text service. The parameter audio is the file object. The content type is the audio file format. The service sends a response stored in the object response. The attribute result contains a python dictionary. The key results value has a list that contains a dictionary. We are interested in the key transcript. We can assign it to the variable recognized\_text as follows. Recognized\_text now contains a string with a transcribed text. Now let's see how to translate the text using the Watson Language Translator. First, we import LanguageTranslatorV3 from ibm\_watson. We assign the service endpoint to the variable url\_lt. You can obtain the service in the lab instructions. You require an API key, see the lab instructions on how to obtain the API key. This API request requires the date of the version, see the documentation. We create a language translator object, LanguageTranslator. We can get a list of the languages that the service can identify as follows. The method returns the language code. For example, English has a symbol en to Spanish, which has the symbol es. In the last section, we assigned the transcribed texts to the variable to recognized\_text. We can use the method translate. This will translate the text. The result is a detailed response object. The parameter text is the text. model\_id is the type of model we would like to use. In this case, we set it to en-es for English to Spanish. We use the method get result to get the translated text and assign it to the variable translation. The result is a dictionary that includes the translation word count and character count. We can obtain the translation and assign it to the variable spanish\_translation as follows. Using the variable spanish\_translation, we can translate the text back to English as follows. The result is a dictionary. We can obtain the string with the text as follows. We can then translate the text to French as follows.

**REST APIs & HTTP Requests - Part 1**

In this video, we will discuss the HTTP protocol. Specifically, we will discuss: Uniform Resource Locator: URL Request Response We touched on REST APIs in the last section. The HTTP protocol can be thought of as a general protocol of transferring information through the web. This includes many types of REST APIs. Recall that REST API’s function by sending a request, and the request is communicated via HTTP message. The HTTP message usually contains a JSON file. When you, the client, use a web page your browser sends an  HTTP request to the server where the page is hosted. The server tries to find the desired resource by default "index.html". I\ if your request is successful, the server will send the object to the client in an HTTP response; this includes information like the type of the resource, the length of the resource, and other information. The table under the Web server represents a list of resources stored in the web server. In this case, an HTML file, png image, and txt file . When the request is made for the information, the web servers sends the the requested information, i.e., one of the files. Uniform Resource Locator: URL Uniform resource locator (URL) is the most popular way to find resources on the web. We can break the URL into three parts. the scheme this is the protocol, for this lab it will always be http:// Internet address or Base URL this will be used to find the location; some examples include www.ibm.com and www.gitlab.com route: this is the location on the web server; for example: /images/IDSNlogo.png Let’s review the request and Response process. The following is an example of the request message for the get request method. There are other HTTP methods we can use. In the start line we have the GET method. This is an HTTP method. In this case, it’s requesting the file index.html The Request header passes additional information with an HTTP request. In the GET method the Request header is empty. Some Requests have a body; we will have an example of a request body later. The following table represents the response. The response start line contains the version number followed by a descriptive phrase. In this case, HTTP/1.0 a status code (200) meaning success, and the descriptive phrase, OK. We have more on status codes later. The response header contains information. Finally, we have the response body containing the requested file, in this case an HTML document. Lets look at other status codes. Some status code examples are shown in the table below. The prefix indicates the class; for example, the 100s are informational responses; 100 indicates that everything is OK so far. The 200s are Successful responses: For example, 200 The request has succeeded. Anything in the 400s is bad news. 401 means the request is unauthorized. 500’s stands for server errors, like 501 for not Implemented. When an HTTP request is made, an  HTTP method is sent. This tells the server what action to perform. A list of several HTTP methods is shown here. In the next video, we will use Python to apply the GET method that Retrieves data from the server and the post method that sends data to the server.

**REST APIs & HTTP Requests - Part 2**

In this video, we will discuss the HTTP protocol using the Requests Library a popular method for dealing with the HTTP protocol in Python In this video, we will review Python library requests for Working with the HTTP protocols. We will provide an overview of Get Requests and Post Requests Let’s review the Request Module in Python. This is one of several libraries including: httplib, urllib, that can work with the HTTP protocol. Requests is a python Library that allows you to send HTTP/1.1 requests easily. We can import the library as follows: You can make a GET request via the method get to www.ibm.com: We have the response object ’r’ , this has information about the request, like the status of the request. We can view the status code using the attribute status\_code, which is 200 for OK. You can view the request headers: You can view the request body in the following line. As there is no body for a GET request, we get a None. You can view the HTTP response header using the attribute headers. This returns a python dictionary of HTTP response headers. We can look at the dictionary values. We can obtain the date the request was sent by using the key Date. The key Content-Type indicates the type of data. Using the response object ‘r’ , we can also check the encoding: As the Content-Type is text/html, we can use the attribute text to display the HTML in the body. We can review the first 100 characters. You can also download other content, see the lab for more. You can use the GET method to modify the results of your query. For example, retrieving data from an API. In the lab we will use httpbin.org A simple HTTP Request & Response Service. We send a GET request to the server. Like before, we have the Base URL in the Route; we append /get. This indicates we would like to preform a GET request. This is demonstrated in the following table: After GET is requested we have the query string. This is a part of a uniform resource locator (URL) and this sends other information to the web server. The start of the query is a ?, followed by a series of parameter and value pairs, as shown in the table below. The first parameter name is ”name” and the value is ”Joseph.” The second parameter name is ”ID” and the Value is ”123.” Each pair, parameter, and value is separated by an equal sign, ”=.” The series of pairs is separated by the ampersand, ”&.” Let’s complete an example in python. We have the Base URL with GET appended to the end. To create a Query string, we use the dictionary payload. The keys are the parameter names, and the values are the value of the Query string. Then, we pass the dictionary payload to the params parameter of the get() function. We can print out the URL and see the name and values. We can see the request body. As the info is sent in the URL, the body has a value of None. We can print out the status code. We can view the response as text: We can look at the key 'Content-Type’ to look at the content type. As the content 'Content-Type' is in the JSAON, we format it using the method json() . It returns a Python dict: The key 'args' has the name and values for the  query string. Like a GET request a POST request is used to send data to a server, but the POST request sends the data in a request body, not the url. In order to send the Post Request in the URL, we change the route to POST: This endpoint will expect data and it is a convenient way to configure an HTTP request to send data to a server. We have The Payload dictionary. To make a POST request, we use the post() function. The variable payload is passed to the parameter data : Comparing the URL using the attribute url from the response object of the GET and POST request, we see the POST request has no name or value pairs in it’s url. We can compare the POST and GET request body. We see only the POST request has a body: We can view the key form to get the payload.

**Optional: HTML for Webscraping**

In this video we will review Hypertext Markup Language or HTML for Webscraping. Lots of useful data is available on web pages, such as real estate prices and solutions to coding questions. The website Wikipedia is a repository of the world's information. If you have an understanding of HTML, you can use Python to extract this information. In this video, you will: review the HTML of a basic web page; understand the Composition of an HTML Tag; understand HTML Trees; and understand HTML Tables. Let’s say you were asked to find the name and salary of players in a National Basketball League from the following web page. The web page is comprised of HTML. It consists of text surrounded by a series of blue text elements enclosed in angle brackets called tags. The tags tells the browser how to display the content. The data we require is in this text. The first portion contains the "DOCTYPE html” which declares this document is an HTML document.

<html> element is the root element of an HTML page, and <head> element contains meta information about the HTML page. Next, we have the body, this is what's displayed on the web page. This is usually the data we are interested in, we see the elements with an “h3”, this means type 3 heading, makes the text larger and bold. These tags have the names of the players, notice the data is enclosed in the elements. It starts with a h3 in brackets and ends in a slash h3 in brackets. There is also a different tag “p”, this means paragraph, each p tag contains a player's salary. Let’s take a closer look at the composition of an HTML tag. Here is an example of an HTML Anchor tag. It will display IBM and when you click it, it will send you to IBM.com. We have the tag name, in this case “a”. This tag defines a hyperlink, which is used to link from one page to another. It’s helpful to think of each tag name as a class in Python and each individual tag as an instance. We have the opening or start tag and we have the end tag. This has the tag name preceded by a slash. These tags contain the content, in this case what’s displayed on the web page. We have the attribute, this is composed of the Attribute Name and Attribute Value. In this case it the url to the destination web page. Real web pages are more complex, depending on your browser you can select the HTML element, then click Inspect. The result will give you the ability to inspect the HTML. There is also other types of content such as CSS and JavaScript that we will not go over in this course. The actual element is shown here. Each HTML document can actually be referred to as a document tree. Let's go over a simple example. Tags may contain strings and other tags. These elements are the tag’s children. We can represent this as a family tree. Each nested tag is a level in the tree. The tag HTML tag contains the head and body tag. The Head and body tag are the descendants of the html tag. In particular they are the children of the HTML tag. HTML tag is their parent. The head and body tag are siblings as they are on the same level. Title tag is the child of the head tag and its parent is the head tag. The title tag is a descendant of the HTML tag but not its child. The heading and paragraph tags are the children of the body tag; and as they are all children of the body tag they are siblings of each other. The bold tag is a child of the heading tag, the content of the tag is also part of the tree but this can get unwieldy to draw.

Next, let’s review HTML tables. To define an HTML table we have the table tag. Each table row is defined with a  <tr>  tag, you can also use a table header tag for the first row. The table row cell contains a set of <td> tags, each defines a table cell. For the first row first cell we have; for the first row second cell we have; and so on. For the second row we have; and for the second row first cell we have; for the second row second cell we have; and so on. We now have some basic knowledge of HTML. Now let's try and extract some data from a web page.

**Webscraping**

In this video we will cover Webscraping. After watching this video you will be able to: define web scraping; understand the role of BeautifulSoup Objects; apply the find\_all method; and webscrape a website. What would you do if you wanted to analyze hundreds of points of data to find the best players of a sports team? Would you start manually copying and pasting information from different websites into a spreadsheet? Spending hours trying to find the right data, and eventually giving up because the task was to overwhelming? That’s where webscraping can help. Webscraping is a process that can be used to automatically extract information from a website, and can easily be accomplished within a matter of minutes and not hours. To get started we just need a little Python code and the help of two modules named Requests and Beautiful Soup. Let’s say you were asked to find the name and salary of players in a National Basketball League, from the following webpage.

We import BeautifulSoup. We can store the webpage HTML as a string in the variable HTML. To parse a document, pass it into the BeautifulSoup constructor. We get the Beautiful Soup object , soup, which represents the document as a nested data structure. BeautifulSoup represents HTML as a set of Tree like objects with methods used to parse the HTML. We will review the BeautifulSoup object Using the BeautifulSoup object, soup, we created The tag object corresponds to an HTML tag in the original document. For example, the tag “title.” Consider the tag <h3>. If there is more than one tag with the same name, the first element with that tag is selected. In this case with Lebron James, we see the name is Enclosed in the bold attribute "b". To extract it, use the Tree representation. Let’s use the Tree representation. The variable tag-object is located here. We can access the child of the tag or navigate down the branch as follows: You can navigate up the tree by using the parent attribute. The variable tag child is located here. We can access the parent. This is the original tag object. We can find the sibling of “tag object.” We simply use the next sibling attribute. We can find the sibling of sibling one. We simply use the next sibling attribute. Consider the tag child object. You can access the attribute name and value as a key value pair in a dictionary as follows. You can return the content as a Navigable string, this is like a Python string that supports BeautifulSoup functionality. Let's review the method find\_all. This is a filter, you can use filters to filter based on a tag’s name, it’s attributes, the text of a string, or on some combination of these. Consider the list of pizza places. Like before, create a BeautifulSoup object. But this time, name it table. The find\_all () method looks through a tag’s descendants and retrieves all descendants that match your filters. Apply it to the table with the tag <tr>. The result is a Python iterable just like a list, each element is a tag object for <tr>. This corresponds to each row in the list- including the table header. Each element is a tag object. Consider the first row. For example, we can extract the first table cell. We can also iterate through each table cell. First, we iterate through the list “table rows,” via the variable row. Each element corresponds to a row in the table. We can apply the method find all to find all the table cells, then we can iterate through the variable cells for each row. For each iteration, the variable cell corresponds to an element in the table for that particular row. We continue to iterate through each element and repeat the process for each row. Let’s see how to apply BeautifulSoup to a webpage. To scrape a webpage we also need the Requests library. The first step is to import the modules that are needed. Use the get method from the requests library to download the webpage. The input is the URL. Use the text attribute to get the text and assign it to the variable page. Then, create a BeautifulSoup object ‘soup’ from the variable page. It will allow you to parse through the HTML page.

**Working with different file formats (csv, xml, json, xlsx)**

After watching this video you will be able to define different file formats such as csv xml and json write simple programs to read and output data and list what python libraries are needed to extract data when collecting data you will find there are many different file formats that need to be collected or read in order to complete a data driven story or analysis when gathering the data python can make the process simpler with its predefined libraries but before we explore python let us first check out some of the various file formats looking at a file name you will notice an extension at the end of the title these extensions let you know what type of file it is and what is needed to open it for instance if you see a title like file example.csv you will know this is a csv file but this is only one example of different file types there are many more such as json or xml when coming across these different file formats and trying to access their data we need to utilize python libraries to make this process easier the first python library to become familiar with is called pandas by importing this library in the beginning of the code we are then able to easily read the different file types since we have now imported the panda library let us use it to read the first csv file in this instance we have come across the file example.csv file the first step is to assign the file to a variable then create another variable to read the file with the help of the panda library we can then call read underscore csv function to output the data to the screen with this example there were no headers for the data so it added the first line as the header since we do not want the first line of data as the header let us find out how to correct this issue now that we have learned how to read and output the data from a csv file let us make it look a little more organized from the last example we were able to print out the data but because the file had no headers it printed the first line of data as the header we easily solve this by adding a data frame attribute we use the variable df to call the file then add the columns attribute by adding this one line to our program we can then neatly organize the data output into the specified headers for each column the next file format we will explore is the json file format in this type of file the text is written in a language independent data format and is similar to a python dictionary the first step in reading this type of file is to import json after importing json we can add a line to open the file call the load attribute of json to begin and read the file and lastly we can then print the file the next file format type is xml also known as extensible markup language while the pandas library does not have an attribute to read this type of file let us explore how to parse this type of file the first step to read this type of file is to import xml by importing this library we can then use the e-tree attribute to parse the xml file we then add the column headers and assign them to the data frame then create a loop to go through the document to collect the necessary data and append the data to a data frame in this video you learned how to recognize different file types how to use python libraries to extract data and how to use data frames when collecting data