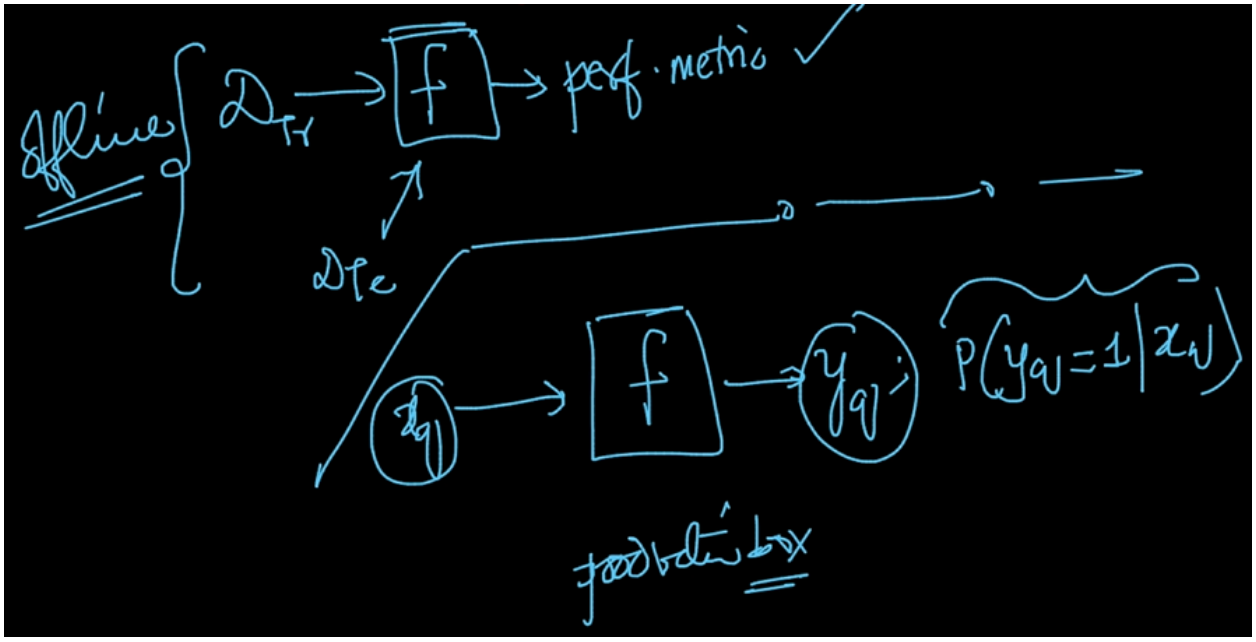
47.1 Basics of Productionization and Deployment

Productionization is an important task in Machine Learning. If we have a training dataset 'D_{Tr}', test dataset 'D_{Te}', then we have to fit the ML model 'f' on 'D_{Tr}', and make predictions on 'D_{Te}', and using any relevant performance metric on the predictions made on 'D_{Te}', we have to assess the model performance.

In productionization, as the model 'f' was already trained, we pass the real-time data through the model and make predictions on it. If we pass a query point 'x_q' along with the value 'y_q', it returns 'y_q^' (or) $P(y_q=1|x_q)$ depending on our problem requirement. Then the model performance is checked on the real-time data predictions made. Productionization is running the models live on the live data.



There are 2 options to productionize our models. They are

a) **Using sklearn**

We have a concept called **Model persistence**. In this concept, we store our ML model 'f' on a storage device (say HDD or SSD). Whenever we want to run our model in production, we have to load it from the storage into the RAM and make predictions on the real-time data.

Please go through the below reference link:
https://scikit-learn.org/stable/modules/model_persistence.html

One way to load a model is, by storing into a string, and storing the string in a disk, and then loading it into the RAM whenever needed. Below is the code snippet for this task discussed at the timestamp 2:42.

```
>>> import pickle
>>> s = pickle.dumps(clf)
>>> clf2 = pickle.loads(s)
```

The other way to load a model is, by storing it first in a file, and then loading the file into the RAM whenever needed. Below is the code snippet for this task discussed at the timestamp 3:20.

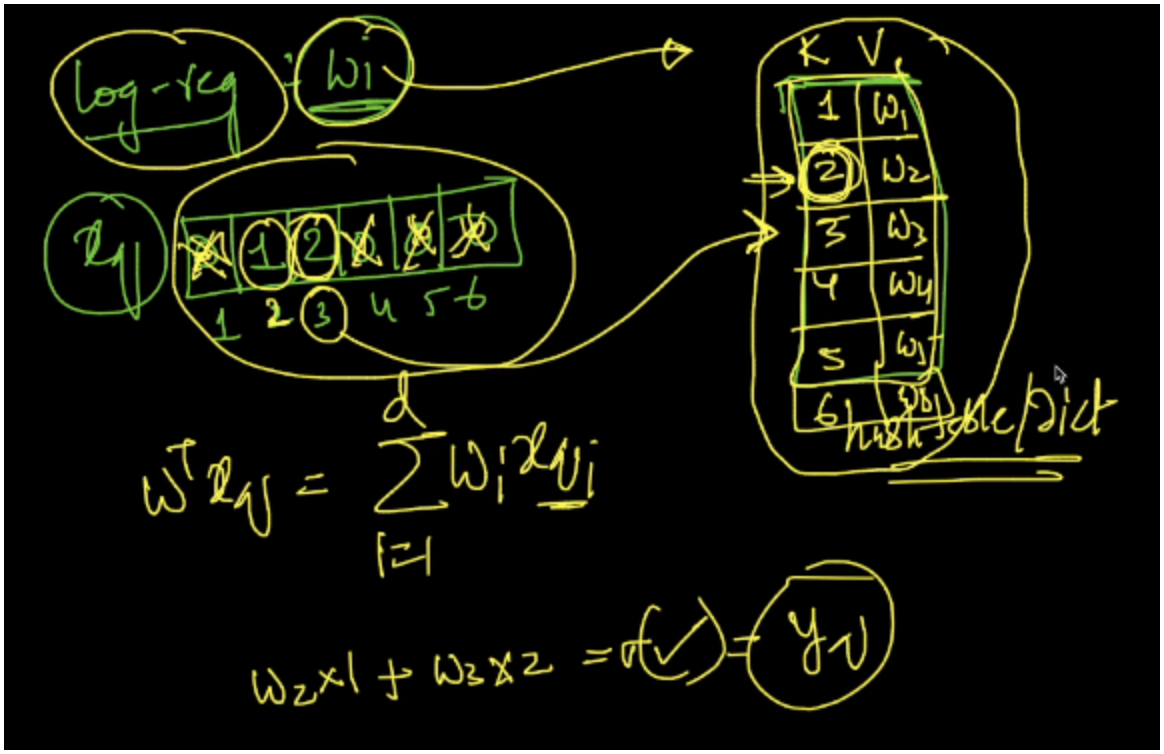
```
>>> from joblib import dump, load
>>> dump(clf, 'filename.joblib')
```

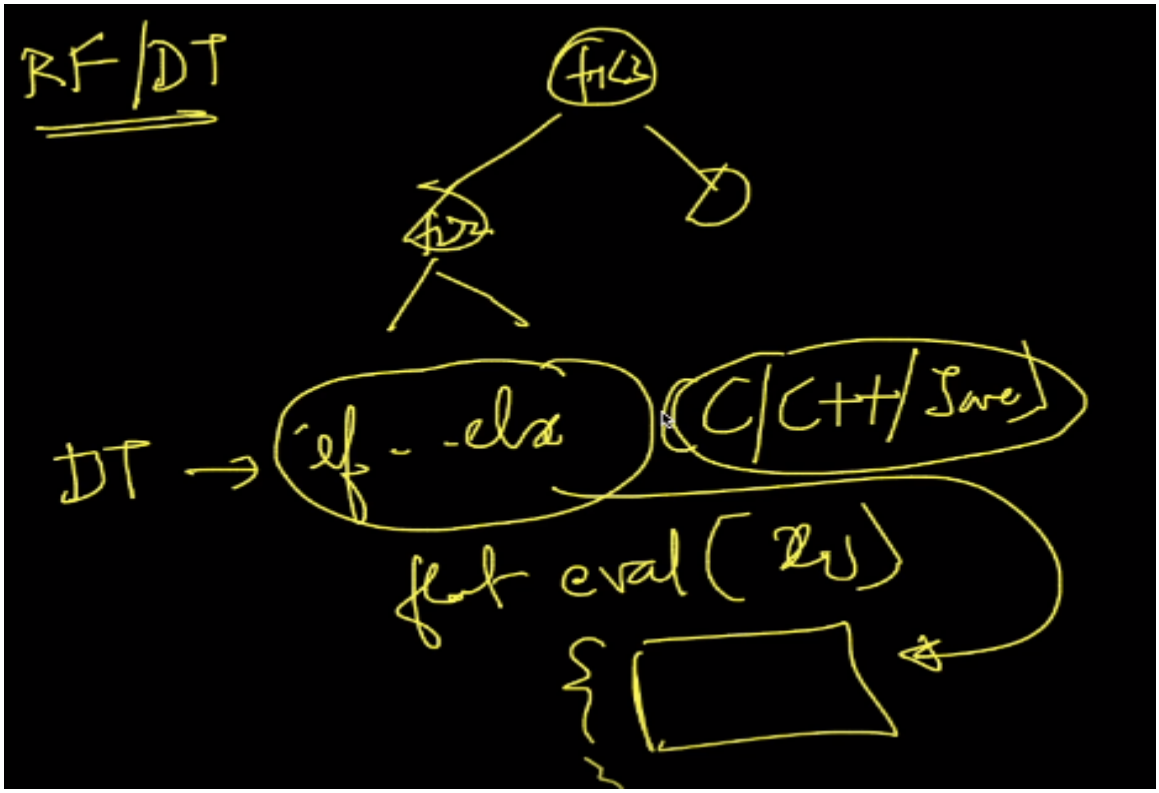
Later you can load back the pickled model (possibly in another Python process) with:

```
>>> clf = load('filename.joblib')
```

b) Custom Implementation approach

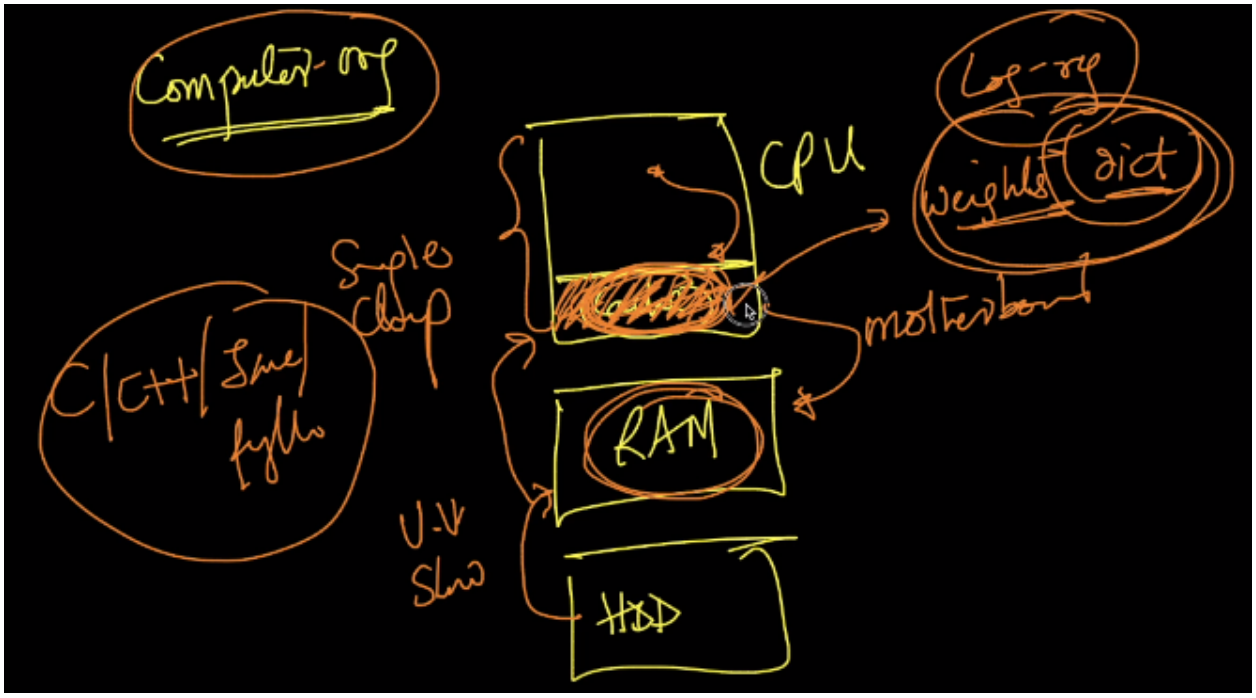
In this approach, we store all the model parameters in a file. For example, if our model is Logistic Regression, we store all the weights in key-value format. If our model is Decision Tree, then we store all the if-else conditions. Whenever we have requirements such as low latency, we go for this approach.





Whenever we have a high throughput (ie., more number of query points at a time for prediction), then the Load balancer distributes the points randomly and uniformly to all the available CPUs for faster processing.

Basics of Computer Organization



For a CPU, reading the data from the cache is faster than reading the data from the RAM. RAM is an external storage device, whereas Cache is the inbuilt storage of a CPU. When we have to read the data from the Hard-disk, then it has to be first loaded into the RAM, and then into the CPU is a very slow process. Hence the data which we need to access regularly, like the look up tables, model parameters, etc, should be stored in the CPU Cache.

47.2 Hands on Session: Deploy an ML model using Flask APIs on AWS

In this session, we are going to build and deploy a model using Flask APIs on AWS (Amazon Web Services).

Workflow :

Model ----> **API** ----> **AWS**
(Built using sklearn,XGBOOST,etc)

Before proceeding on how to deploy a model on AWS, we will first deploy it on our local machine.

A sneak peek of this session.

1.) Build a model on your local box (Amazon Fine Food Reviews) and store the model and other key model related variables in .pkl files. Pickle files are used to store python variables.
2.) Launch a micro instance on AWS.
3.) Connect to the AWS box.
4.) Move the files to an AWS EC2 instance/box[scp]
5.) Install all packages related to the AWS box.
6.) Run app.py on the AWS box.
7.) Check the output in the browser.

Software needed :

1.) Anaconda

Packages needed :

1.) Pip3
2.) Pandas
3.) Numpy

4.) Sklearn
5.) Beautifulsoup4
6.) Lxml
7.) Flask
8.) Re

Please download the required files for this session from this link.

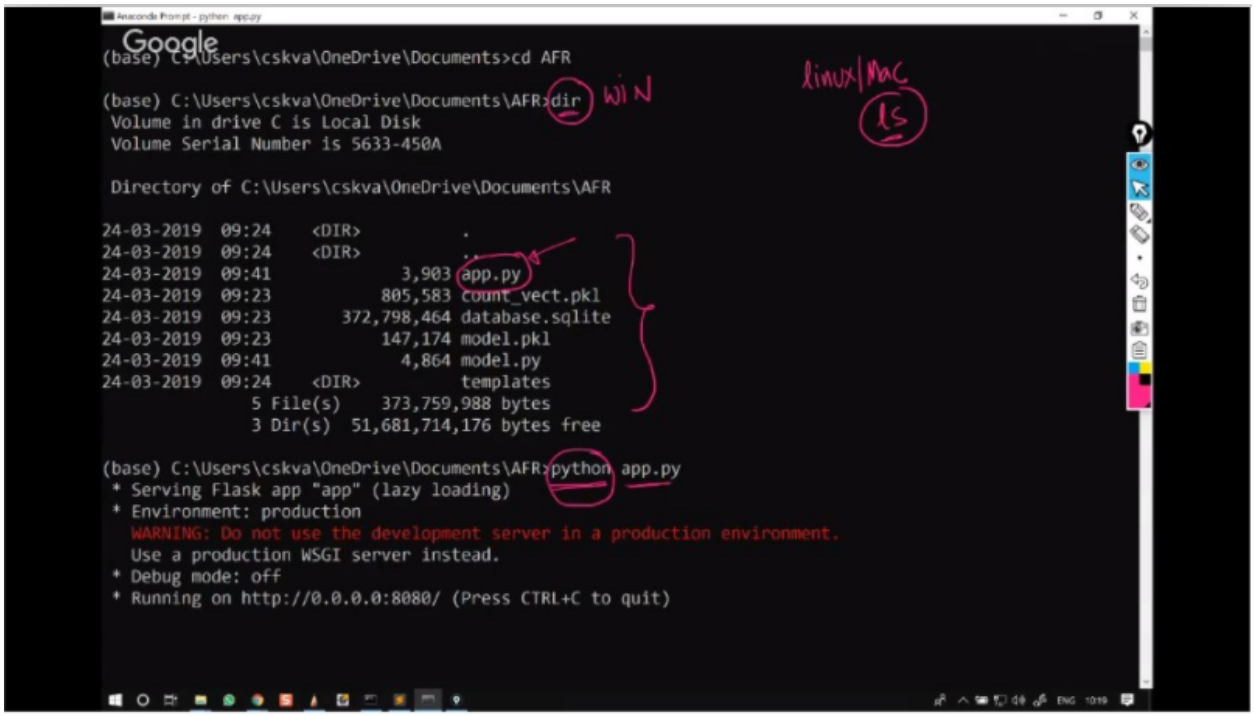
1.) Once the download is complete, unzip all the files.

2.) Then, go to the directory named AFR.

3.) Press Shift+Right click simultaneously.

4.) Click on “Open Powershell window here”.

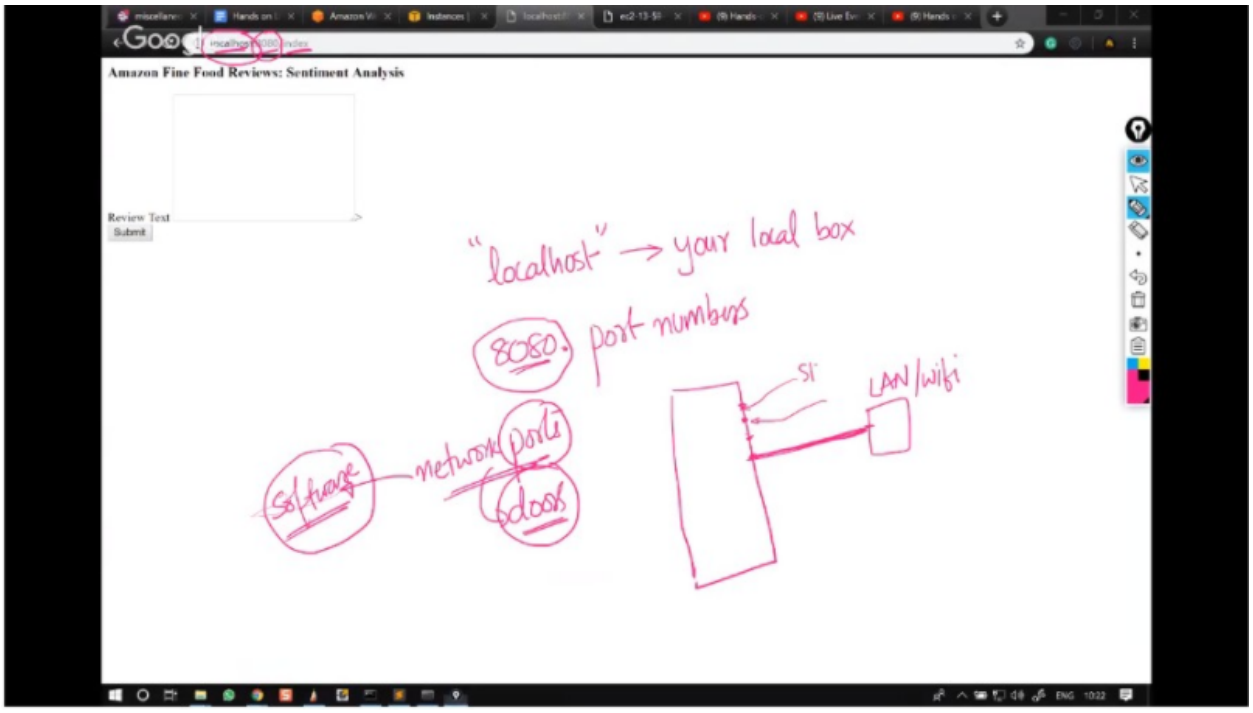
5.) Type python app.py . (We assume python is already installed).



Timestamp : 22:42

Refer to the above image for more details.

6.) Simply copy paste the link on the browser.
7.) We can see a web page loaded in our browser.



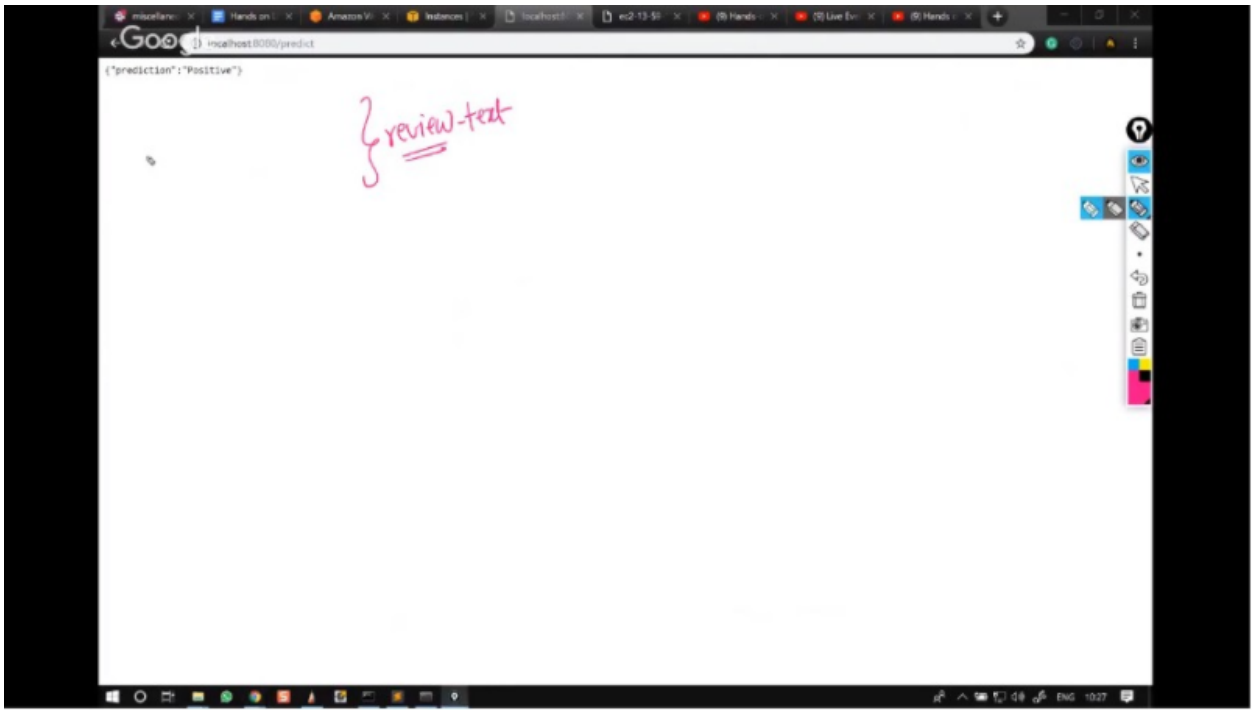
Timestamp : 25:33

Port : A port is simply a door where you can access software through it. In simple terms, computers have IP addresses and software installed in them has port numbers. To communicate to other computers over the internet , we need IP addresses. But, in order to communicate with the software installed in other computers, we need port numbers. It's simply an address given to a software where a user can use them to find it and communicate with it over the internet.

So, now we can provide a sample text on the text box and then click on the Submit button.

Note : If we press Ctrl+C on the command prompt, the application will get stopped.

Once we click on the submit button, we will get the output in another web page.



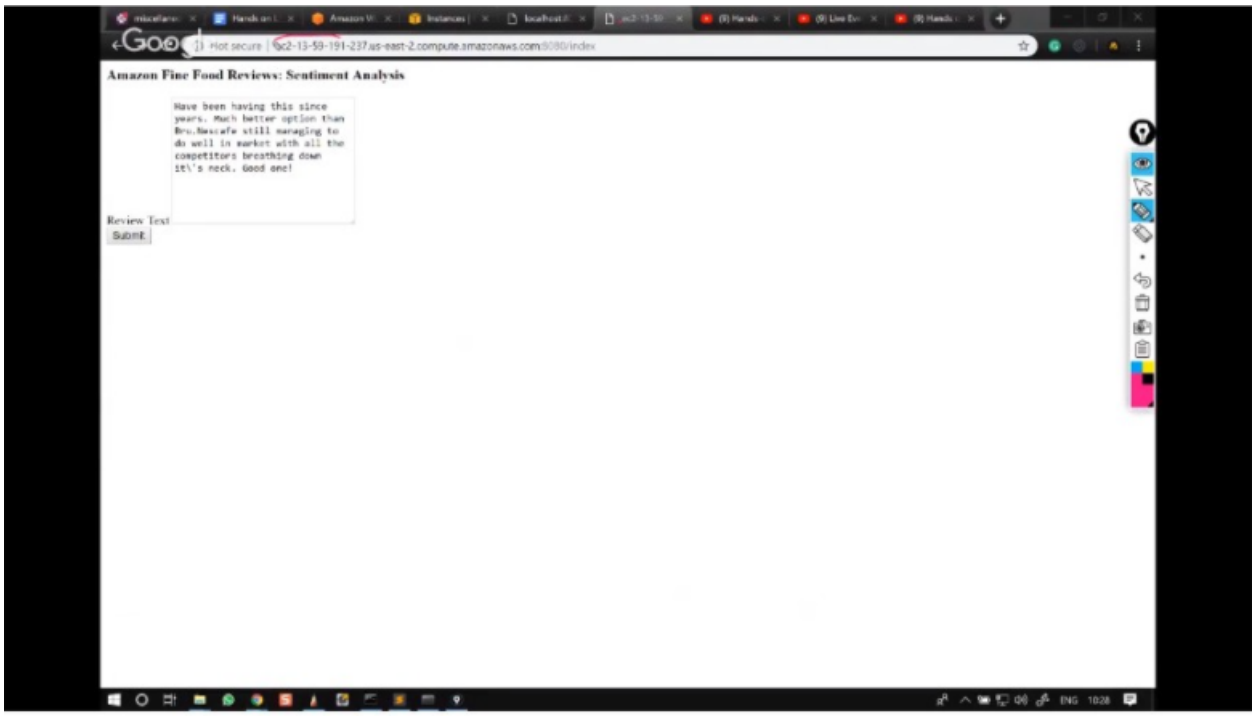
Timestamp : 30:25

We can see from the above screenshot that we got the prediction as “positive”.

In the above example , we ran it on our local machine. Now, we will try to run the same with a global web address. In simple terms, accessing the web application using http://0.0.0.0:8080/ will work only on our computer. But, if we want a web address which can work on any device, then we should deploy our application on AWS. Once deployed, we will get a web address for that specific application. Using that, we can access it from any device.

Try opening this link ec2-13-59-191-237.us-east-2.compute.amazonaws.com:8080/index on your web browser.

The same web application will be opened.



Timestamp : 31:17

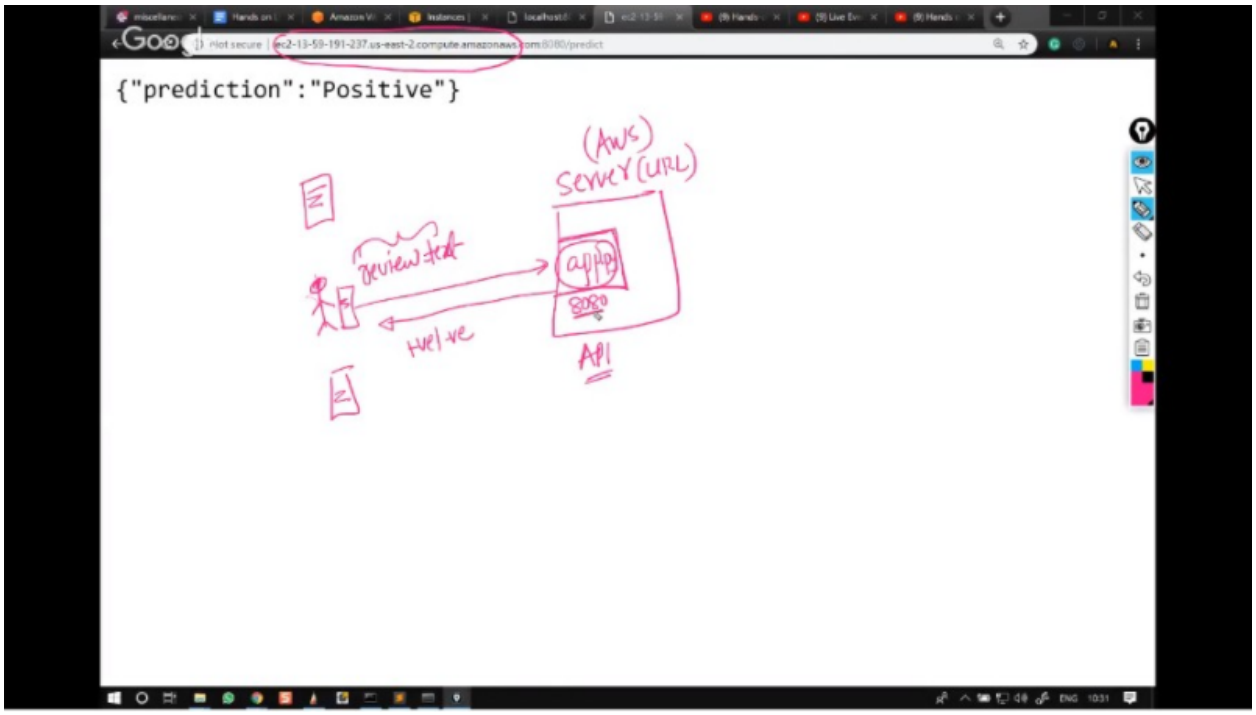
Now, you can provide a sample text on the text box and click on the submit button. Then, you can see the output.

Let’s compare both ways.

1.) In the first method, we used localhost to run or host our web application. This means, all the process,hardware to run the web application is consumed in our computer. The machine learning model is executed on our computer .
2.) In the second method, we used AWS server. So, the model is executed on the AWS server rather than in our machine and the output is sent. AWS serve is nothing but another machine sitting

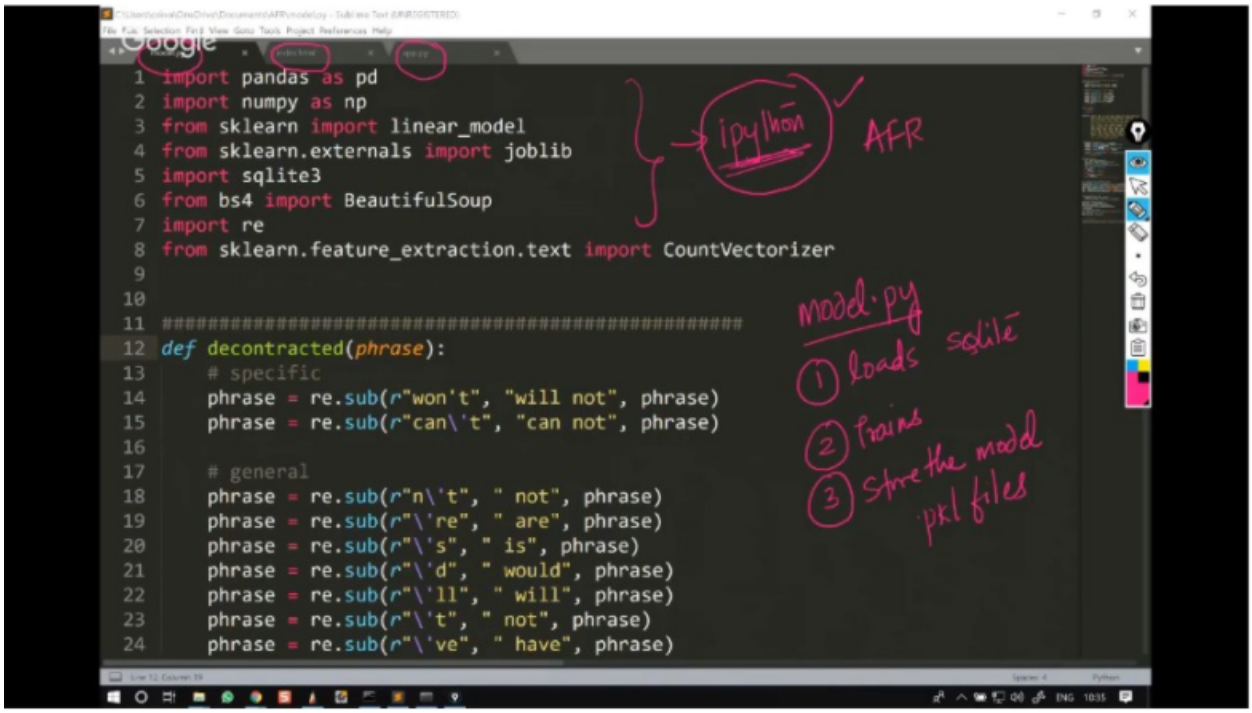
somewhere. It has good hardware and software configurations so that we can even execute complex models rather than relying on our computer or upgrading ours. This is the key difference between them.

Let’s see an illustration explaining the process.



Timestamp : 34:34

1.) First we typed the review on the text box.
2.) Then, we clicked the submit button.
3.) Once we click the submit button, the review is sent to the AWS server via API.
4.) In the AWS server, the review is passed to the model. We get the output from the model.
5.) The output is sent back as a json to our computer.
6.) Then, the output is displayed on the web page.

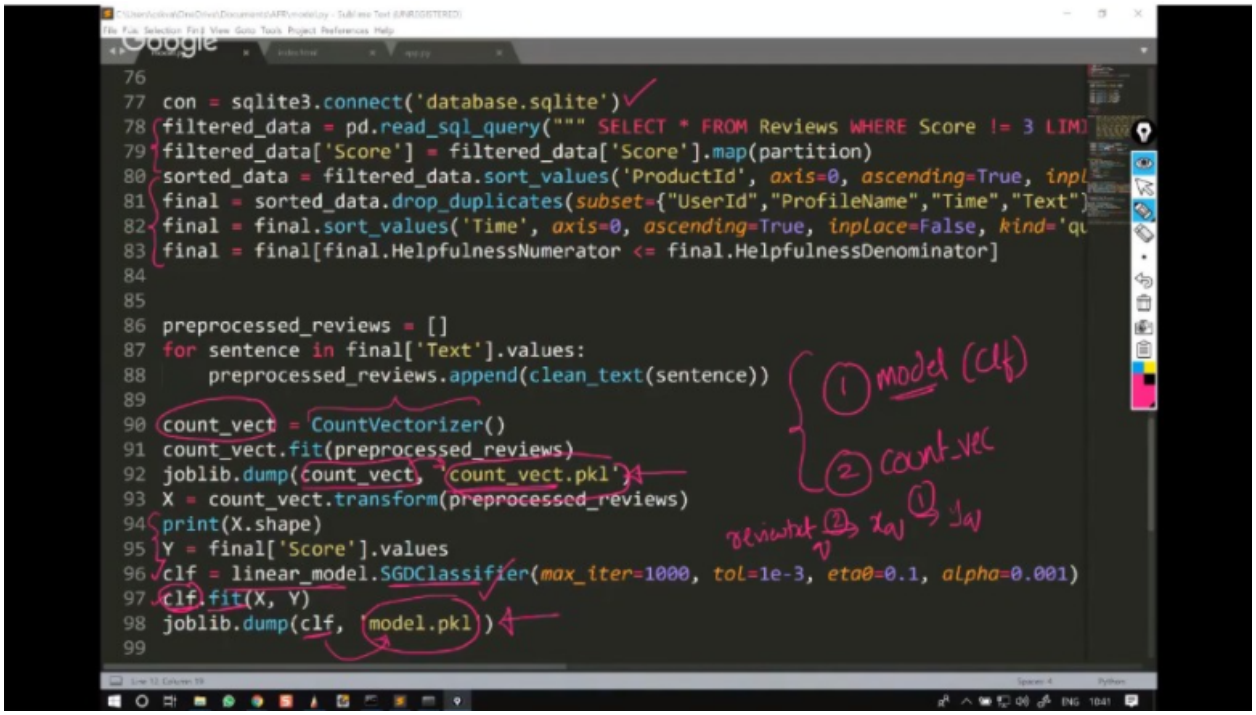


Timestamp : 38:34

Model.py

1.) Loading the data using sqlite
2.) Training the model
3.) Storing the best model in .pkl files

Some of the functions in Model.py are already covered before in our course. We will look at the functions which are necessary for this session.



Timestamp : 44:01

We need both countvectorizer and our model for prediction. So, we store them as a pickle file.

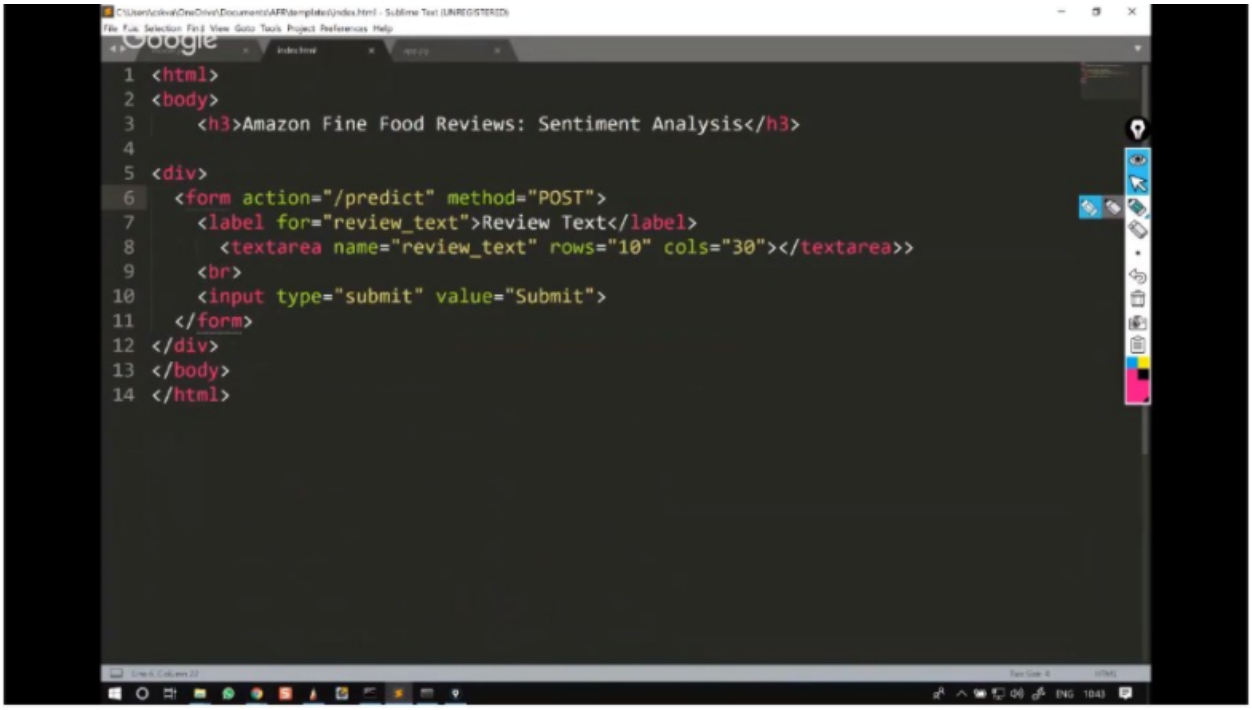
Why do we need countvectorizer?

So,given a query point in the test time, we need to convert it to numerical representation since we are dealing with text data. So, for that we use countvectorizer.

Why do we need a model ?

Without a model, we can't predict the polarity of our review.

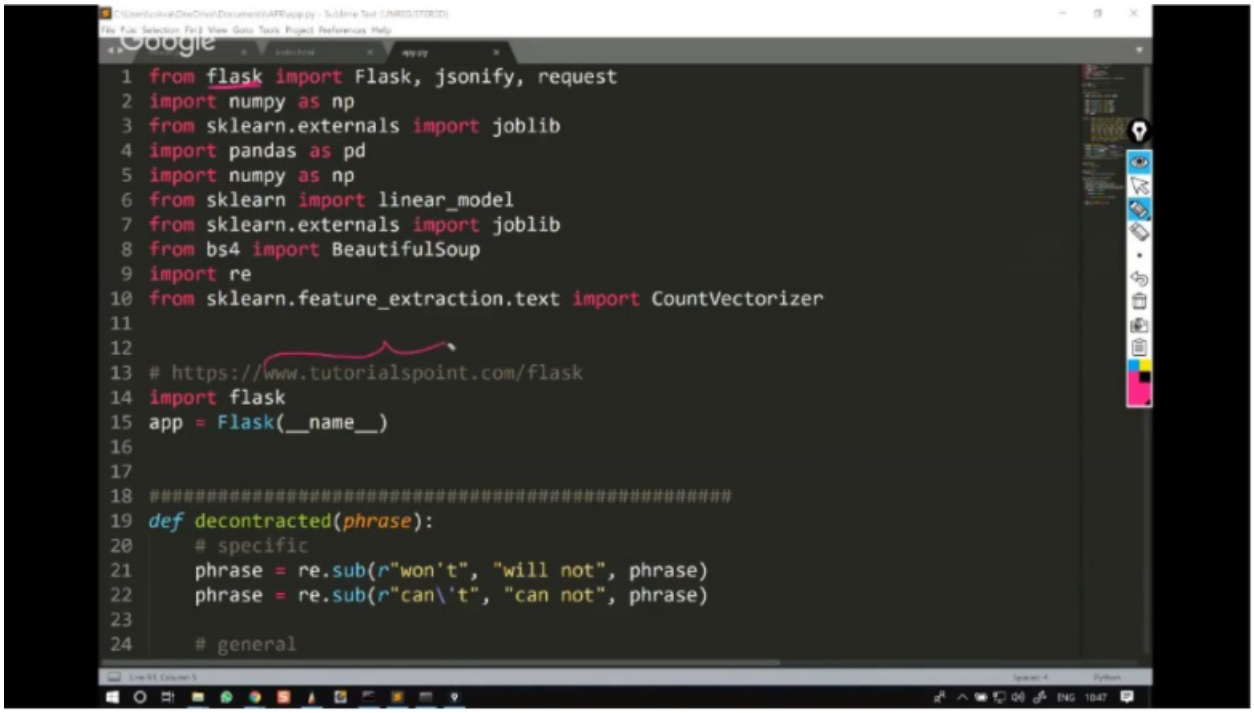
Next, we will look at index.html which resides in the templates folder.



Timestamp : 46:33

1.) The above code is written in HTML (Hyper Text Markup Language).
2.) This language is used to design the webpage.
3.) <h3> denotes the header of the web page.
4.) We are defining a form since we need to type a text in a text box.
5.) For the submit button, we are defining an input tag.
6.) Once we click on the submit button, it will call the **predict** function which is provided in the action attribute.
7.) Method = “POST” simply means, we are posting the request to the predict function.

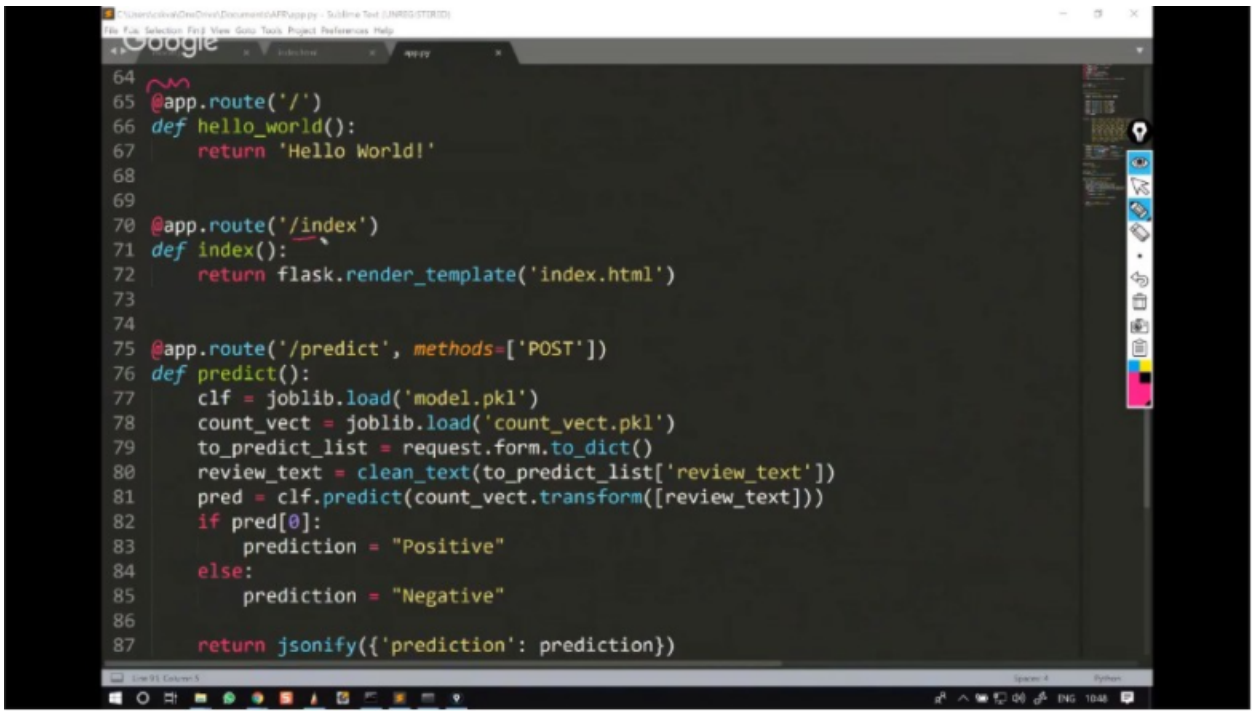
app.py



```
from flask import Flask, jsonify, request
import numpy as np
from sklearn.externals import joblib
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.externals import joblib
from bs4 import BeautifulSoup
import re
from sklearn.feature_extraction.text import CountVectorizer


# https://www.tutorialspoint.com/flask
import flask
app = Flask(__name__)


##########################################################
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
```

Timestamp : 50:21

1.) First, import the flask library using import flask
2.) app = Flask(__name__). Creating a flask instance.



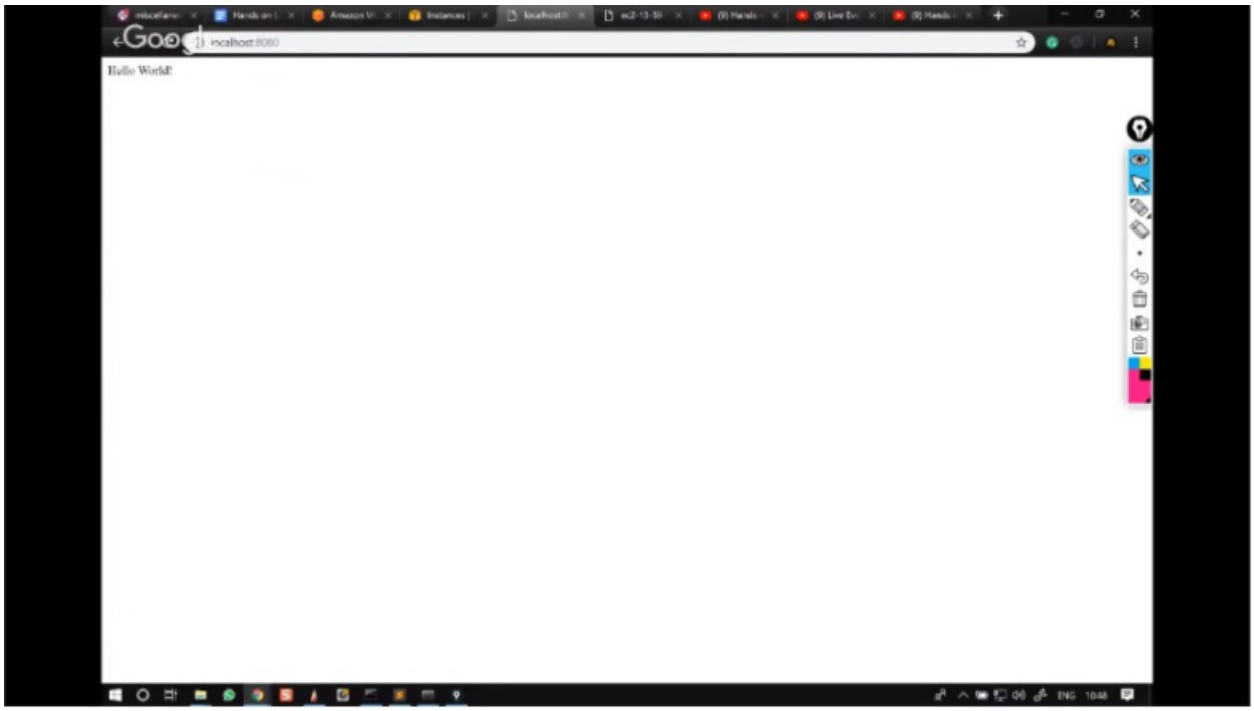
```
@app.route('/')
def hello_world():
    return 'Hello World!'


@app.route('/index')
def index():
    return flask.render_template('index.html')


@app.route('/predict', methods=['POST'])
def predict():
    clf = joblib.load('model.pkl')
    count_vect = joblib.load('count_vect.pkl')
    to_predict_list = request.form.to_dict()
    review_text = clean_text(to_predict_list['review_text'])
    pred = clf.predict(count_vect.transform([review_text]))
    if pred[0]:
        prediction = "Positive"
    else:
        prediction = "Negative"

    return jsonify({'prediction': prediction})
```

Timestamp : 51:20

@app.route(‘/’)
def hello_world():
return ‘Hello World’

This simply says, when a user tries to enter the web address http://0.0.0.0:8080/ , the web page will display “Hello World”.

@ is used to define a decorator in python.

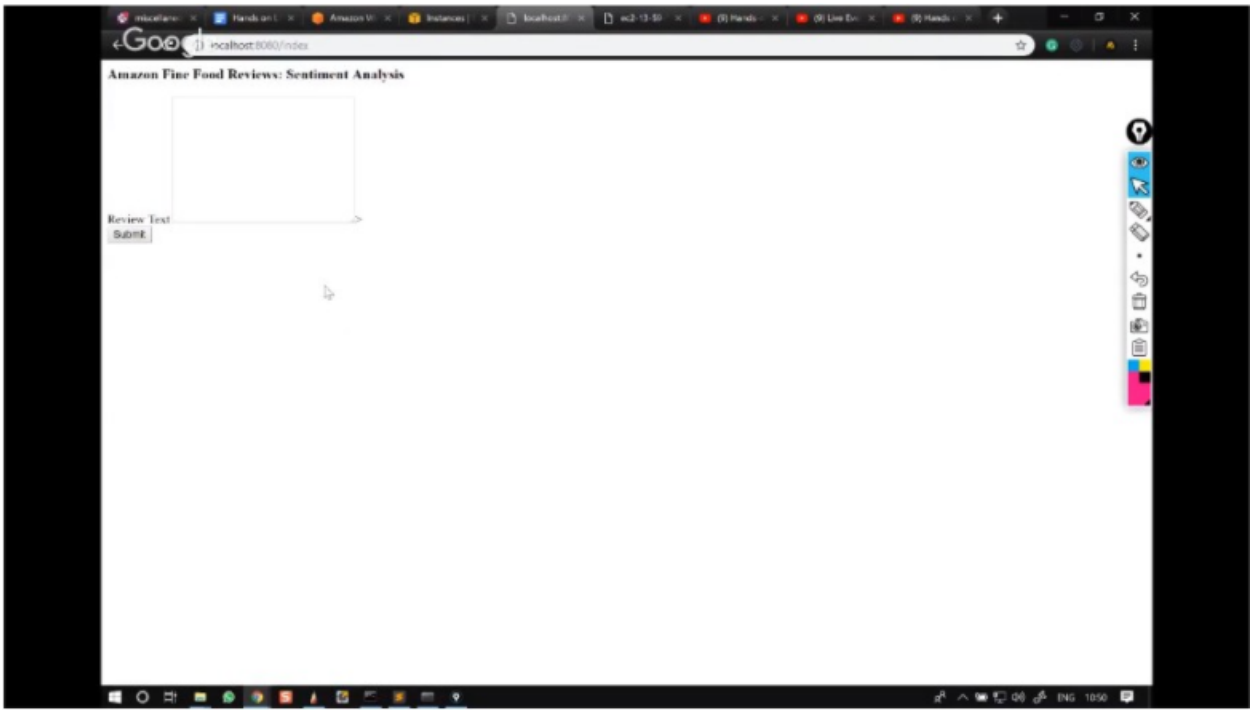


Timestamp : 51:49

We can see from the image that the text “Hello World” is displayed.

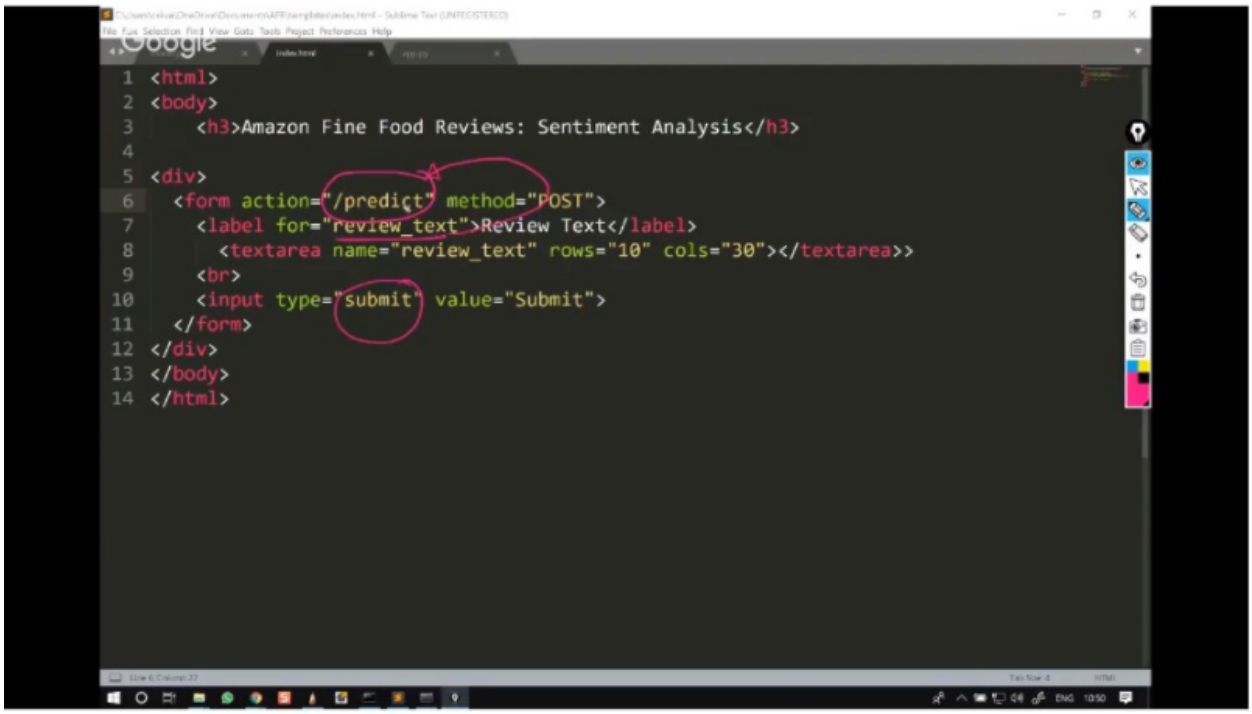
@app.route(‘/index’)
def index():
return flask.render_template(‘index.html’)

If a user tries to access this website http://0.0.0.0:8080/index , then the index.html residing in the folder will get rendered in our web page. In simple terms, in index.html we have some html code. So, when the user tries to access the above link, the index.html will get rendered in the webpage.



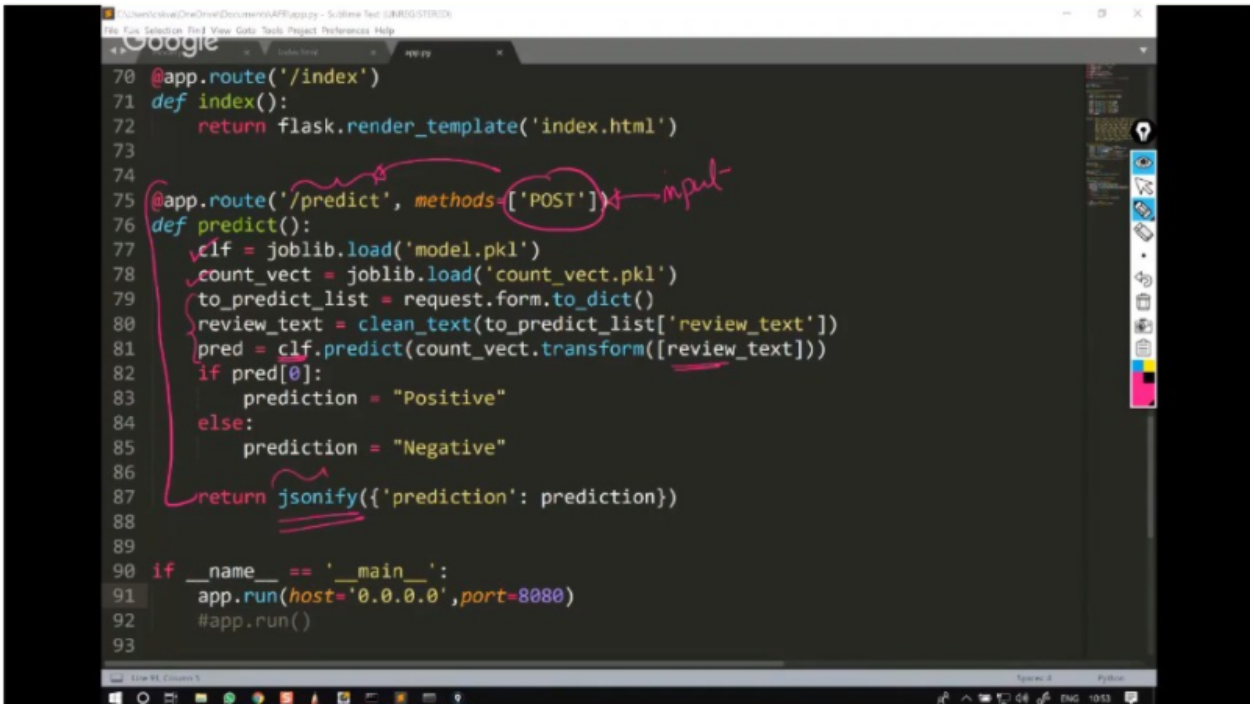
Timestamp : 53:02

The output after opening this link http://0.0.0.0:8080/index .



Timestamp : 53:49

1.) First, we will type a sample review in the text box.
2.) Then, we click on the submit button.
3.) Once the button is pressed, the review is posted to the predict function or the text is passed to the predict function which is defined in the action and method attribute.
4.) Let’s see what the predict function does.



Timestamp : 54:03

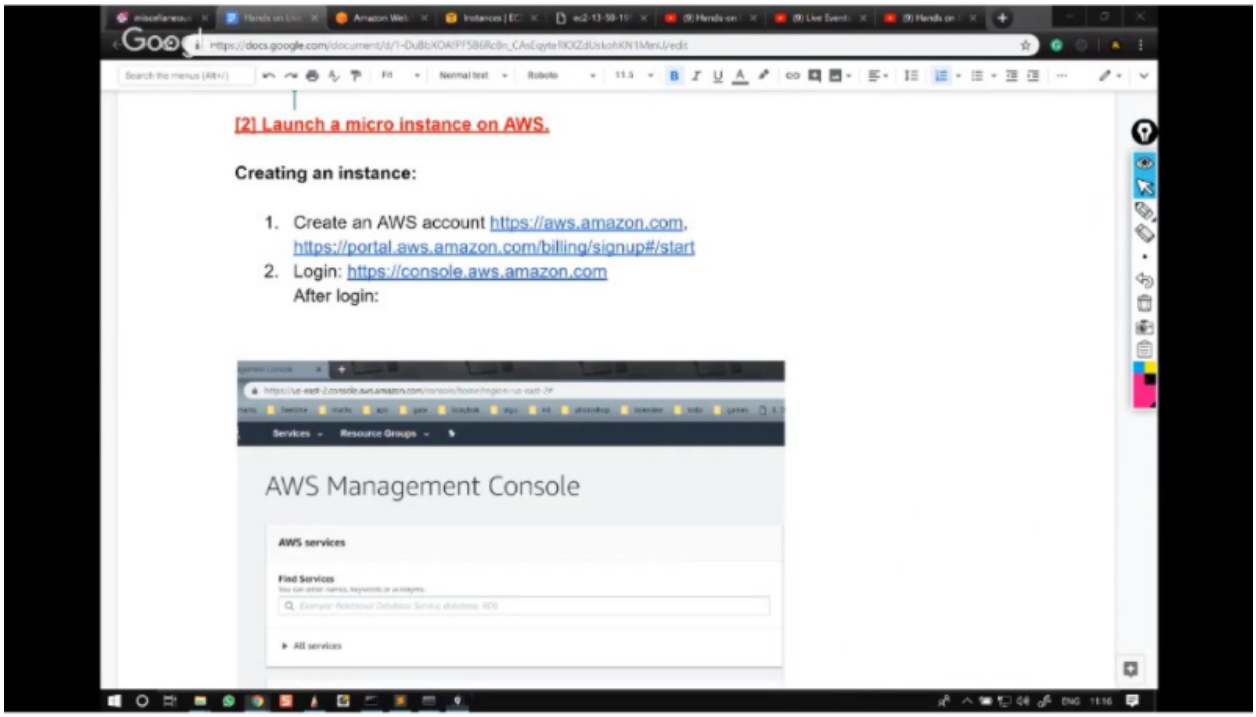
5.) route('/predict',method=['POST']) : This means, whenever a user tries to access the web address http://0.0.0.0:8080/predict , the predict function will be called in the background.
6.) In the predict function, we first load the model file which is saved in pickle format i.e, .pkl
7.) Then, we will load the count vectorizer too.
8.) Now, we need to access the text sent as a request to the predict function.
9.) For that, we are accessing the 'review_text' from the to_predict_list dictionary.
10.) Then, we will transform the text using the countvectorizer and pass it to the predict function.
11.) Then we will return the polarity of the review in json format by using the jsonify method.
12.) app.run(host='0.0.0.0',port=8080) . It simply means, the web application will be hosted on the web address http://0.0.0.0:8080.
13.) You can change the port number if you wish.

Launch a micro instance on AWS

1.) We are leasing a box on AWS which is EC2 (Elastic Compute Cloud).
2.) Then, we are hosting our model on it.

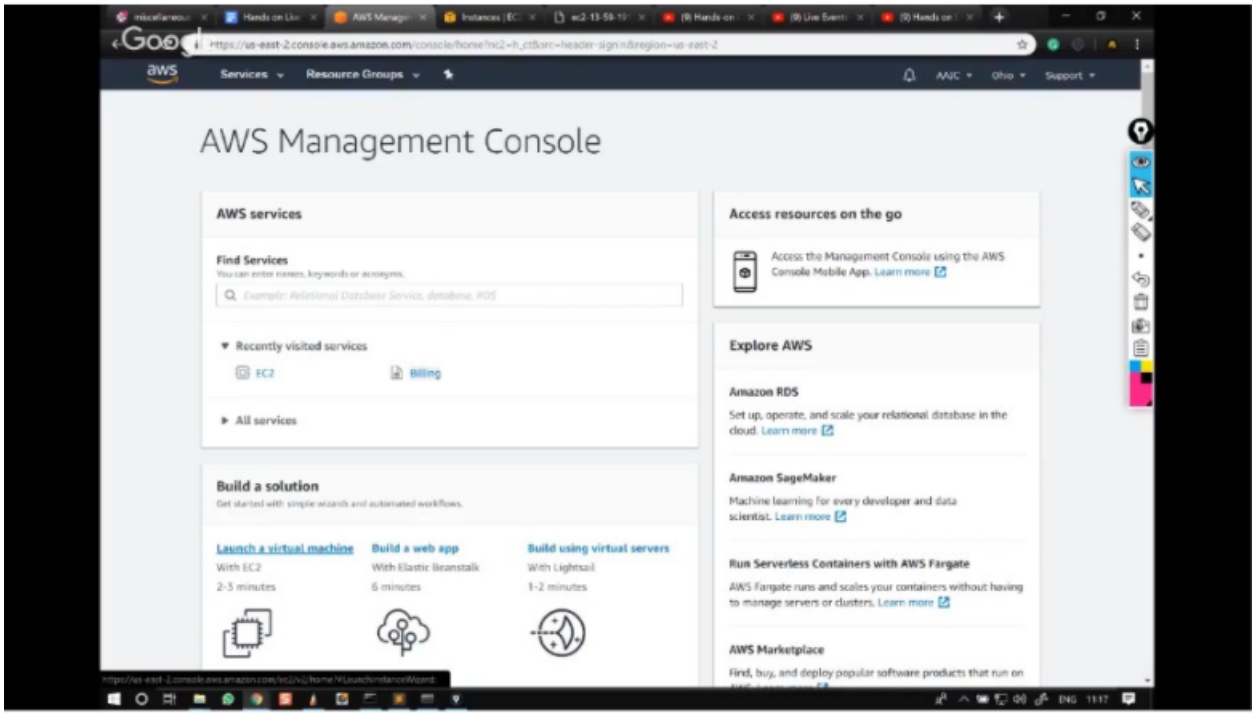
Creating an instance :

1.) Create an AWS account by referring to this
2.) Login using this link.



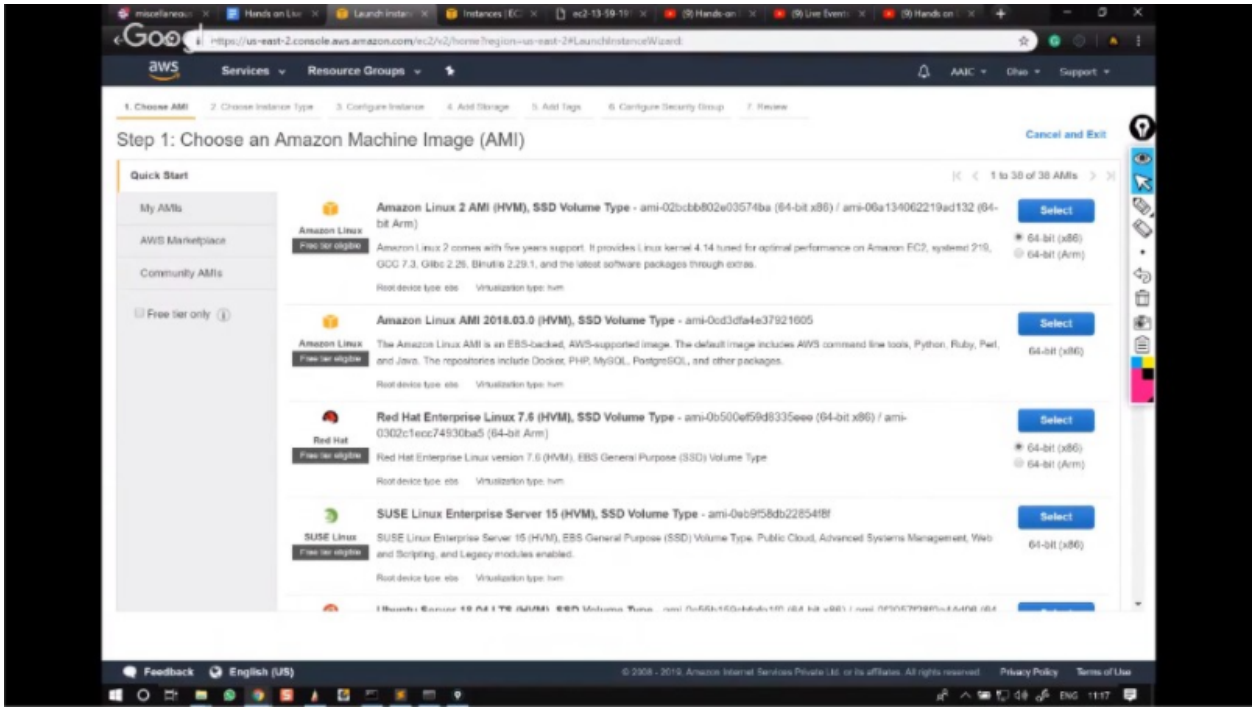
Timestamp : 01:19:40

3.) After we login, a console page is displayed.



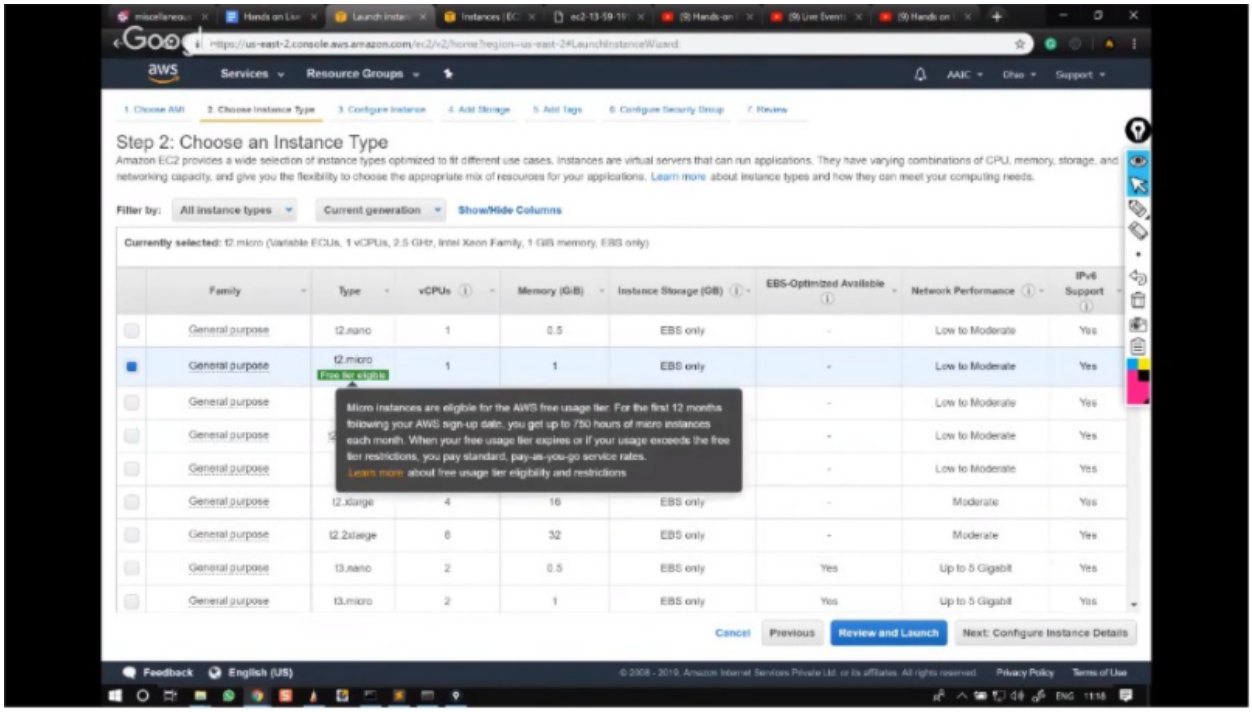
Timestamp : 01:20:17

4.) Click on Launch a virtual machine.



Timestamp : 01:20:29

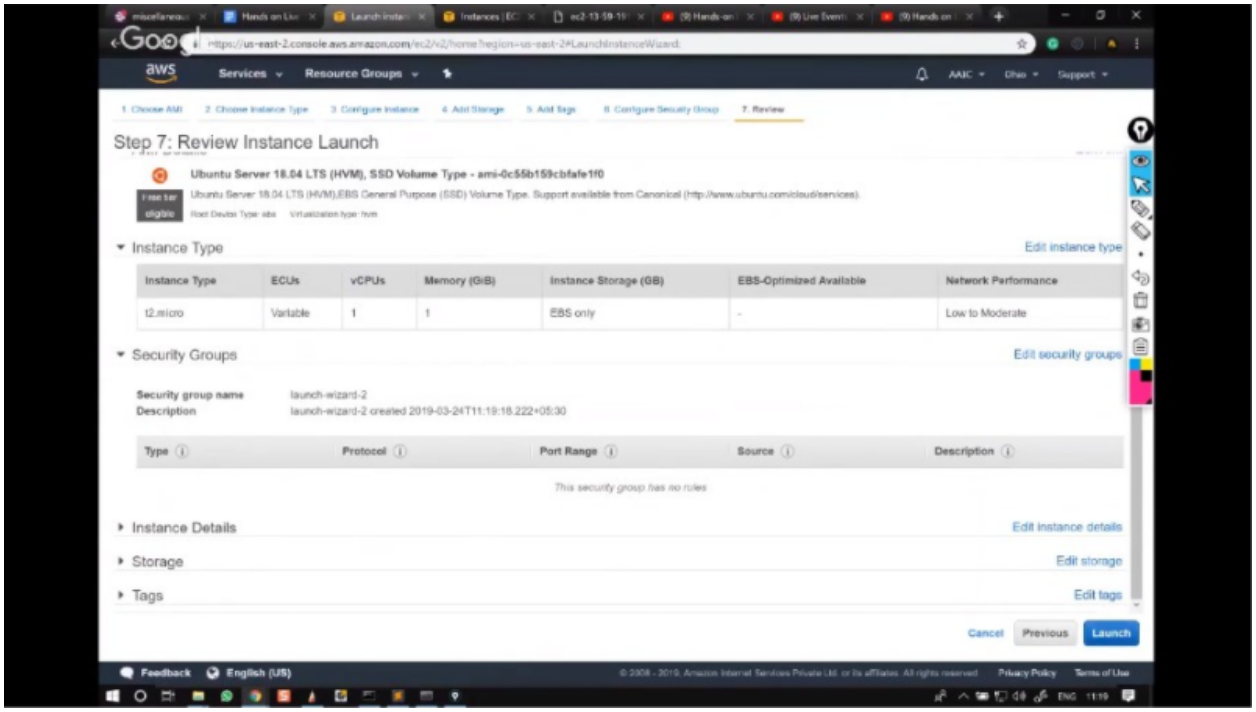
5.) Please select the Ubuntu Server as it's free and also it's a micro instance. This comes under free-tier.



Timestamp : 01:21:35

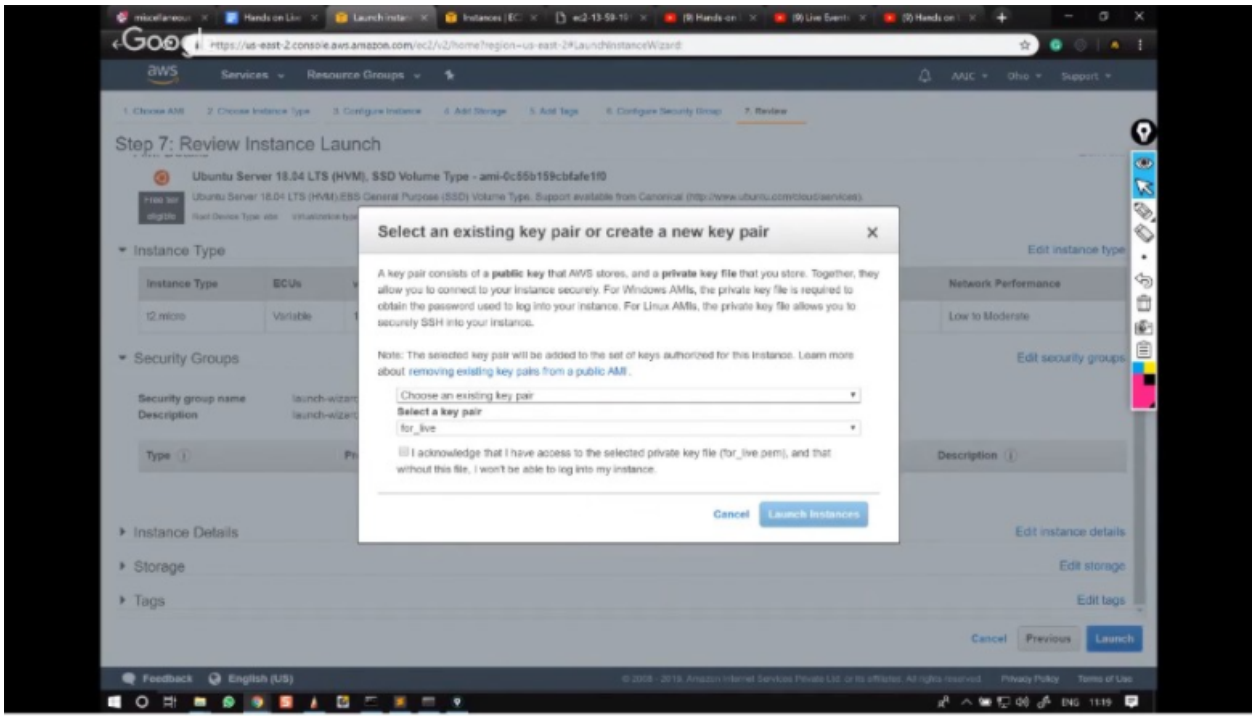
6.) We are going to select the one which is highlighted in the above image.

7.) Click on Review and Launch.



Timestamp : 01:22:31

8.) Then click on Launch.



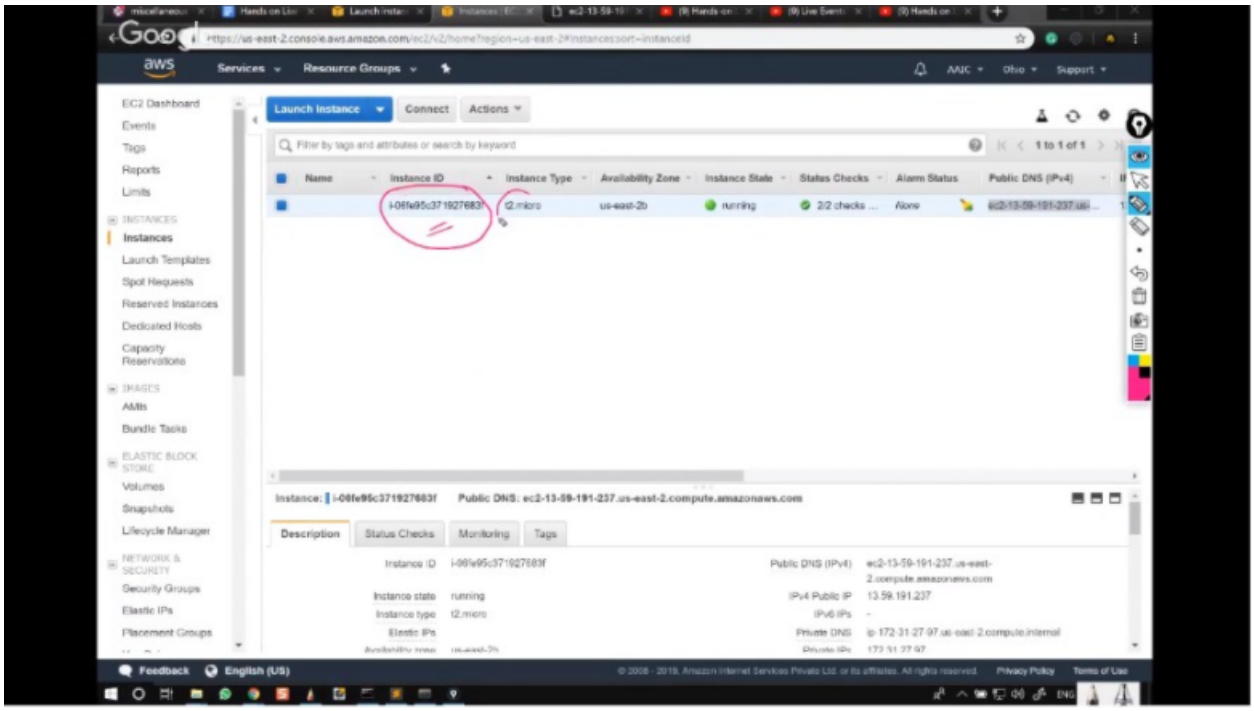
Timestamp : 01:22:35

9.) Once you click on Launch, the above page is displayed.

Let’s understand key-pair.

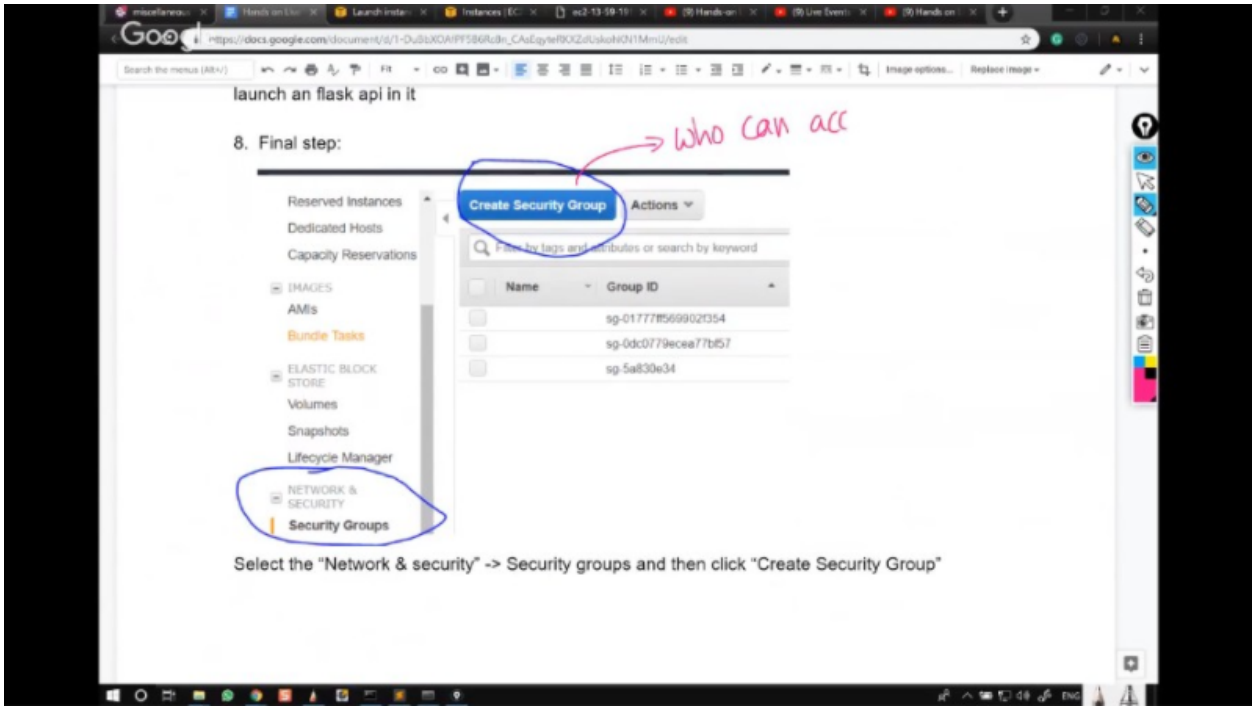
Key-pair -> Encrypted key to safely login to your remote box.

10.) Then select Create a new key-pair from the drop-down menu.
11.) You will get a key file with extension .pem
12.) Once you have downloaded the file, then select the key file.
13.) Then click on Launch instances.



Timestamp : 01:25:17

14.) We can see the Public DNS of our instance which we can use to access our instance.



Timestamp : 01:28:24

15.) Finally, we can create a security group. It will provide some settings where we can provide access to certain users and other things. For now, we can leave it with the default settings.

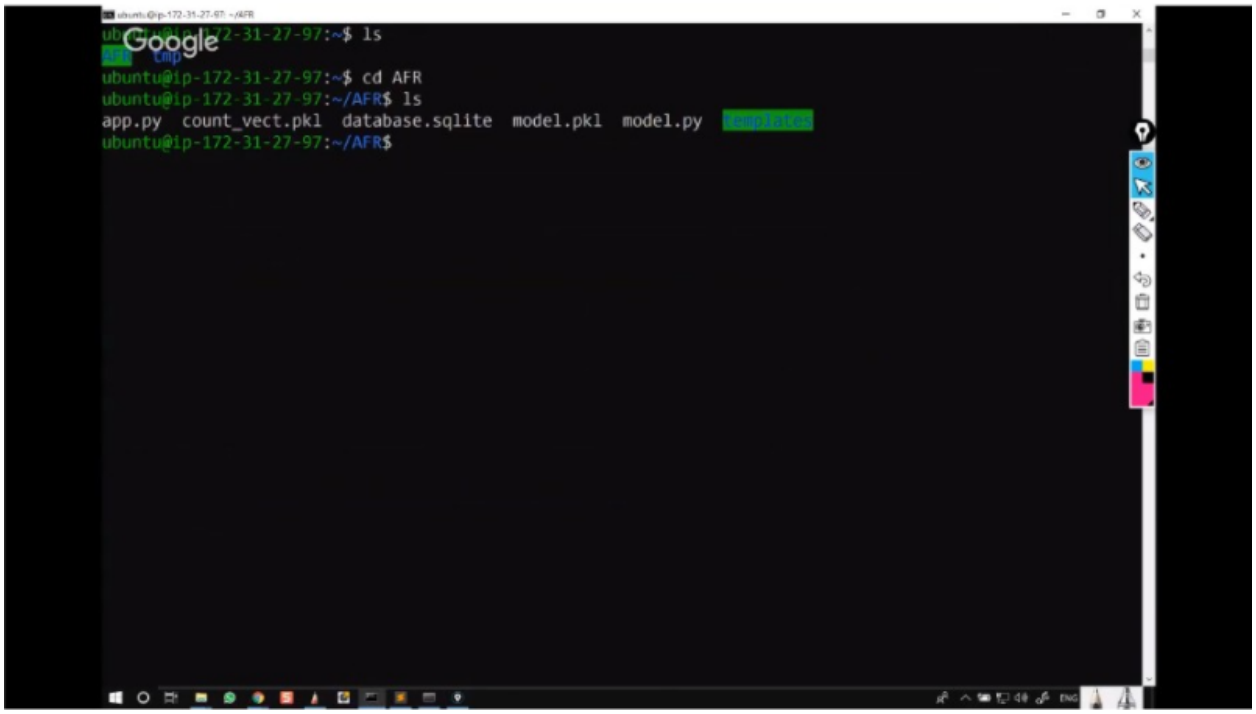
Connecting to the AWS instance

1.) Go to the command prompt by searching for command prompt in the start menu.
2.) Go to the directory where the key-pair file lives using cd command.
3.) Then type the command ssh -i "key-pair_name.pem" ubuntu@ec2-13-59-191-237.us-east-2.compute.amazonaws.com

Ssh means secure shell. With this , we can connect to any other device. You have to provide your key-pair name in place of key-pair_name.

4.) After you press enter, we will get connected to the instance.
5.) First, we need to copy the code related files to the AWS instance so that we can run from there.

6.) For that, we can use the scp command.
7.) Type scp -r -i “key-pair_name.pem” ./AFT ubuntu@ec2-13-59-191-237.us-east-2.compute.amazonaws.com:~/ (scp - secure copy -r recursive (it will recursively copy all the files in a folder))
8.) Then, go to the AFR folder by typing the command cd AFR.
9.) Check whether the files are properly copied by listing out the folder using ls command.



Timestamp : 01:38:05

We can see from the above image that all files are copied correctly.

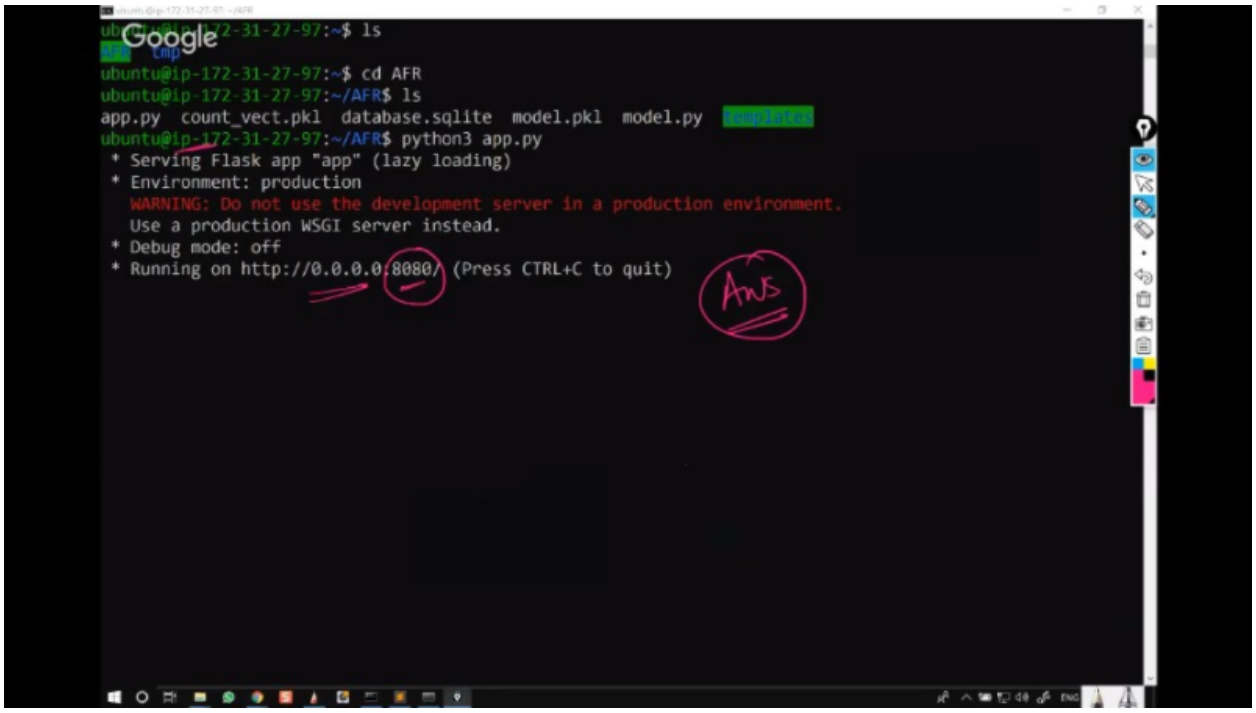
Install all packages needed on the AWS box.

1.) Type sudo apt-get install python3-pip. (It will install the pip).
2.) Then type pip3 install <>
Where <> is replaced by the following packages

1.) pip3

2.) pandas
3.) numpy
4.) sklearn
5.) beautifulsoup4
6.) lxml
7.) flask
8.) Re

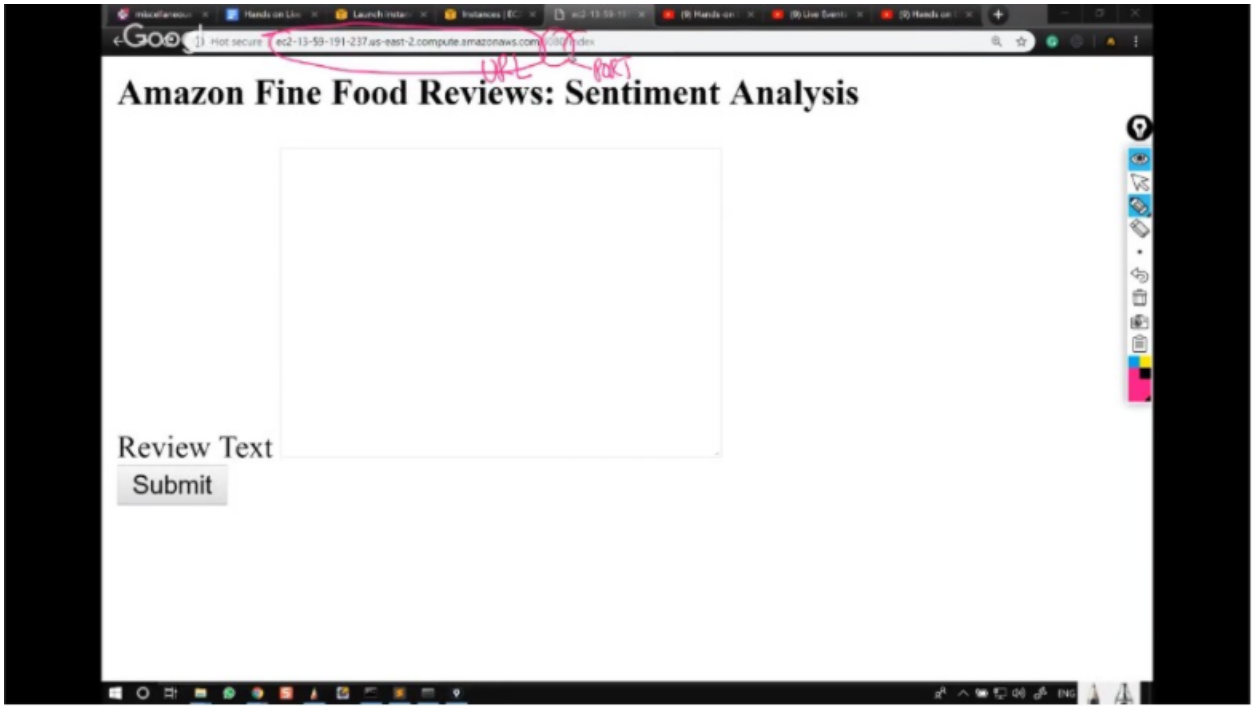
Then type python3 app.py. This will run the app.py as we did before.



Timestamp : 01:40:41

Then go to the web browser and type ec2-13-59-191-237.us-east-2.compute.amazonaws.com:8080/index in the address bar.

Then the following web page gets loaded.



Timestamp : 01:41:15

If we write some text and click on the Submit button, the predict function gets called and we get the output.

47.3 DataScience webapps using Streamlit - Part 1

Showcasing your work on machine learning/deep learning is as important as building a model. There are many approaches to it. Some of them are as follows :

1.) **Sharing the Jupyter/Colab notebook.** This approach is not widely used. The reason being is that a customer or end user can't understand the code behind a model. They are not required to know about it. Only programmers can understand it. So, it's used within the programmers community.
2.) **Writing a blog.** This approach is widely adopted in practice. People who can understand the language can easily understand our work. Usually blogs contain illustrations,code walkthrough,references and many more. To be simple, it's a multimedia version of our work.
3.) **Building a web application.** This is a nice approach enjoyed by the end users. This is because an end user can communicate with our work and understand it through some simple UI and other controls. But building a web application requires some work. There are many languages and frameworks one must be familiar with to build web applications. We already discussed productionizing ml models as a web app using flask and ec2.

Streamlit :

Streamlit is an open-source web app framework for machine learning and data science teams. Why do we stress upon machine learning and data science teams ? The reason is that most ml/dl experts are not aware about web technologies and they are primarily interested in building models. To make their life simpler, streamlit was introduced. Using python as a language and streamlit framework, we can build web applications easily.

The session is splitted into two parts:

1.) We cover data visualization by building a simple web application.
2.) Deep learning and streamlit.

First, we will spend some time discussing streamlit.

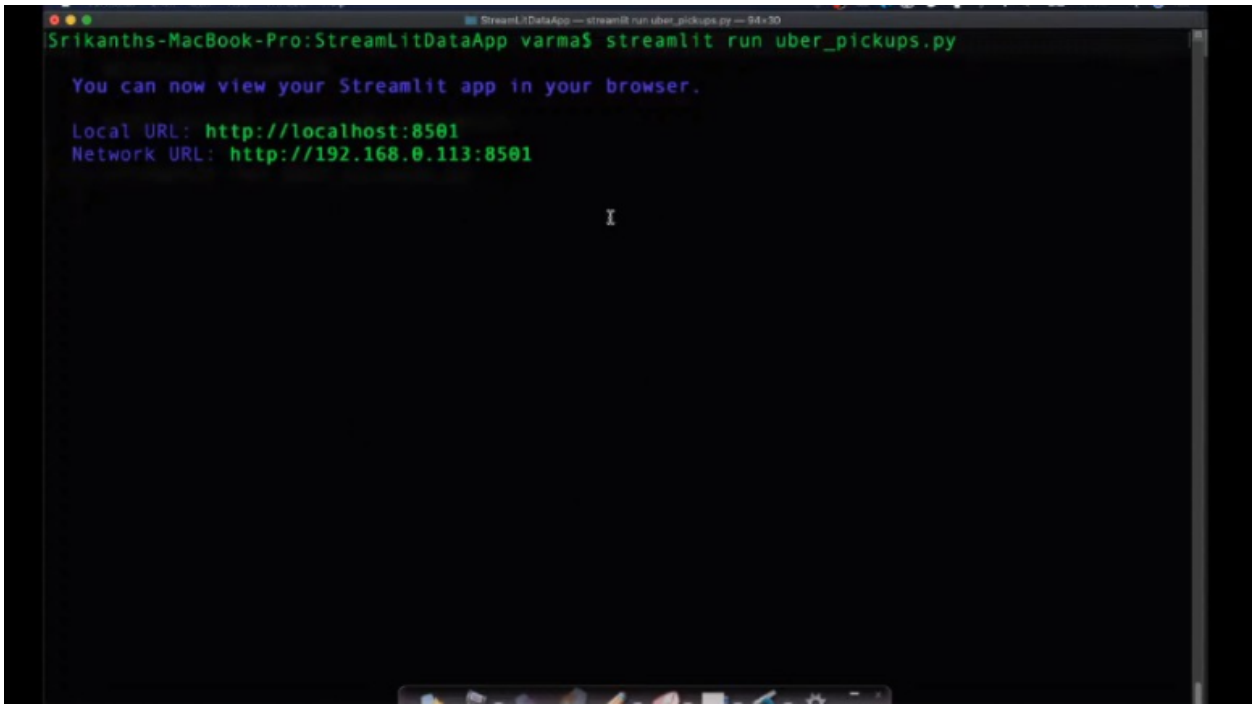
How to install streamlit in our local system ?

Type **pip3 install --upgrade streamlit** in the command prompt/power shell/anaconda command prompt and press the Enter key.

How to run a streamlit web application ?

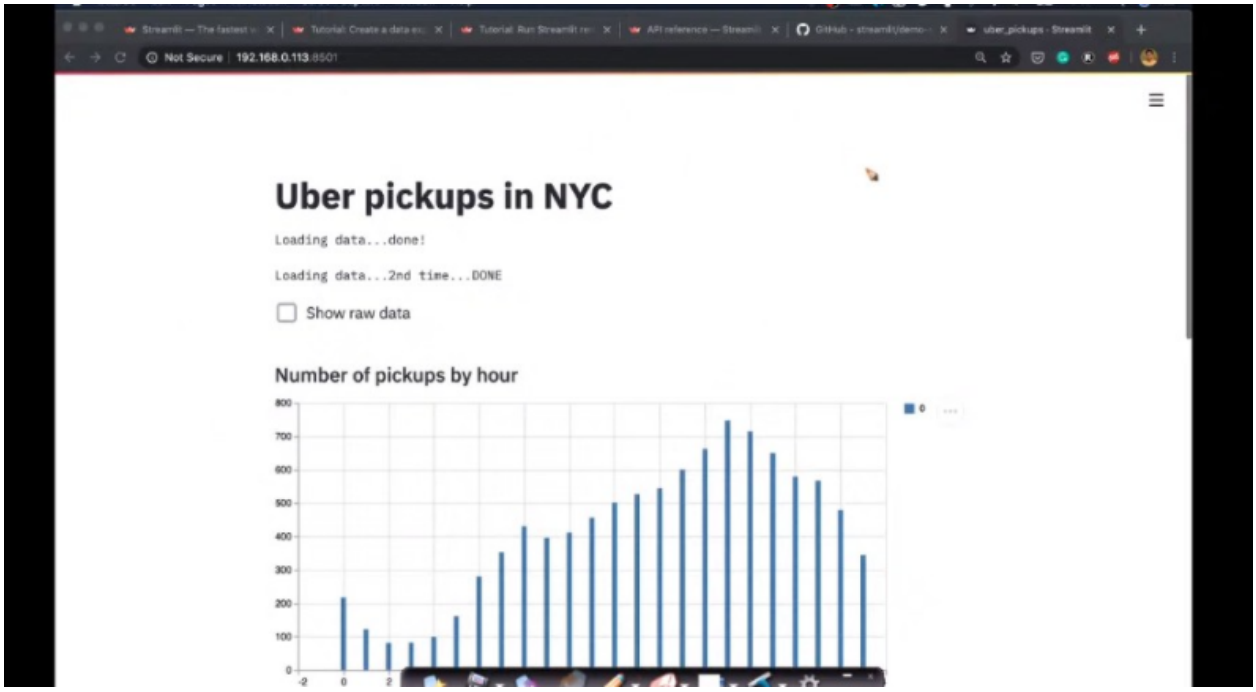
Type **streamlit run file_name.py** where file_name is the name of the file name and press the Enter key.

Once you press the Enter key, the web application is served in our local system on the default port if not specified. Now, we can copy the link from the Local URL tab and paste it in our web browser.



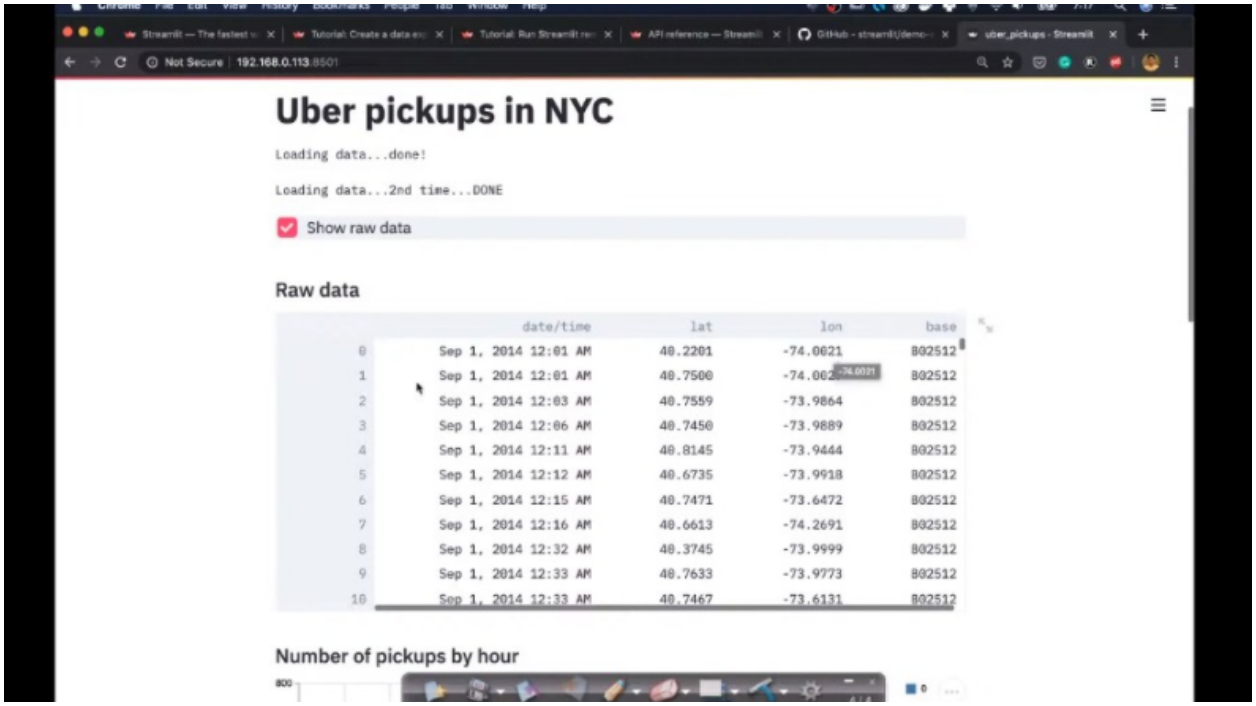
Timestamp : 18:06

By default, streamlit uses 8501 as the port number. Port number simply is the address of our web application residing in our computer memory.



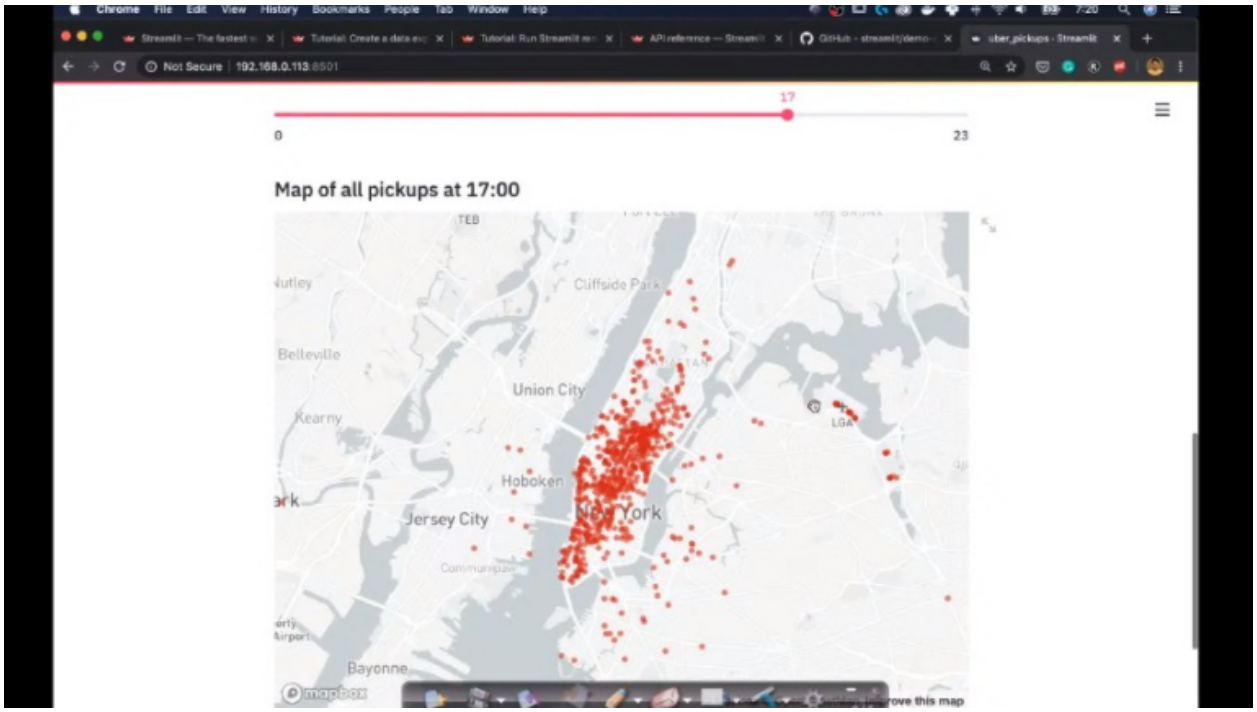
Timestamp : 20:43

This is not only a static interface. You can communicate with it. For example, you can press the checkbox button named “Show raw data” and it will dynamically alter the content in our web application.

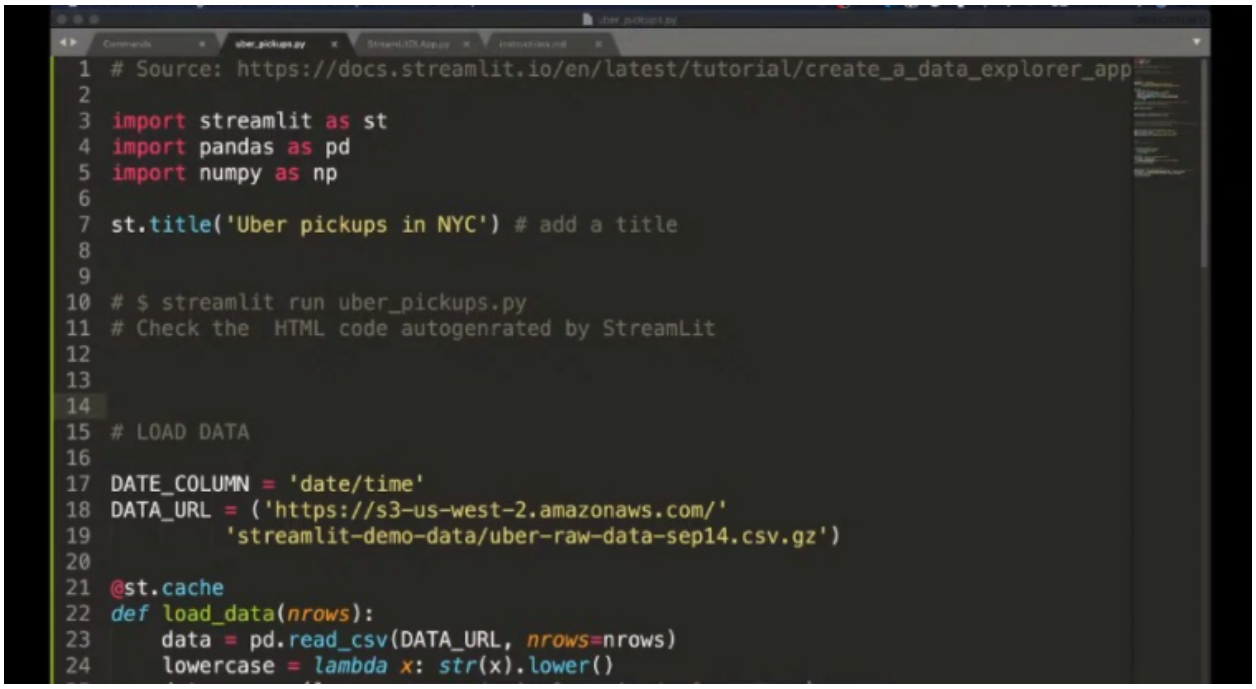


Timestamp : 21:30

We can plot graphs,histograms,maps,etc.



Timestamp : 23:45



```python
# Source: https://docs.streamlit.io/en/latest/tutorial/create_a_data_explorer_app

import streamlit as st
import pandas as pd
import numpy as np

st.title('Uber pickups in NYC') # add a title


# $ streamlit run uber_pickups.py
# Check the  HTML code autogenrated by StreamLit



# LOAD DATA

DATE_COLUMN = 'date/time'
DATA_URL = ('https://s3-us-west-2.amazonaws.com/'
         'streamlit-demo-data/uber-raw-data-sep14.csv.gz')

@st.cache
def load_data(nrows):
    data = pd.read_csv(DATA_URL, nrows=nrows)
    lowercase = lambda x: str(x).lower()
```

Timestamp : 25:54

We use the keyword **import** to import all the required libraries.

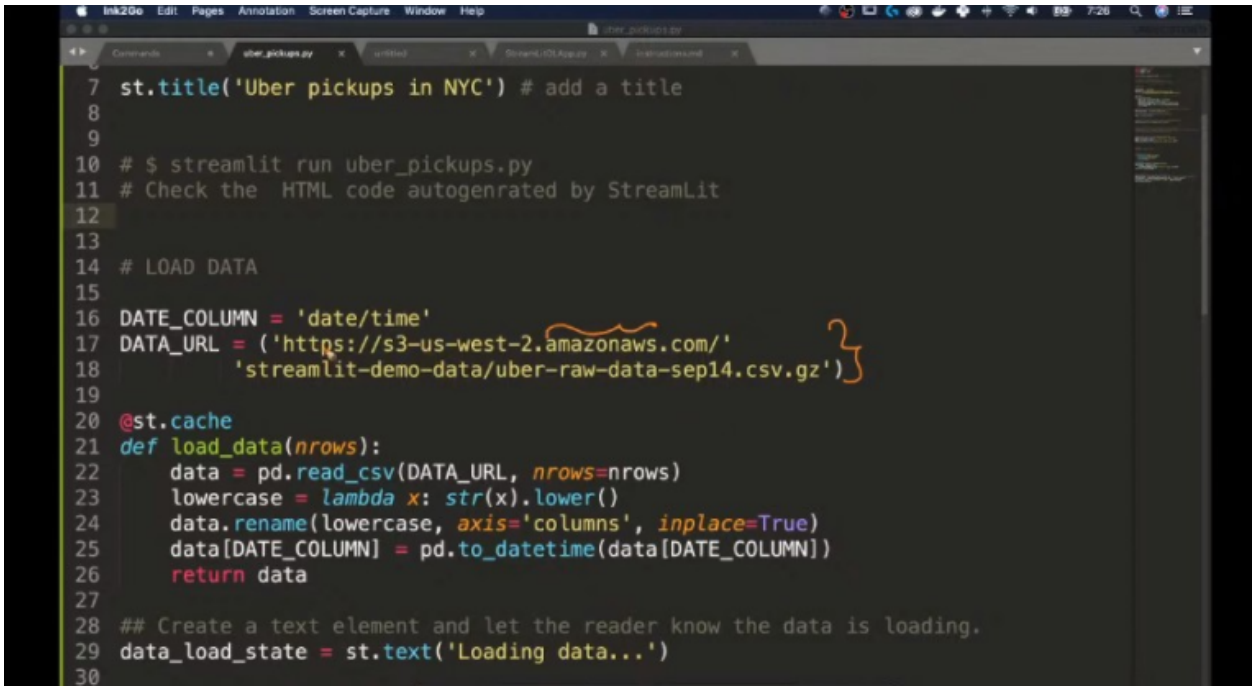
st.title(x) : This method is used to set the title for our web application. Here, the method takes a single parameter which is of string data type. Once we execute the function, the title of our web application is set to x.

Note : Whenever you make some changes to the code, you don't need to run the web application from the start. In your web application, you can see a button named Rerun in the top right corner. You can simply click on it. It will re-run it for you. This button will be visible if you make some modifications to the code.

Basically, when you execute the method st.title(x), in the backend it creates the code in html,javascript and other web frameworks. To check this, you can press right click and then click on Inspect. It will show you the code which is responsible for the web page. This will not be in python. As machine learning/deep learning experts are known to use python , streamlit is designed for it.

For data visualization, we need data right ?

For that, we are going to use the uber pickups data in New York City.



```python
st.title('Uber pickups in NYC') # add a title


# $ streamlit run uber_pickups.py
# Check the  HTML code autogenrated by StreamLit


# LOAD DATA

DATE_COLUMN = 'date/time'
DATA_URL = ('https://s3-us-west-2.amazonaws.com/'
         'streamlit-demo-data/uber-raw-data-sep14.csv.gz')

@st.cache
def load_data(nrows):
    data = pd.read_csv(DATA_URL, nrows=nrows)
    lowercase = lambda x: str(x).lower()
    data.rename(lowercase, axis='columns', inplace=True)
    data[DATE_COLUMN] = pd.to_datetime(data[DATE_COLUMN])
    return data

## Create a text element and let the reader know the data is loading.
data_load_state = st.text('Loading data...')

```

Timestamp : 29:59

The csv file is hosted on the amazon aws server.

The load_data function simply reads the data from the url specified in the DATA_URL variable.

Lowercase is a lambda function which converts the parameter to a string and then changes it to lower case.

Then, we are renaming the columns to lowercase.

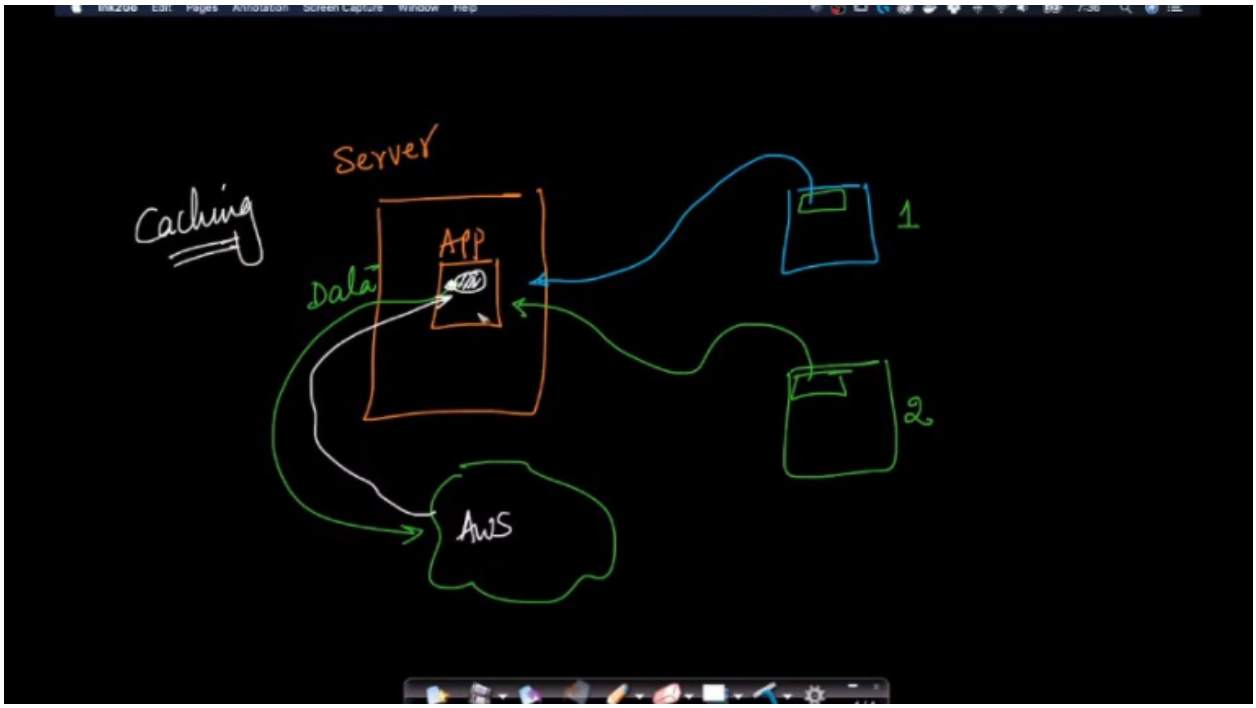
There is a date column in the data frame. We convert into pandas date time objects by using the method pd.to_datetime method. This is because we can manipulate it easily. For example, we can visualize dates in proper format.

st.text(x) : This simply puts a text in the web page with x being displayed. Here, x is a string.

The load_data function also takes in the parameter nrows which specifies how many records to be loaded from the csv file.

Let's learn about @st.cache which is a decorator in python language.

For this we will consider a situation where it really is used.



Timestamp : 39:51

1.) Let's say we have created a simple web application using a streamlit framework and hosted it on amazon aws server or any server.
2.) This web application is meant for downloading data from the web and then displaying it.
3.) Let's say we have two users u1 and u2 who are going to access this web application.

4.) Let's say at first, user u1 accesses it.
5.) The web application will simply download the data from the web and display it for user u1.
6.) Now, again if user u2 accesses it, then it agains downloads the same data and then displays it.
7.) There is no mistake in this case. But, what if the data is really huge ?
8.) Since both users are accessing the same data, why do we need to load it every time when a user needs it.
9.) One simple solution is to simply save the data in the memory when the user u1 accesses it.
10.) If the user u2 tries to access the same content, then we can simply load the data from the memory and then display it rather than downloading it and then displaying. This is nothing but caching. But it's restricted to our memory space.

Whenever we add a decorator to a function named @st.cache, these below mentioned things happen internally.

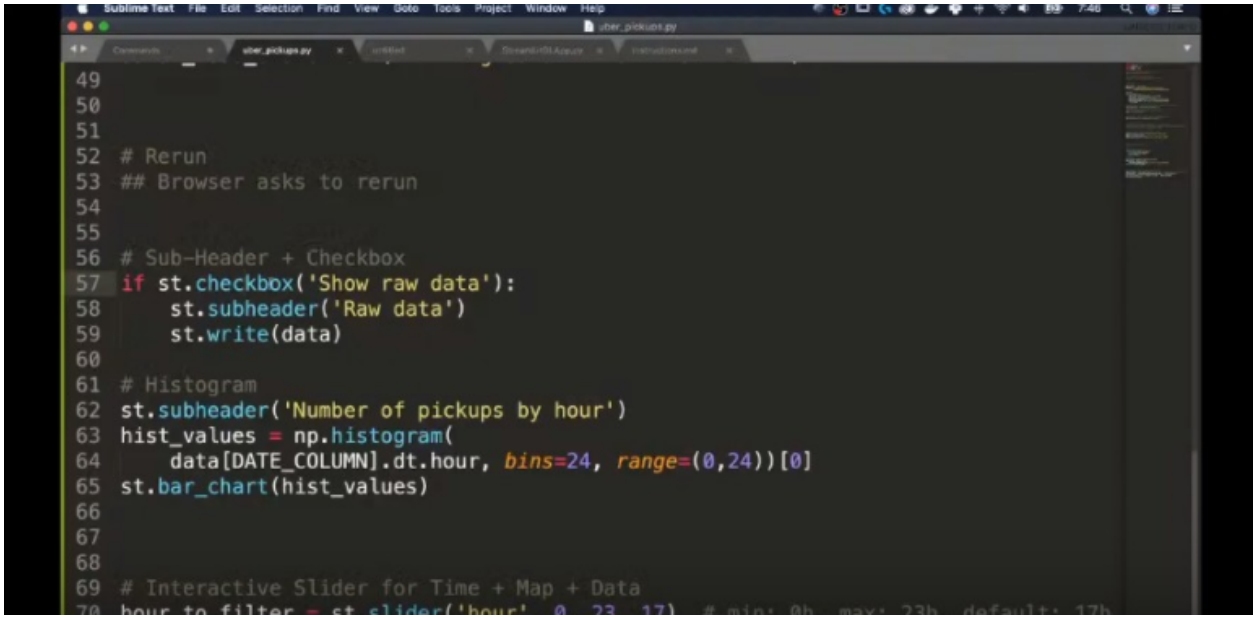
1.) When the function is executed, a dictionary is created which contains key and value pairs.
2.) The key contains the function name and the parameter. The value is the corresponding output of that function. So, if any where the same function is called with the same parameter, then we simply retrieve the value from the dictionary and then return it.

Some important points.

1.) If the bytecode of the function is changed , the entry from the dictionary is removed. For example, in future you may add some modifications to the existing function. The parameter may be the same but there are some lines added.
2.) It works only for deterministic and non-randomized functions. This is self-evident. If the function is itself not deterministic , then there is no need to cache it. Since, the output is going to be different every time you execute it.

Caching is nothing but memoization in the programming paradigm.

st.checkbox(x): It creates a checkbox with the text displayed on it as x.

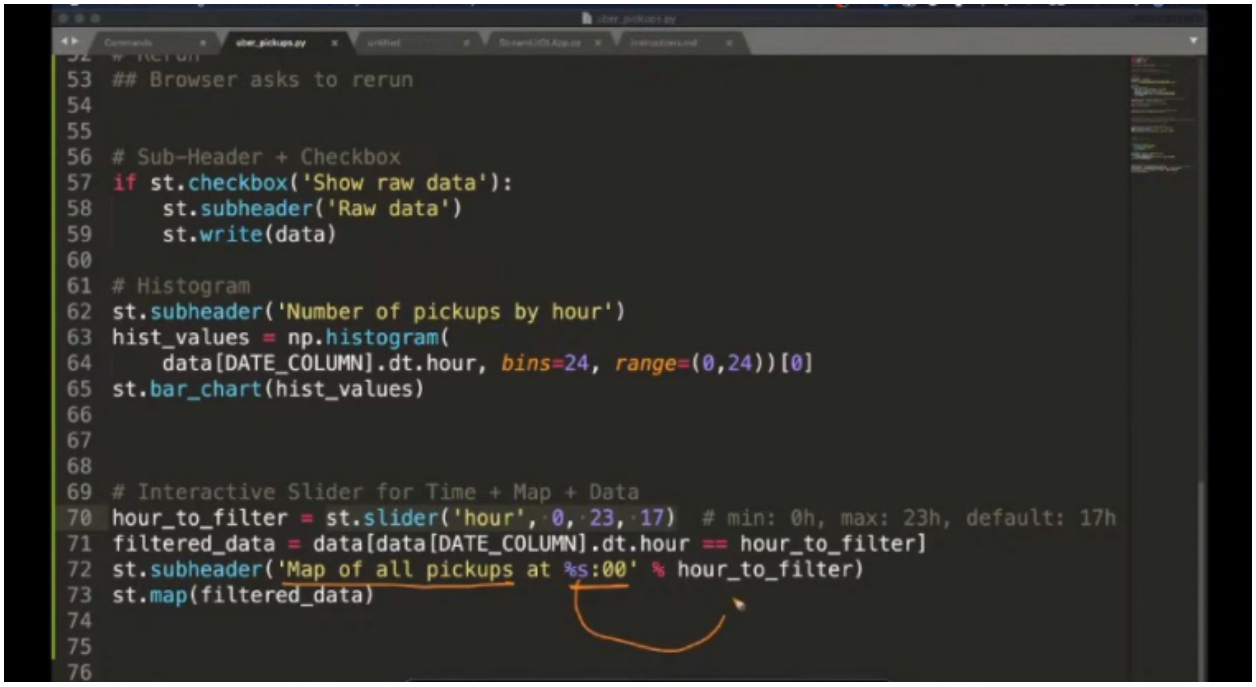


Timestamp : 49:53

The lines in the if block are executed only when the checkbox is clicked. When the checkbox is unchecked, the content is cleared.

st.write(object) : st.write method simply displays the object which is passed to it. For example, we can display a pandas dataframe by executing st.write(data) where data is the dataframe object.

st.bar_chart(x) : It plots a bar chart based on the histogram passed which is x. For this, we can use the histogram method available in the numpy library.

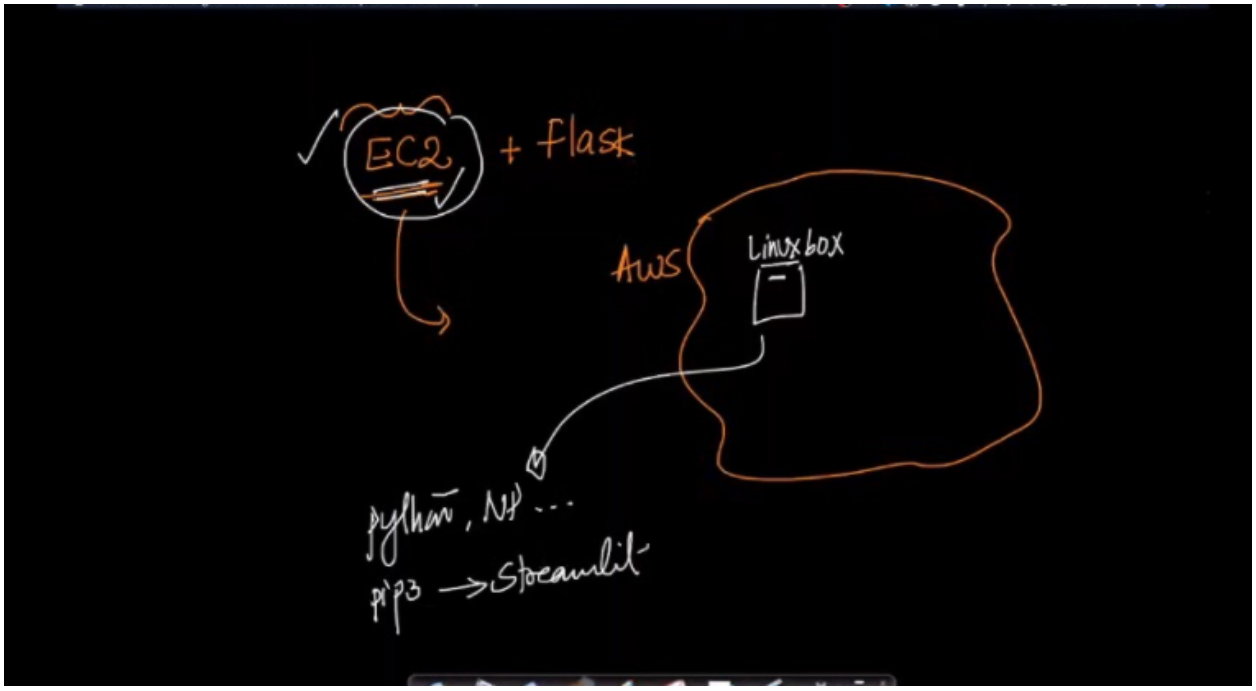


```
53 ## Browser asks to rerun
54
55
56 # Sub-Header + Checkbox
57 if st.checkbox('Show raw data'):
58     st.subheader('Raw data')
59     st.write(data)
60
61 # Histogram
62 st.subheader('Number of pickups by hour')
63 hist_values = np.histogram(
64     data[DATE_COLUMN].dt.hour, bins=24, range=(0,24))[0]
65 st.bar_chart(hist_values)
66
67
68
69 # Interactive Slider for Time + Map + Data
70 hour_to_filter = st.slider('hour', 0, 23, 17)  # min: 0h, max: 23h, default: 17h
71 filtered_data = data[data[DATE_COLUMN].dt.hour == hour_to_filter]
72 st.subheader('Map of all pickups at %s:00' % hour_to_filter)
73 st.map(filtered_data)
74
75
76
```

Timestamp : 55:50

We use the method slider from streamlit library which displays a slider where the user can drag the slider for choosing a value. The first argument is the label of the slider. The second is the starting range. Third one is the end of the range. Finally, the last argument specifies the default. We are filtering the data and then displaying it as a map.

Till now, we have discussed the functionalities delivered by the streamlit library and how to run it on our local machine. Next, we will look at how to deploy this web application on a server like AWS.



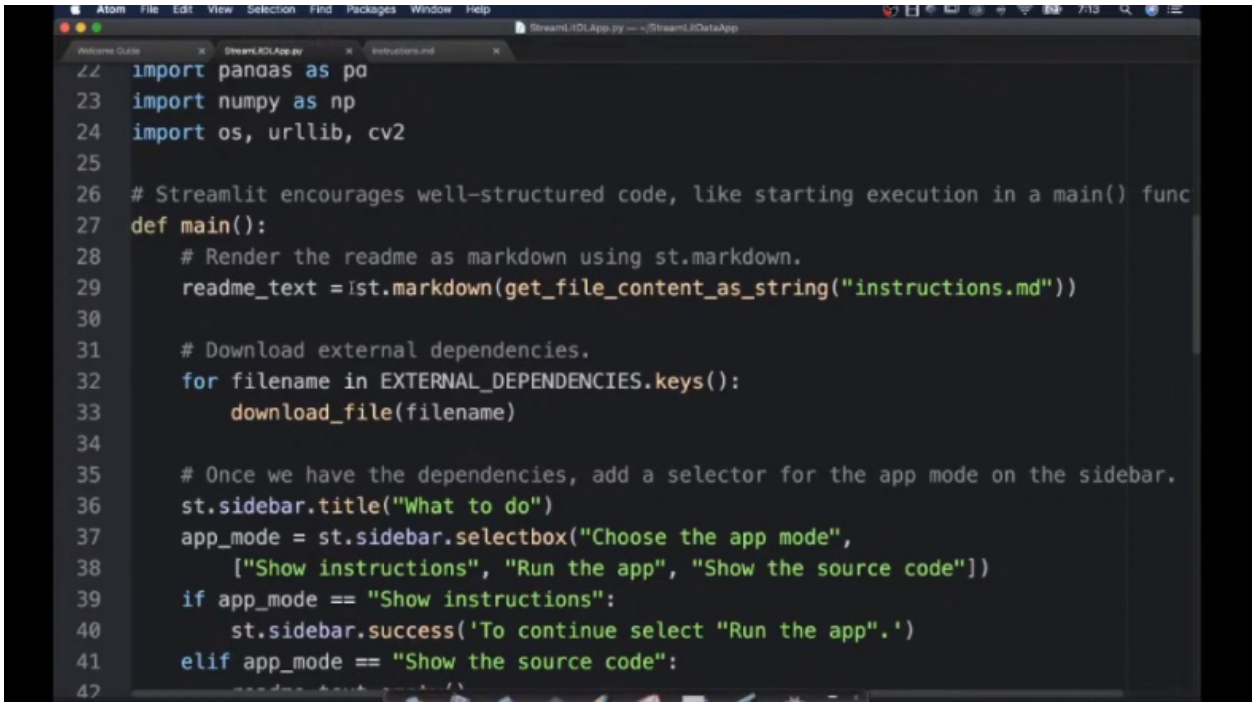
Timestamp : 59:36

The above illustration explains the process in a simpler way.

There are many functionalities in streamlit library. We can always refer to the documentation for the same.

47.4 DataScience webapps using Streamlit - Part 2

Let’s dive into the code.

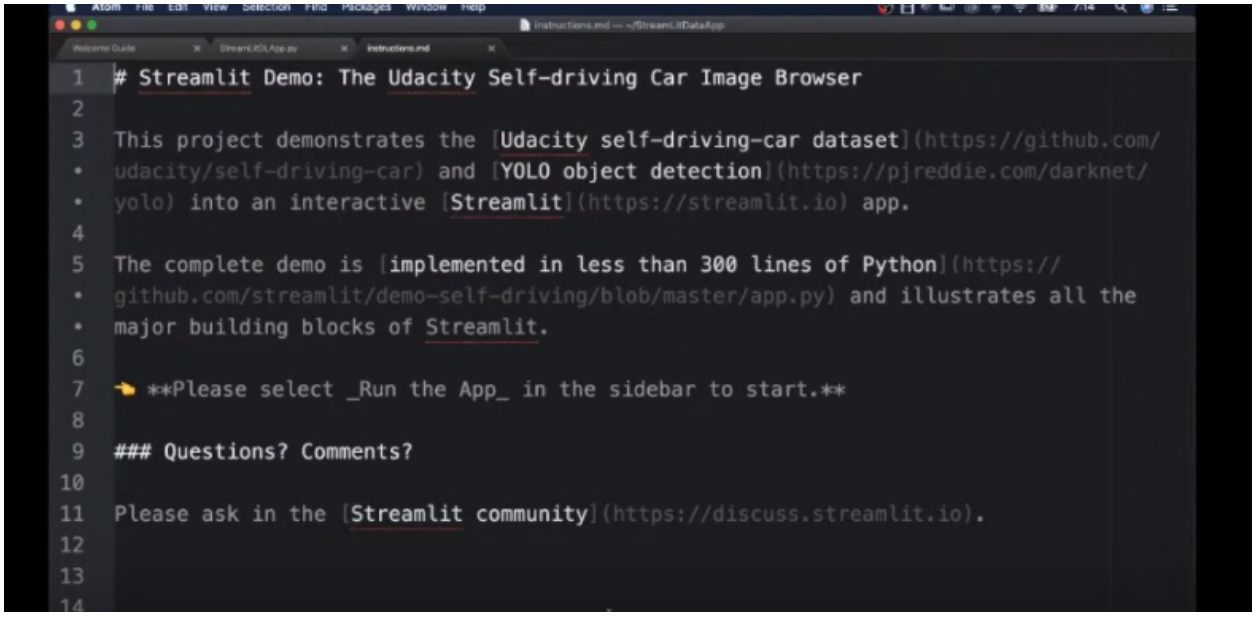


```
import pandas as pd
import numpy as np
import os, urllib, cv2

# Streamlit encourages well-structured code, like starting execution in a main() func
def main():
    # Render the readme as markdown using st.markdown.
    readme_text = st.markdown(get_file_content_as_string("instructions.md"))

    # Download external dependencies.
    for filename in EXTERNAL_DEPENDENCIES.keys():
        download_file(filename)

    # Once we have the dependencies, add a selector for the app mode on the sidebar.
    st.sidebar.title("What to do")
    app_mode = st.sidebar.selectbox("Choose the app mode",
        ["Show instructions", "Run the app", "Show the source code"])
    if app_mode == "Show instructions":
        st.sidebar.success('To continue select "Run the app".')
    elif app_mode == "Show the source code":
```

Timestamp : 15:26

Everything starts with the main function. Let’s look at the content.

First, we imported a bunch of libraries. Some of the key libraries are streamlit and cv2. Streamlit is responsible for building web applications and cv2 is responsible for building computer vision applications.

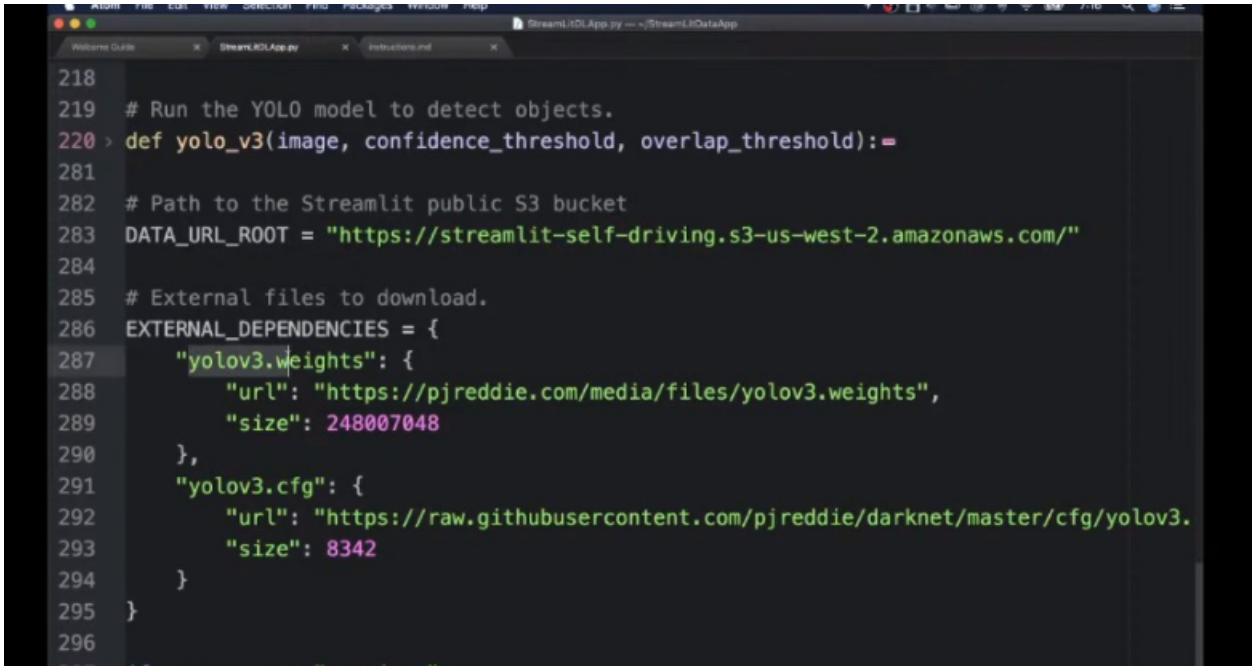


Timestamp : 16:00

The above is the instructions.md file. It contains some instructions needed for this web application. Our task is to simply render the content in this file into our web application.

readme_text = st.markdown(get_file_content_as_string(“instructions.md”)) -> This code simply takes the file “instructions.md” and parses the string in it. Then, it will render the content on the web page as a markdown.

EXTERNAL_DEPENDENCIES :



```
218
219  # Run the YOLO model to detect objects.
220  def yolo_v3(image, confidence_threshold, overlap_threshold):
281
282  # Path to the Streamlit public S3 bucket
283  DATA_URL_ROOT = "https://streamlit-self-driving.s3-us-west-2.amazonaws.com/"
284
285  # External files to download.
286  EXTERNAL_DEPENDENCIES = {
287      "yolov3.weights": {
288          "url": "https://pjreddie.com/media/files/yolov3.weights",
289          "size": 248007048
290      },
291      "yolov3.cfg": {
292          "url": "https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.
293          "size": 8342
294      }
295  }
296
```

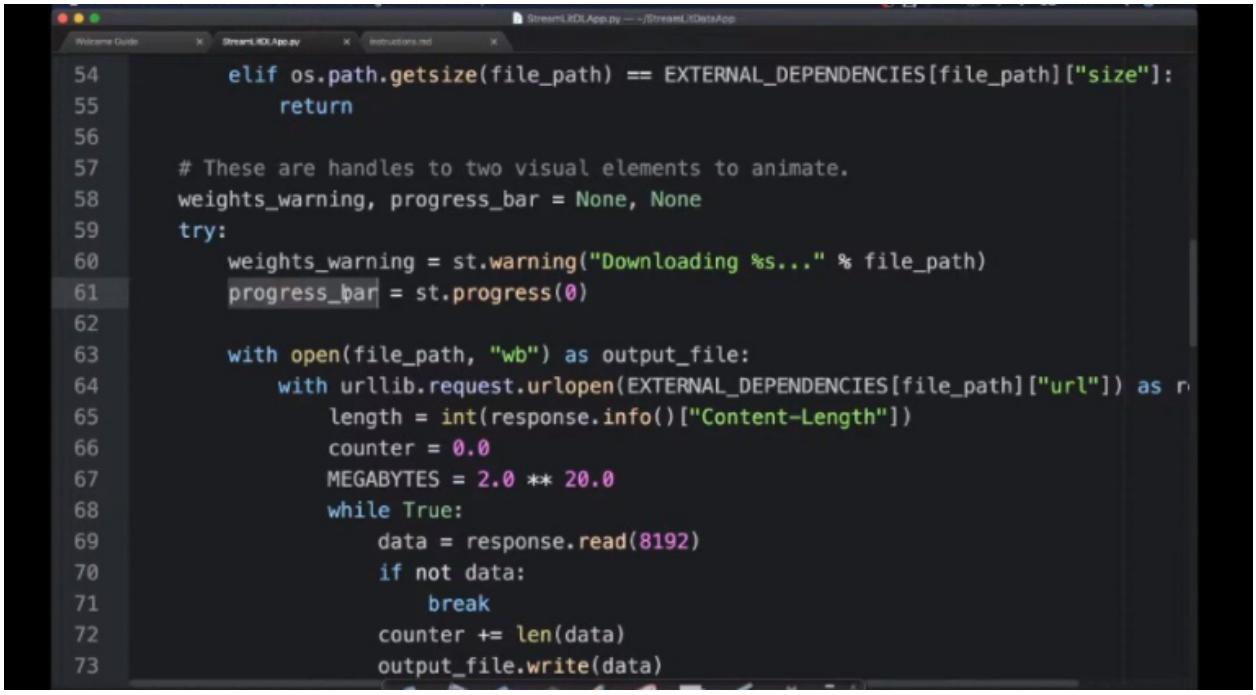
Timestamp : 18:11

We are creating a dictionary where the keys are the entity we need. The value represents the description of retrieving it.

Here, we can get the weights of the yolov3 model from the url specified where the size represents the memory space occupied by the weights.

Similarly, the same goes for the yolov3.cfg which is the configuration file. This contains the schema of the yolo v3 model.

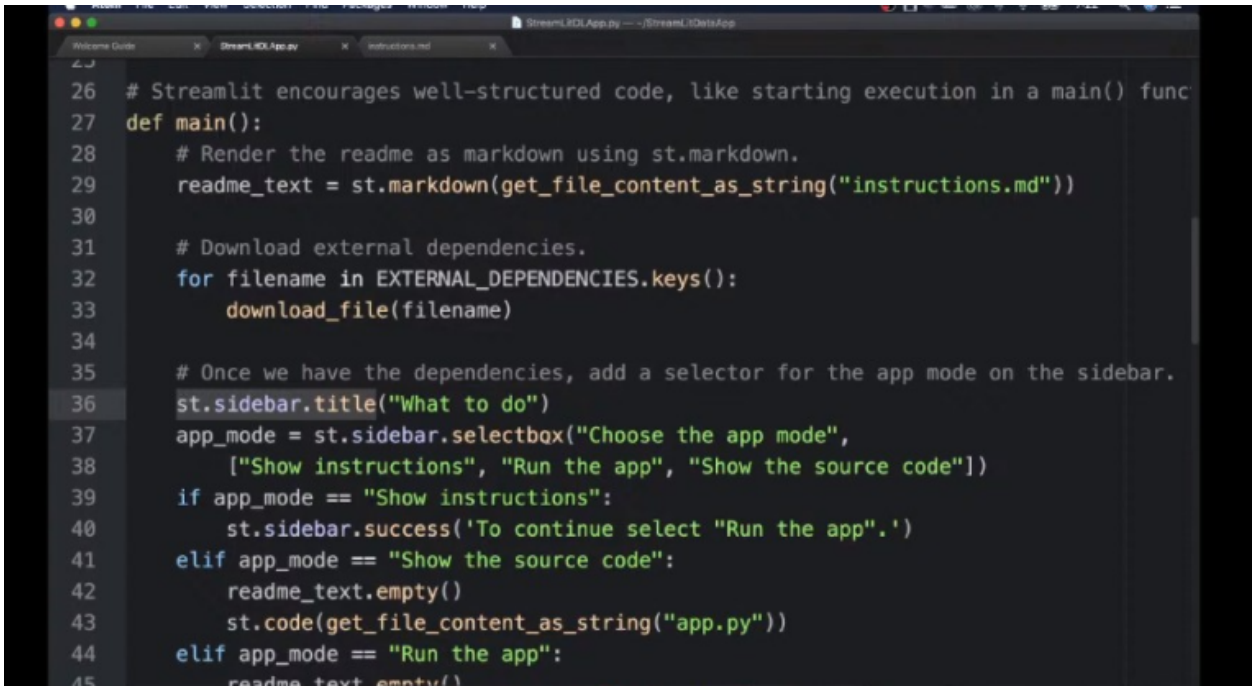
In the download_file function, we first check whether the file has already been downloaded. If not, we try to download the files. By using the urllib library and the functions available in it, we are parsing the url and downloading it.



```
        elif os.path.getsize(file_path) == EXTERNAL_DEPENDENCIES[file_path]["size"]:
            return

    # These are handles to two visual elements to animate.
    weights_warning, progress_bar = None, None
    try:
        weights_warning = st.warning("Downloading %s..." % file_path)
        progress_bar = st.progress(0)

        with open(file_path, "wb") as output_file:
            with urllib.request.urlopen(EXTERNAL_DEPENDENCIES[file_path]["url"]) as r
                length = int(response.info()["Content-Length"])
                counter = 0.0
                MEGABYTES = 2.0 ** 20.0
                while True:
                    data = response.read(8192)
                    if not data:
                        break
                    counter += len(data)
                    output_file.write(data)
```

Timestamp : 21:01

We are also computing how many bytes of data we are downloading. In case of any errors like network connectivity, we simply empty the progress bar and raise an error message.

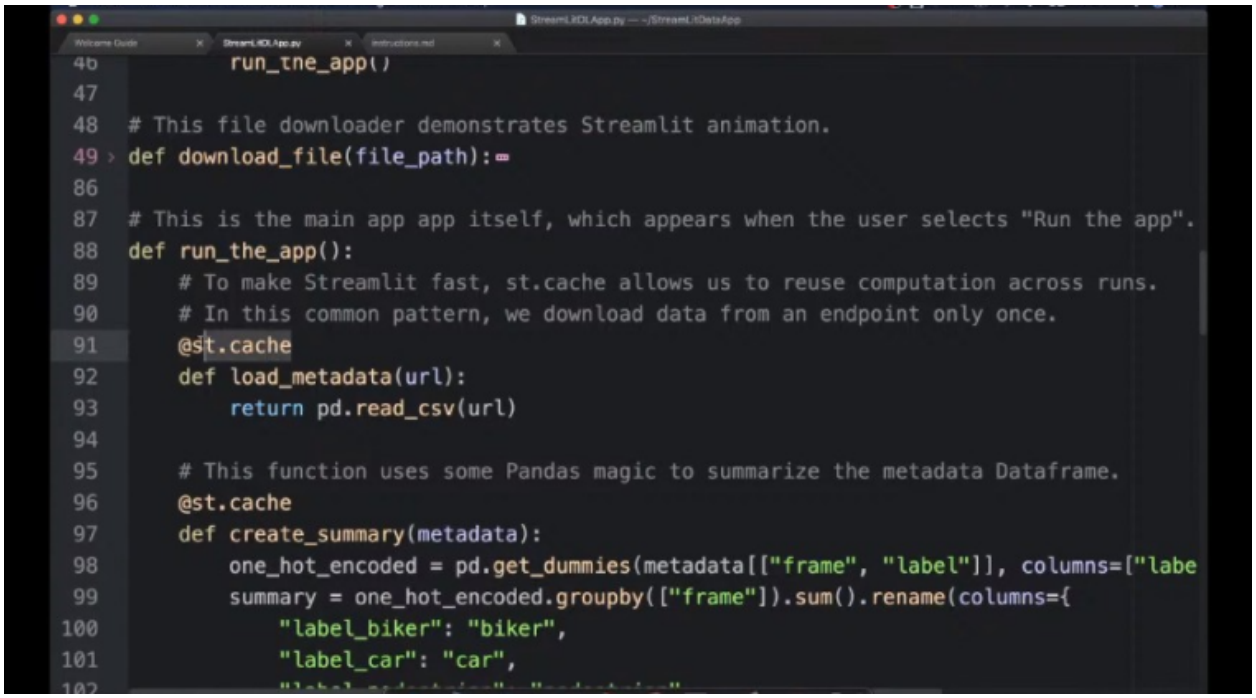


```
# Streamlit encourages well-structured code, like starting execution in a main() func
def main():
    # Render the readme as markdown using st.markdown.
    readme_text = st.markdown(get_file_content_as_string("instructions.md"))

    # Download external dependencies.
    for filename in EXTERNAL_DEPENDENCIES.keys():
        download_file(filename)

    # Once we have the dependencies, add a selector for the app mode on the sidebar.
    st.sidebar.title("What to do")
    app_mode = st.sidebar.selectbox("Choose the app mode",
        ["Show instructions", "Run the app", "Show the source code"])
    if app_mode == "Show instructions":
        st.sidebar.success('To continue select "Run the app".')
    elif app_mode == "Show the source code":
        readme_text.empty()
        st.code(get_file_content_as_string("app.py"))
    elif app_mode == "Run the app":
```

Timestamp : 24:31

We are basically creating a sidebar with title and a select box. This select box contains some options where a user can click and select. Based on the selected option, we perform certain operations. The operations we are going to perform are clearly presented in the above image.

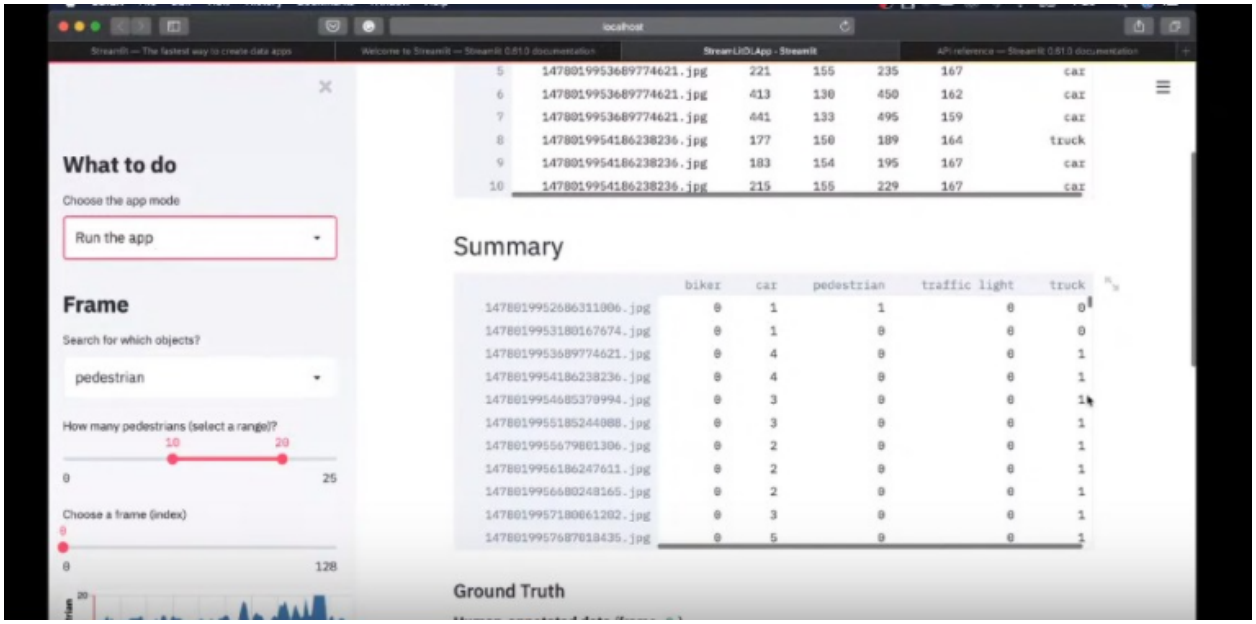
empty() method simply clears the content from the screen.



Timestamp : 27:23

We are using the @st.cache decorator here. We already discussed this functionality in the previous part. The load_metadata function simply reads the data from the url.

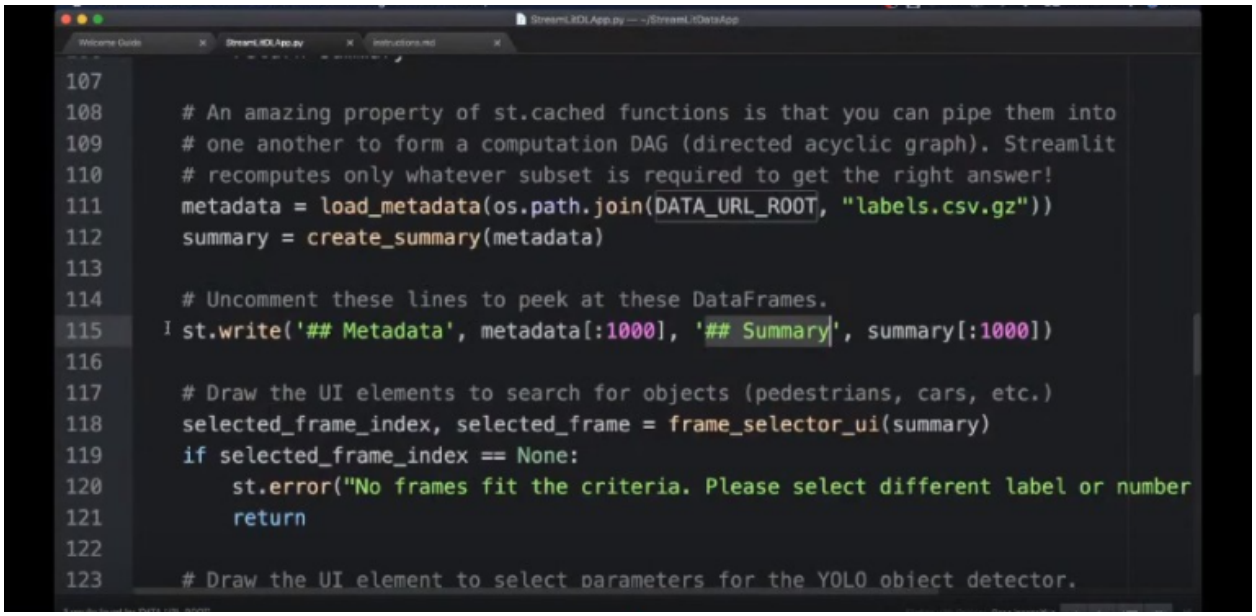
In the create_summary function, we first perform a one-hot encoding on the frame and label. We finally compute the number of labels for each image and then display it.



Timestamp : 28:27

We also print the coordinates of the bounding boxes. It's specified by xmin,ymin,xmax,ymax.

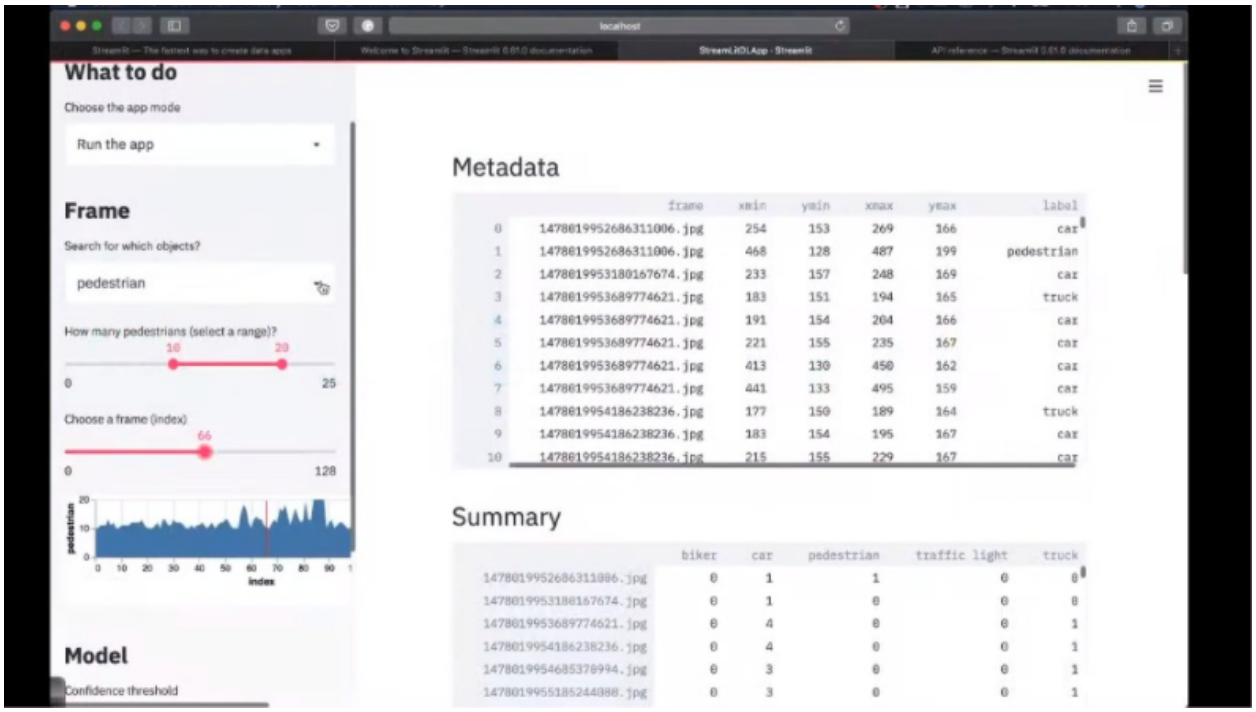
st.write function writes the data. Note : We can pass as many data frames we want and label it correspondingly.



Timestamp : 32:53

frame_selector_ui :

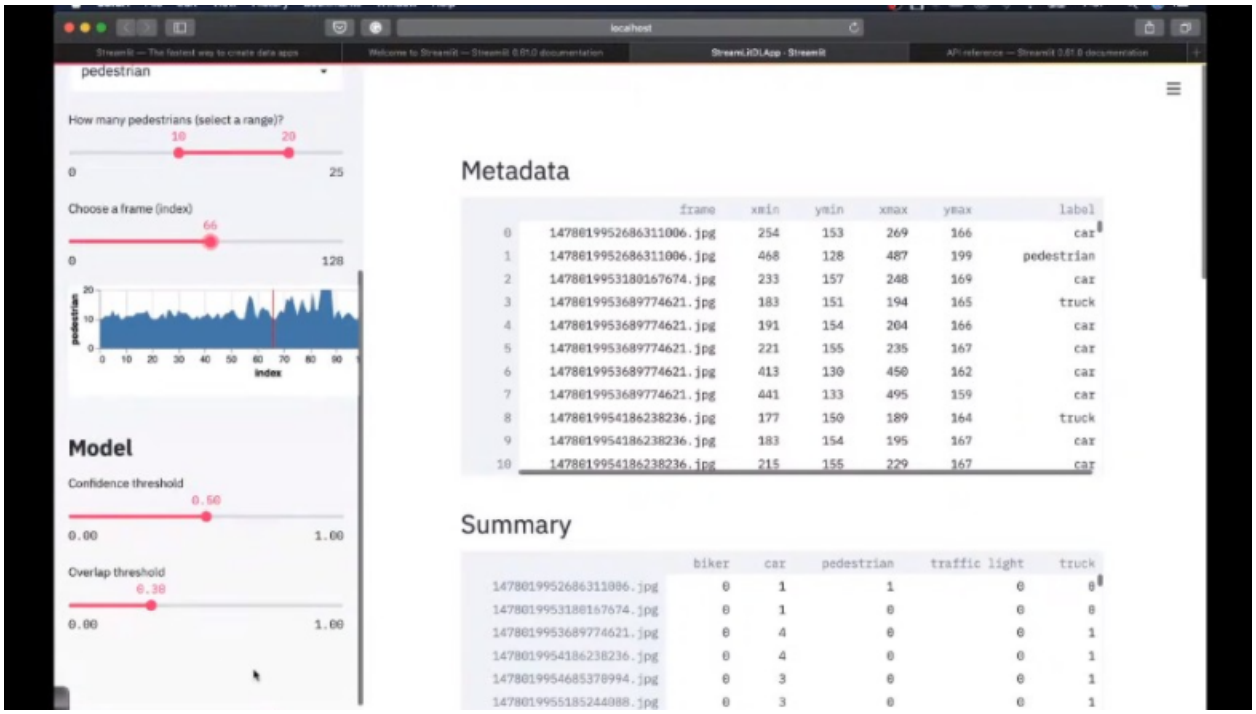
1.) First, we will create a sidebar with a markdown named Frame using the st.sidebar.markdown method.
2.) We create a select box which holds all the classes as options. This is done using st.sidebar.selectbox.
3.) Then, we create a slider which is used to select the range based on the object we chose. This is done using the st.sidebar.slider method.
4.) get_selected_frames : This function takes summary,object_type,min_elts and max_elts as the parameters and returns all the frames for which the former conditions are satisfied. If there are no frames available , then we return None.
5.) Now, we will create a slider from which we can choose the frame index based on the selected frames.



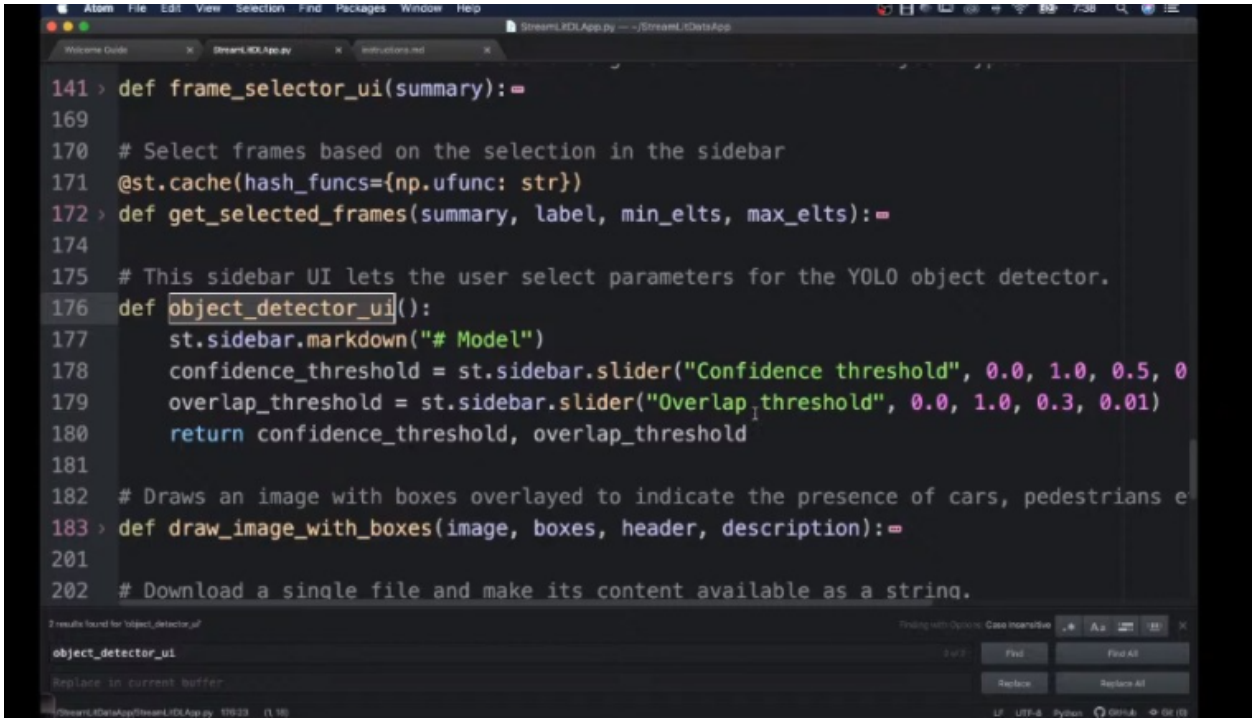
Timestamp : 37:39

Next, we plotted the bar chart of the number of objects which have been selected by the user vs the index.

object_detector_ui () : This function creates the sidebar and other things which are responsible for the model. These functionalities remain the same as discussed previously.

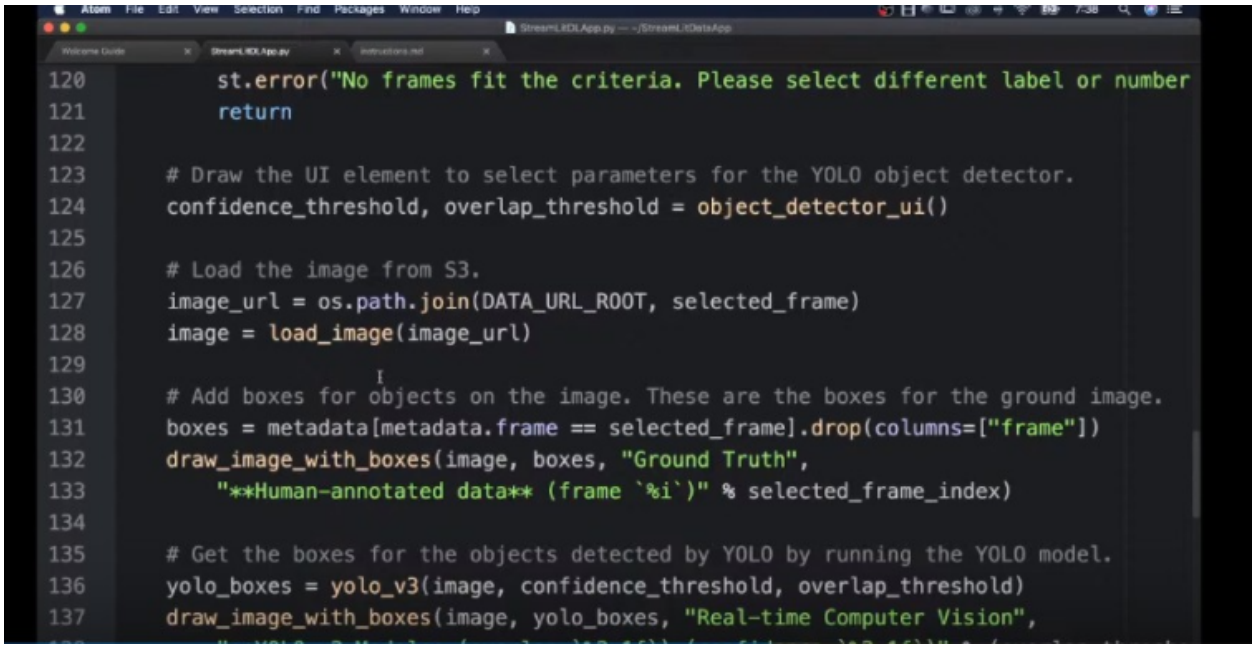


Timestamp : 39:48



```
141 › def frame_selector_ui(summary):⋯

169

170  # Select frames based on the selection in the sidebar
171  @st.cache(hash_funcs={np.ufunc: str})
172 › def get_selected_frames(summary, label, min_elts, max_elts):⋯

174

175  # This sidebar UI lets the user select parameters for the YOLO object detector.
176  def object_detector_ui():
177      st.sidebar.markdown("# Model")
178      confidence_threshold = st.sidebar.slider("Confidence threshold", 0.0, 1.0, 0.5, 0
179      overlap_threshold = st.sidebar.slider("Overlap threshold", 0.0, 1.0, 0.3, 0.01)
180      return confidence_threshold, overlap_threshold
181
182  # Draws an image with boxes overlayed to indicate the presence of cars, pedestrians e
183 › def draw_image_with_boxes(image, boxes, header, description):⋯

201

202  # Download a single file and make its content available as a string.
```

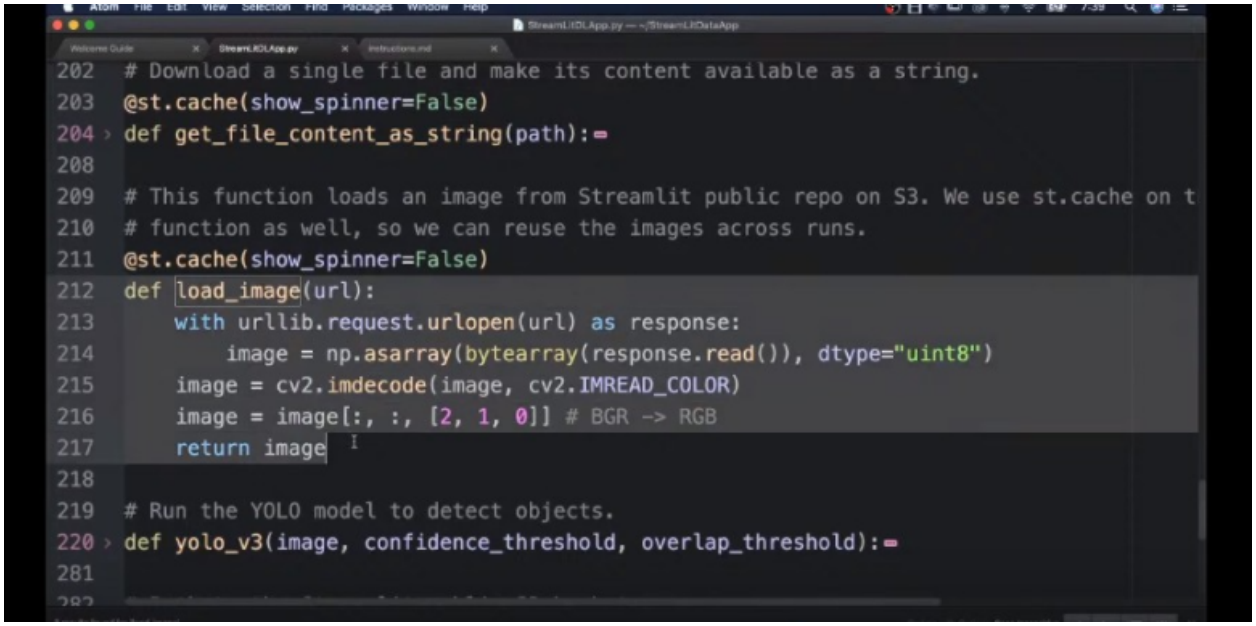
Timestamp : 40:05



```
120          st.error("No frames fit the criteria. Please select different label or number
121          return
122
123      # Draw the UI element to select parameters for the YOLO object detector.
124      confidence_threshold, overlap_threshold = object_detector_ui()
125
126      # Load the image from S3.
127      image_url = os.path.join(DATA_URL_ROOT, selected_frame)
128      image = load_image(image_url)
129
130      # Add boxes for objects on the image. These are the boxes for the ground image.
131      boxes = metadata[metadata.frame == selected_frame].drop(columns=["frame"])
132      draw_image_with_boxes(image, boxes, "Ground Truth",
133          "**Human-annotated data** (frame `%i`)" % selected_frame_index)
134
135      # Get the boxes for the objects detected by YOLO by running the YOLO model.
136      yolo_boxes = yolo_v3(image, confidence_threshold, overlap_threshold)
137      draw_image_with_boxes(image, yolo_boxes, "Real-time Computer Vision",
```

Timestamp : 40:46

First, we will load the image specified by the image_url. This is selected by the user.



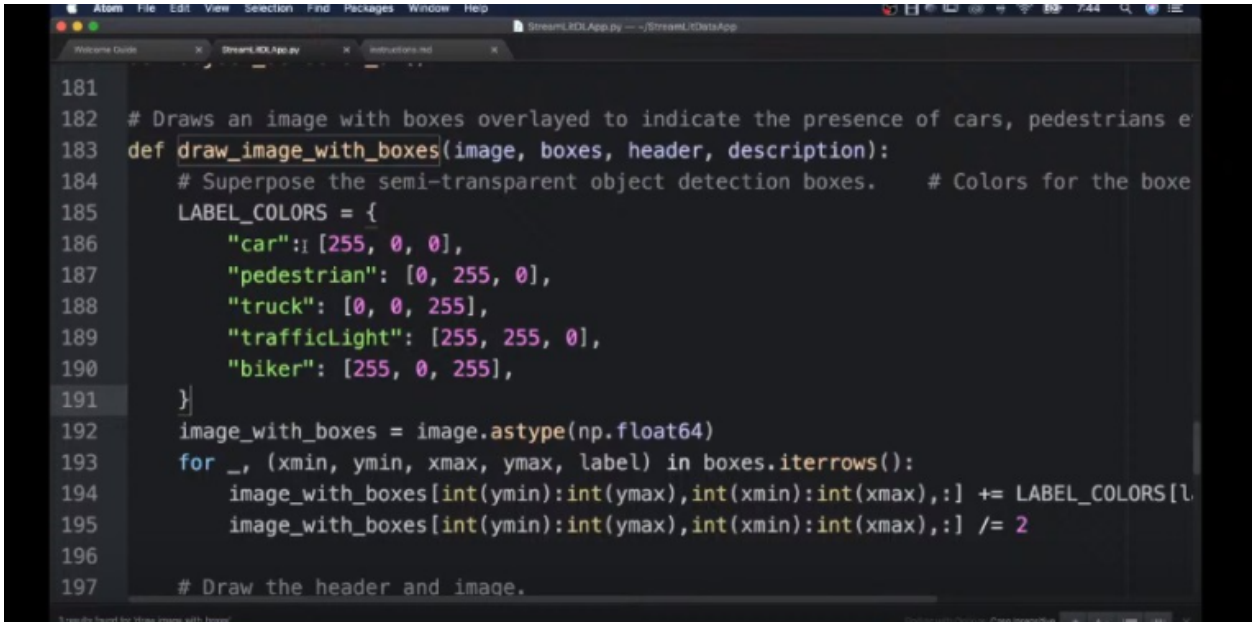
Timestamp : 41:50

Using the urllib library, we try to open the url. Then we read the response as an numpy array by using the np.asarray method. Then, we decode the array as an image using the imdecode method from the open cv library. We also convert the image to RGB scale.

Now, refer to the timestamp 40:46.

We get the information of the bounding boxes from the dataframe.

Then, we draw the bounding boxes on the image using the draw_image_with_boxes function.

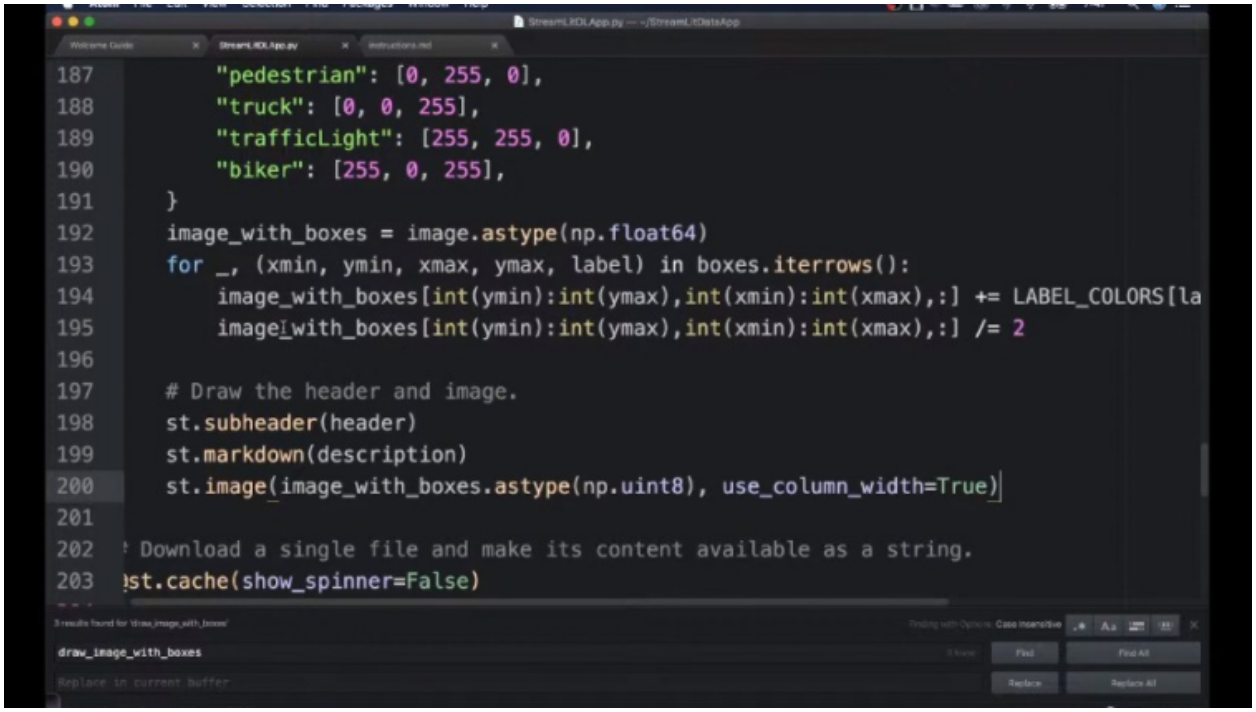


```
181
182 # Draws an image with boxes overlayed to indicate the presence of cars, pedestrians e
183 def draw_image_with_boxes(image, boxes, header, description):
184     # Superpose the semi-transparent object detection boxes.    # Colors for the boxe
185     LABEL_COLORS = {
186         "car": [255, 0, 0],
187         "pedestrian": [0, 255, 0],
188         "truck": [0, 0, 255],
189         "trafficLight": [255, 255, 0],
190         "biker": [255, 0, 255],
191     }
192     image_with_boxes = image.astype(np.float64)
193     for _, (xmin, ymin, xmax, ymax, label) in boxes.iterrows():
194         image_with_boxes[int(ymin):int(ymax),int(xmin):int(xmax),:] += LABEL_COLORS[l
195         image_with_boxes[int(ymin):int(ymax),int(xmin):int(xmax),:] /= 2
196
197     # Draw the header and image.
```

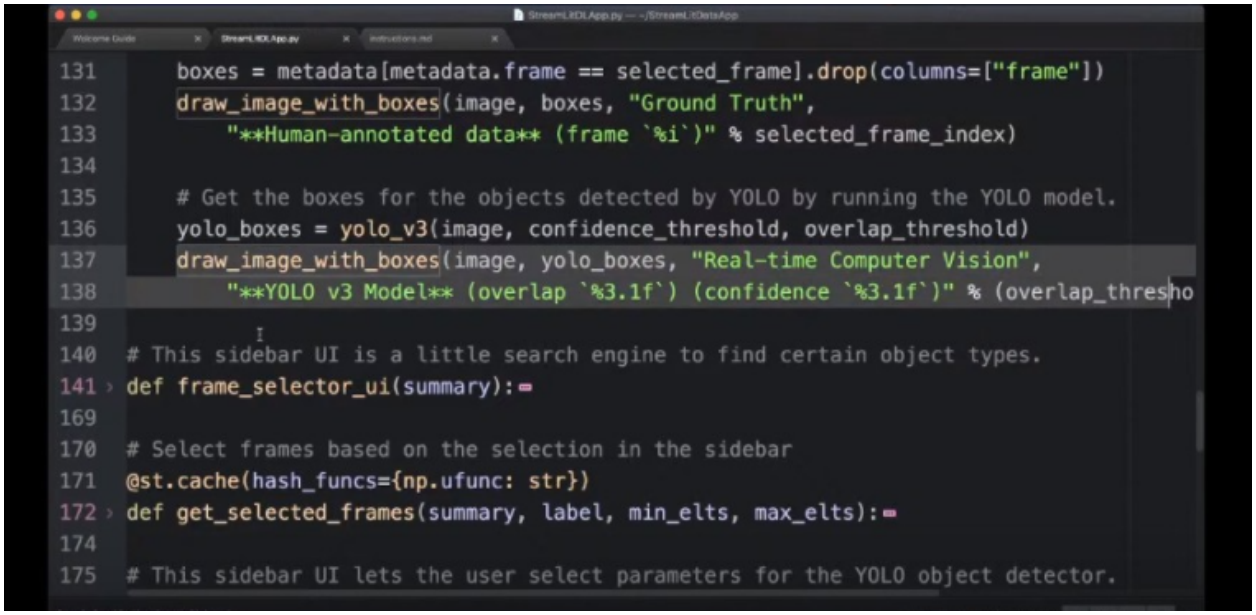
Timestamp : 46:11

We are using different colors for each of the classes.

An image or frame contains several objects. Each object has a bounding box associated with it. We are going to color those bounding boxes. A bounding box is specified by four coordinates namely xmin,ymin,xmax and ymax. We are slicing the image based on those coordinates and filling the colors. Then, we display the image along with the colored bounding boxes.



Timestamp : 49:55



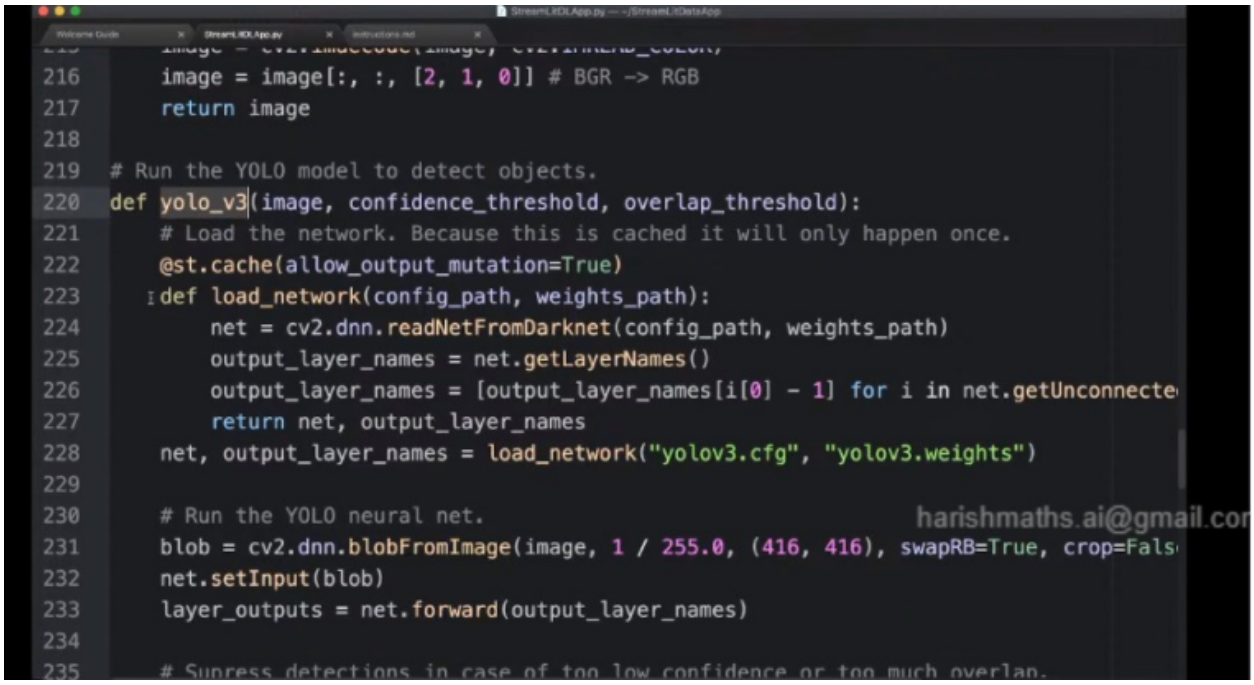
Timestamp : 50:48

Then, we run the yolo model with the following things:

1.) Image
2.) Confidence threshold

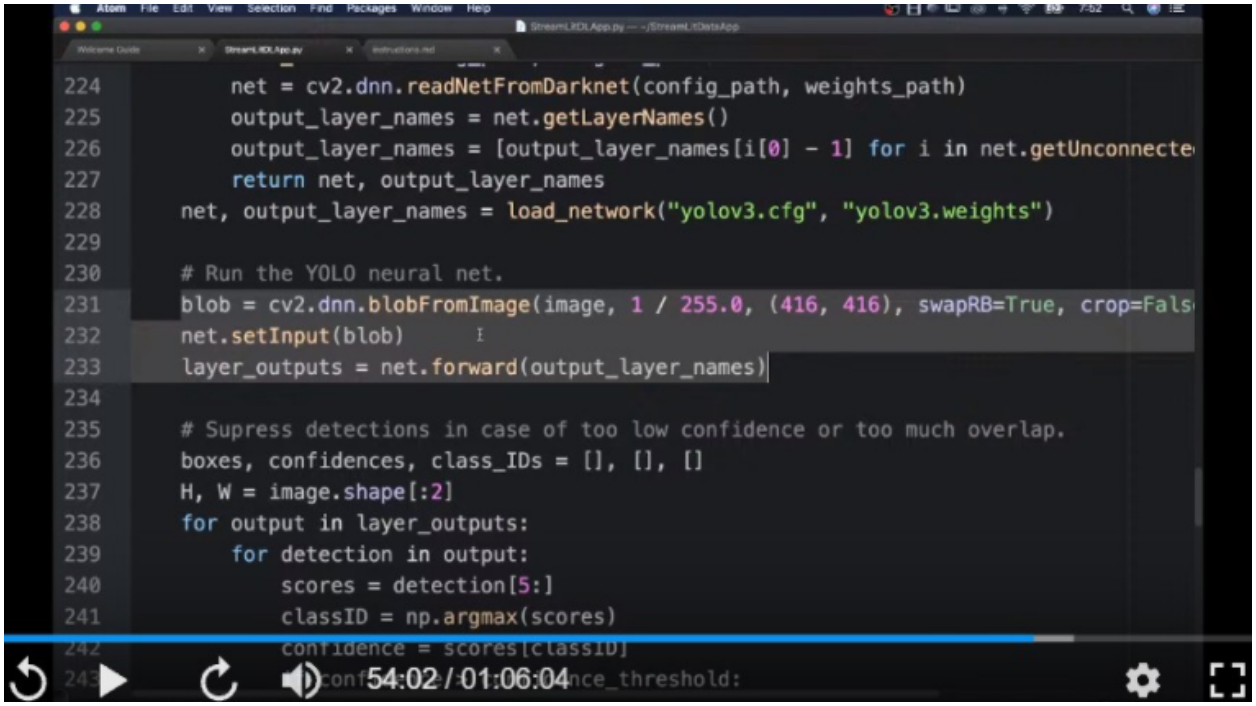
3.) Overlap threshold.

This is performed by the yolo_v3 function. Once we run the yolo v3 model, we get the predicted bounding boxes. Then, with the draw_image_with_boxes function, we draw the predicted bounding boxes on the given image.



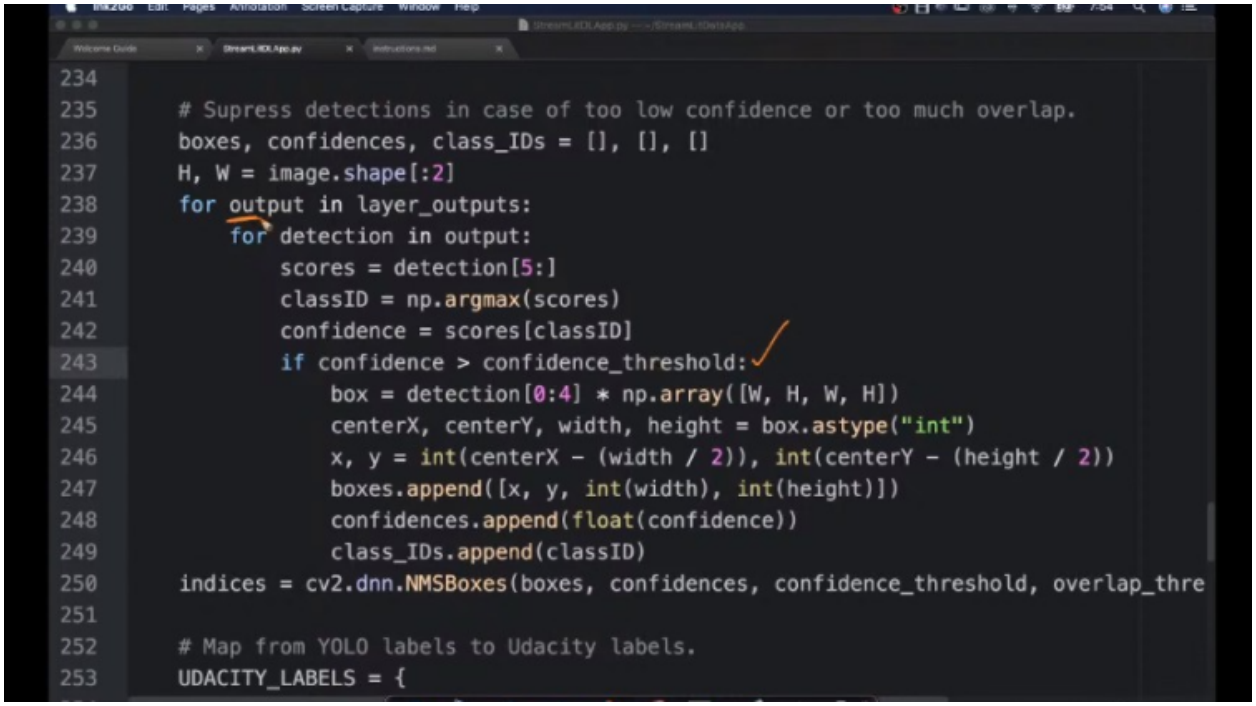
Timestamp : 51:42

The cv2.readNetFromDarknet method loads the yolo v3 model specified by the config_path and the weights_path. Darkenet is the base architecture of the yolo v3 model. Once we have the model, we simply pass the image to the model by calling the forward method.



Timestamp : 54:02

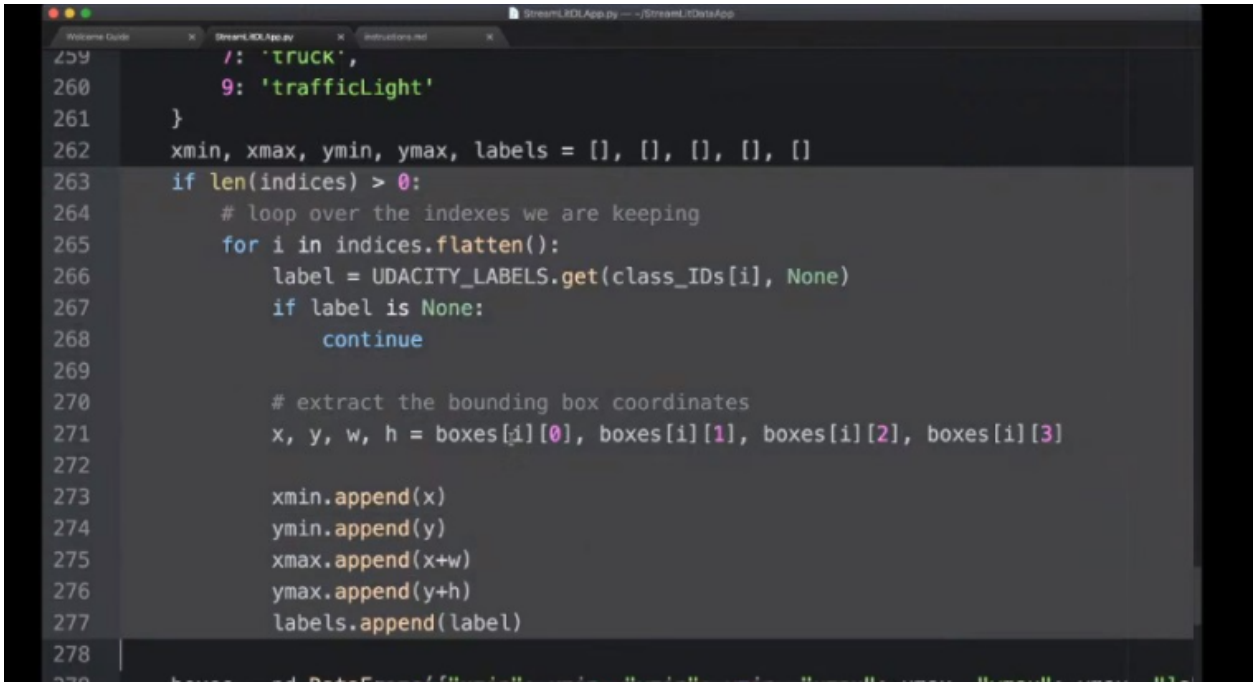
The layer_outputs has certain information about the predictions. These are the scores, confidence thresholds. We simply perform the argmax operation on the scores array and get the corresponding class label or id. The layer_outputs also contains the information about the predicted bounding boxes.



```
234
235     # Supress detections in case of too low confidence or too much overlap.
236     boxes, confidences, class_IDs = [], [], []
237     H, W = image.shape[:2]
238     for output in layer_outputs:
239         for detection in output:
240             scores = detection[5:]
241             classID = np.argmax(scores)
242             confidence = scores[classID]
243             if confidence > confidence_threshold:
244                 box = detection[0:4] * np.array([W, H, W, H])
245                 centerX, centerY, width, height = box.astype("int")
246                 x, y = int(centerX - (width / 2)), int(centerY - (height / 2))
247                 boxes.append([x, y, int(width), int(height)])
248                 confidences.append(float(confidence))
249                 class_IDs.append(classID)
250     indices = cv2.dnn.NMSBoxes(boxes, confidences, confidence_threshold, overlap_thre
251
252     # Map from YOLO labels to Udacity labels.
253     UDACITY_LABELS = {
```

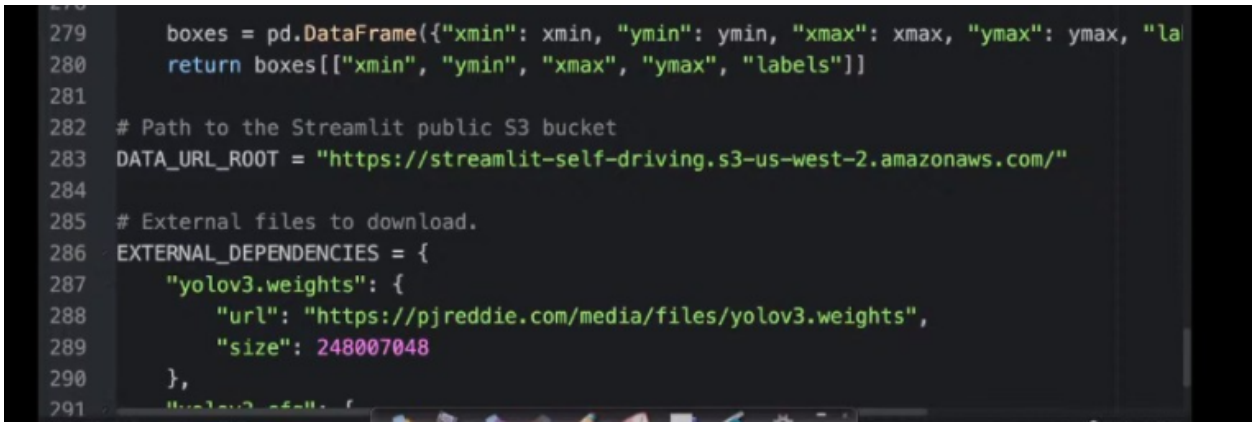
Timestamp : 55:59

Then, we perform the non-max suppression algorithm for refining the bounding boxes. We discussed this algorithm in one of our course videos. To know more about how to extract the exact location of those bounding boxes using centerX,centerY,width and height, please refer to the yolo v3 session in our course.



Timestamp : 57:34

We simply append those coordinates to a list. Then, we create a dataframe out of it for displaying on the screen.



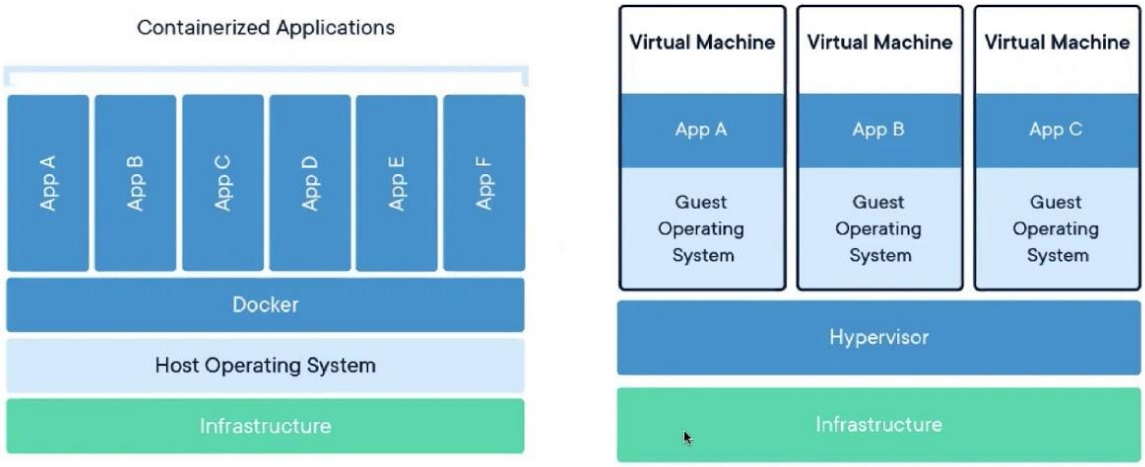
Timestamp : 57:41

47.5 ML model productionization using Heroku

Agenda

1) EC2 vs Heroku Architectures
2) Python Virtual Environment
3) Flask API + Gunicorn
4) Deploying Flask APIs on Heroku
5) Deploying Streamlit on Heroku

Virtualization vs Containerization

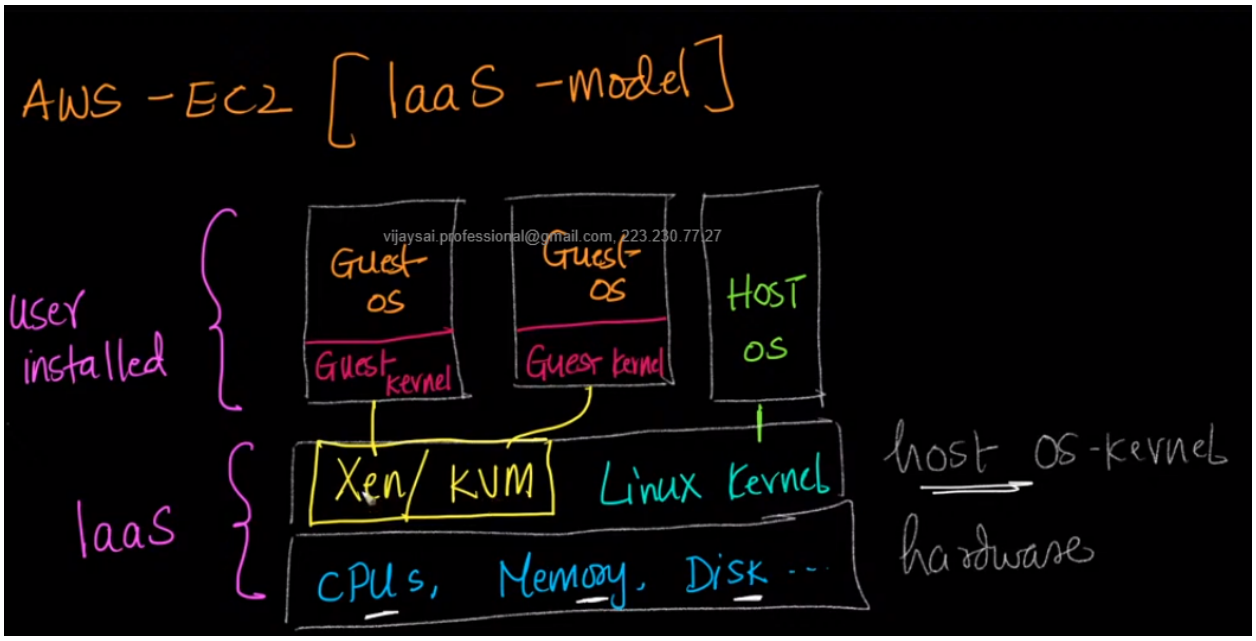


The virtualization approach is followed in Infrastructure as a Service (Iaas) system, whereas the Containerization approach is followed in the Platform as a service (PaaS) system.

In the virtualization setup, we have the hardware as the infrastructure layer, and on top of it, we have the Hypervisor layer which is a part of the fundamental operating system. This Hypervisor is also called a **Virtual Machine Monitor (VMM)**. The virtual machines sit on top of the Hypervisor. The Hypervisor communicates with the operating system and maintains the virtual machines. The job of Hypervisor is to create a virtual machine and install any operating system (Windows, Linux, etc) and then build an application on top of it. Every virtual machine needs an operating system to be installed, and due to this, they are heavy.

In the containerization setup, we have a host operating system installed on our infrastructure(hardware), and then on top of the operating system, we install a containerization layer(like docker). The applications are built on top of the containerization layer. Containerization can be considered as a lightweight alternative to virtualization because here we install the operating system only on the containerization layer that too only once, whereas in virtualization, we install the operating system on every virtual machine.

Let us have a look at the Amazon EC2 setup as shown below.

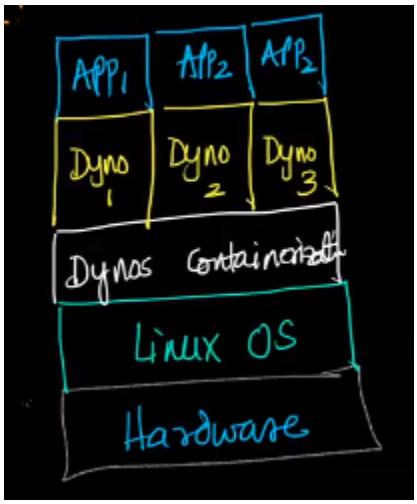


In the Amazon EC2 setup, we have the infrastructure (hardware such as CPUs, memory, Disk, etc) and a Hypervisor layer (Xen/KVM) on top of it, with Linux installed on it. Again from here, depending on the user requirements (like how many CPUs are needed, how much RAM is needed, which OS is needed), the virtual machines/boxes/instances are created and can be used by the user. The maintenance and management of these virtual boxes/instances is taken care by the hypervisor layer. Amazon EC2 works on the Infrastructure-as-a-service (IaaS) model architecture. In this setup, the virtual machines/instances are represented as **Guest OS** in the picture above. The virtual machine (here it is Guest OS) that was created depending on the user requirements, is now ssh into EC2 instance, and we can then run our application on it remotely. So in this

type of system, the user is getting Infrastructure(hardware) and he/she has to install the OS, packages, and everything according to his/her requirements manually.

Heroku (PaaS model)

Heroku doesn't use any virtual machines. Here we'll have the infrastructure(hardware) and the operating system(typically Linux OS) is installed on top of it. Then comes the containerization layer(In most of the PaaS models, we use docker as the container, but here in Heroku, we use Dynos as the container). Again there are multiple containers above this, and each application is built on top of each container(dyno).



Heroku is built mostly for web-based applications. Heroku simplifies application development. So far in AWS, if we want to build an application, we first have to configure a virtual machine(by specifying the RAM, number of CPUs, OS that has to be installed, etc) and install the libraries required, and then we can go for building an application. But here in Heroku, the procedure is as follows

a) Create a file **requirements.txt** which contains all the libraries/dependencies along with the version numbers, that have to be installed.
b) The **Procfile** contains information about all files in the folder, and all commands that should be executed, in order to build an application.
c) Depending on the file types, Heroku autodetects the type of application(whether Python app, Ruby app, PHP app, etc)

d) Once after having all the files that are needed to build an application, we have to push them onto the git and then start running.

Web Dynos (vs) Worker Dynos

Web Dyno is a container that can get the web requests and respond to the web requests, whereas Worker Dyno is a container that responds either to the web dyno or another worker dyno.

The Prepacked and compressed form of our application is called **Slug**. It is easier for the Containerization layer to process our application faster.

Virtual Environment

Whenever we need the same libraries/packages with multiple versions installed, then it is not possible to keep multiple versions on the same system. In such a case, we need the Virtual Environment in which we can install specific packages of desired versions and configurations. We can keep multiple virtual environments on our machine. Sometimes we may work on multiple projects and each project might require different versions of packages. In such a case, as it is not possible to install multiple versions of the same package, we create a virtual environment for each project and install the packages on them as per our requirement.

Syntax to install Virtual Environment

pip install virtualenv

(or)

conda install -c conda-forge virtualenv

Example to create a Virtual Environment

*python -m venv **venv/***

In the above syntax, we are asking Python to create a virtual environment. The first

'venv' indicates Python to create a virtual environment and the second **'venv'**(the one mentioned in bold font) indicates the folder name to store the information about the created virtual environment.

Command to activate the above created virtual environment

source venv/bin/activate (in linux)

(or)

.\venv\Scripts\activate (in windows)

Command to deactivate the above created virtual environment

deactivate

In general, a virtual environment is used if we need multiple versions of a programming language/framework to be installed. After the installation of the virtual environment and the required packages, we have to either build a Flask API (or) borrow an existing API. For now, we shall borrow the IRIS API that was already created. Here we can first run this API on the local PC and check if it is working smoothly.

In order to write the list of all the packages installed in the virtual environment onto a file, we can execute the command

pip freeze > requirements.txt

This statement writes the names of all the libraries installed on the virtual environment into the file '**requirements.txt**'.

Note: For the code and commands, please go through the PDF file from the google drive link, that was added in the description of the video session.

47.6 Testing and Debugging ML/AI systems end to end

Data Debugging and Testing

Let us assume we have a dataset D = $<x_i, y_i>$ where $x_i \in R^d$ (f_1, f_2,…, f_d)

Let us assume we have a feature 'f_1' that denotes the day of the week. The possible values for this feature are the numbers from 1 to 7. The numbers like 0 and 8 and values like NaN aren't accepted. This feature is a categorical and an ordinal feature. We have to first check if there are any values that are outside the domain of the valid values. (For example, for day of the week, the values 1 to 7 are considered to be the valid values. Any value outside this range is considered to be invalid). Similarly, we also have to check the distribution of the values of this feature, and make sure it is not skewed. Having a skew in the distribution of any feature would impact the ML model badly.

Similarly let us assume we have a feature 'f_i' that denotes the states in a country. Here we should make sure that these values come from the names of the existing states only. For example in India we have around 29 states, so the values of this feature should be only from these 29 values. Also we should make sure only one form of the values is followed. For example, some points might contain the values as "Andhra Pradesh", whereas some might contain the value as "AP", some points might contain the values as "Madhya Pradesh", whereas some might contain the value as "MP", etc. We should make sure either the full forms (ie., Andhra Pradesh, Uttar Pradesh, Madhya Pradesh, Jammu and Kashmir, etc) are used or only the short forms (ie., AP, UP, MP, J&K, etc) are used. The dataset shouldn't contain a few values in full form and the remaining in short forms.

Also we need to check the distribution of the data. For example, we might have huge data from the states like Uttar Pradesh, Maharashtra, Bihar, etc and a small amount of data from the states like Sikkim, Manipur, Mizoram, etc. We need to be aware of this, because such distributions affect the models badly. The models might perform better on the data points with

states of higher frequencies, and the points with the states of lower frequencies.

Similarly let us assume we have a feature 'f_i' that denotes the age of an individual. It is a discrete numerical value. We should make sure there are no invalid values like text data, NaN, insensible values(like age being 140 years, 180 years, etc). Also sometimes if the user is not willing to provide his/her age, then the default values are filled automatically, which is not encouraged.

Let us assume we have image data, then we might see differences in the size of the images, types of images (like grayscale or RGB), duplicate images, etc, which aren't accepted. Similarly if our feature takes only the text data, then we should make sure all the words are coming from the same language, no misspellings and maintaining proper sentence formation, etc

Data Splitting

There are 2 strategies followed while splitting the dataset. They are

a) Random Splitting
b) Time Based Splitting

If we perform random splitting instead of time-based splitting or vice versa, then we encounter data leakage. Also we should always perform the statistical difference test between D_{Train} and D_{cv}, D_{cv} and D_{Test}.

Feature Transforms

Whenever you apply Standardization on the features, sometimes you may come across division by zero error (this happens when the standard deviation is 0. Standard deviation becomes 0, if all the values in the feature are the same). In such cases, we have to add a small minor value to the standard deviation in the denominator, so that we could get rid of the division by zero error.

One more problem is with the Feature Imputation. Let us assume we are working on a binary classification problem, and the number of points with missing feature values are more in class 0, when compared to class 1, then directly applying the imputation technique and using the data for modeling will affect the model. In case, if there are more imputations

performed on the points with class 0, and less imputations on the points with class 1, then it makes sense to add a feature '**isMissing**' which takes 1, if the feature value is missing, otherwise it takes 0.

Also when there are outliers associated with each feature, we have to clip them. If a numerical feature has a certain number of values, then if we have a value that is far outside the range, then we have to replace it with the maximum value, if the value is greater than the max, value, or else, we have to replace it with the min.value if the value is less than the minimum value.

When we apply one-hot encoding on a categorical feature with 'k' values, we should make sure only 1 column should have the value 1, and the remaining k-1 columns should have the value 0.

In case, if we apply Normalization, we should make sure that the values of the transformed feature lie in the range [0,1]. In case, if we apply standardization, we should make sure that for the transformed feature values, mean = 0, and standard deviation = 1.

Whenever we apply count based Bag of Words vectorization on the text data, we should make sure the sum of the counts of all the features in a vector is equal to the number of words present in the text.

Whenever we apply TF-IDF vectorization, we should check for the words with highest and smallest TF and IDF scores.

Whenever we apply Word2Vec vectorization, we should check for the most similar and the dissimilar words from the word vectors.

Whenever we are working on the image data, we should visualize the kernels in each stage.

Model Debugging and Testing

We have to perform univariate analysis to know which features are the best and which are the worst. Sometimes we can get this information from the domain experts also. It is a good practice to check with them before discarding the worst features.

We have to build a simple baseline model using the best features obtained from the univariate analysis and measure the model performance. In case, if the baseline model performance is good, then it is guaranteed that building complex models using these features(used in the baseline

model) could not only give a better performance, but also add more weightage to the model. If the baseline model doesn't give a good score, then we can ignore such features.

We first have to build a random model (or) a dominant class model(ie., which returns the dominant class, if the training dataset is imbalanced) and then build a simple ML model, and compare these models in terms of the performance scores. If we could get better scores, then we can go for the complex models, and these complex modes should perform well at predictions on both the CV data and the test data.

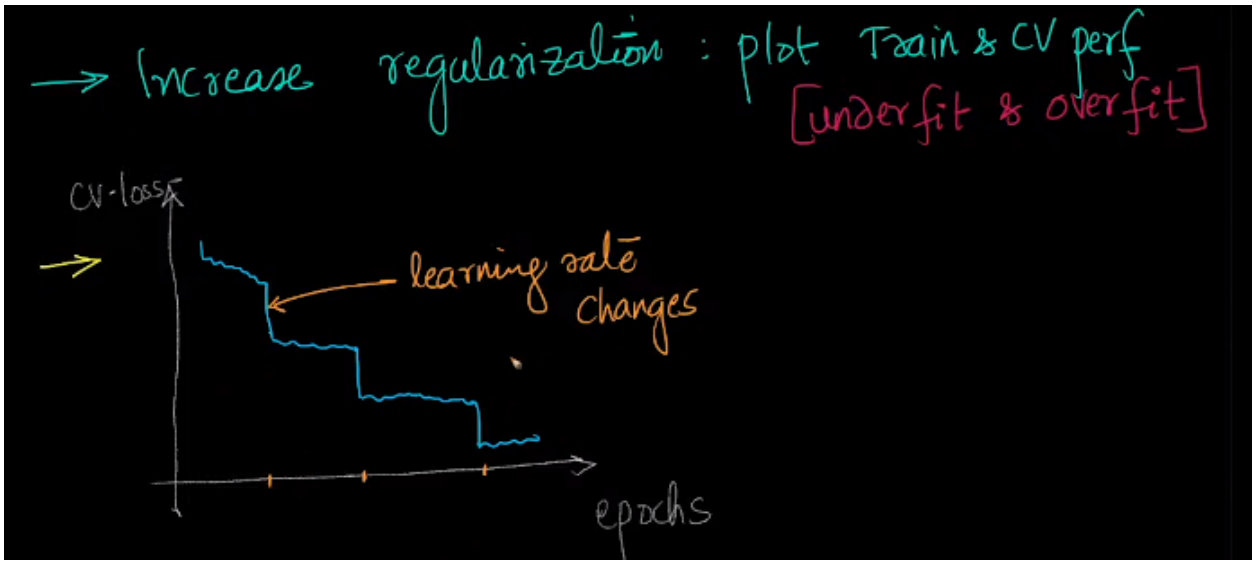
Model Implementation Testing

We have to take a few points in the training data and then train a model without applying regularization. The model has to overfit.

Similarly, we have to initialize all the weights to a constant value(either 0 or 1). Also, we have to keep tracking the gradients and the learning rate, by building the plots for each epoch.

Keep increasing the regularization and build the plots showing how the training and the CV scores are varying with an increase in the number of epochs. We have to monitor both overfitting and underfitting.

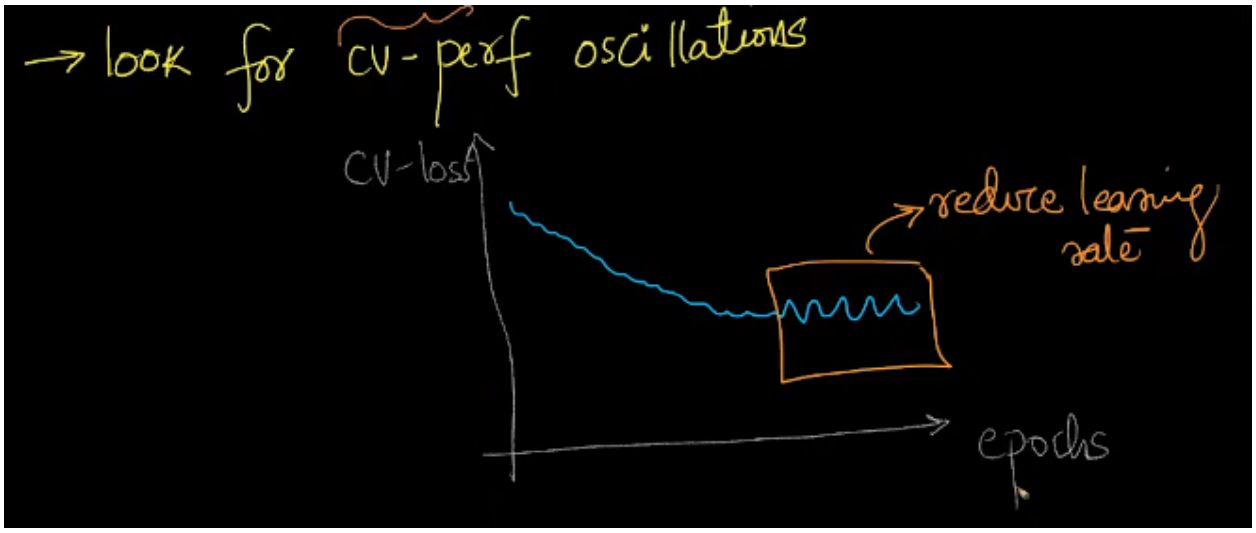
All these are the sanitary checks that are to be performed in order to check whether our code implementation is working correctly or not.



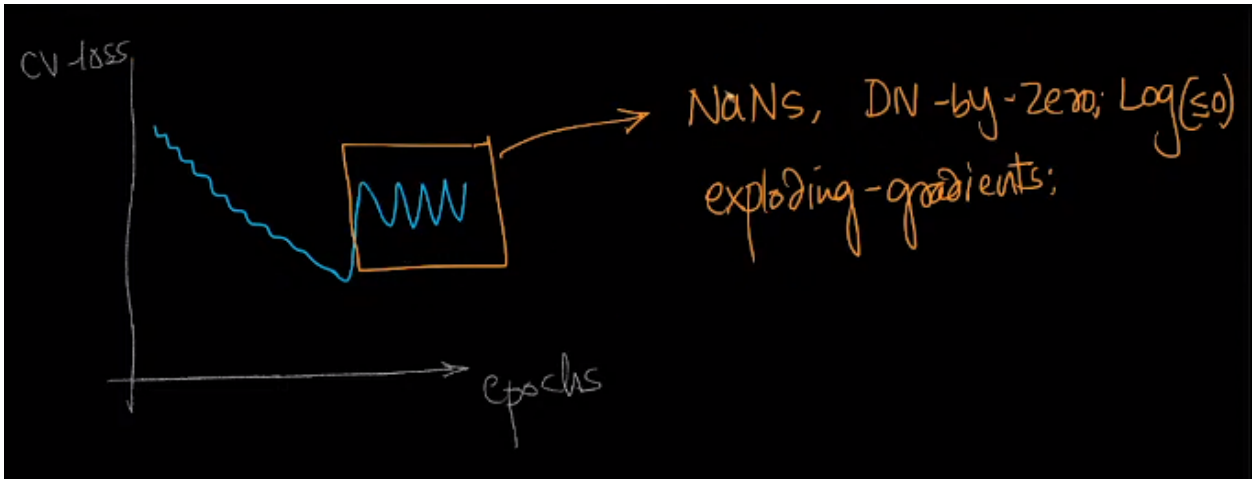
When we build a line/scatter plot with the epochs on the 'X' axis, and the training and the validation losses on the 'Y' axis, then whenever we

come across the behaviour of losses as shown above, then it means at the sharp drop in the losses, the learning rate is changing, and the code that updates/changes the learning rate is working.

Sometimes we also come across the behaviour as shown below, which means we need to reduce the learning rate, as the loss function is oscillating.



Sometimes we come across the behaviour of the losses as shown below.



This case indicates the presence of NaN values, Division by Zero error, Logarithm value being less than 0, exploding gradients, etc.

The below case shows us a cycle of increase and decrease in the loss. This indicates for every epoch, the same batch is being repeated. So at the end of every epoch, we have to shuffle the points in the batches randomly.

Unit Tests for each function

Here we have to check the values of the

a) **Sigmoid(z)**, at z=0, a large positive value (say 10000, so that we should get sigmoid output as 1), and at a large negative value (say -10000, so that we should get sigmoid output as -1)
b) **Derivative of sigmoid**, at z=0, a large positive value, and at a large negative value.
c) **ReLU(z)**, at z=0, a large positive value, and at a large negative value.
d) **Derivative of relu**, at z=0, a large positive value, and at a large negative value.

One more important thing to be observed is the distribution of the inputs in the deeper layers of the neural networks. If these distributions are varying, we have to apply batch normalization.

One more issue we encounter is the **division by zero**, specially when we apply the data normalization (or) batch normalization. In such cases, we have to add a small value '**ε**' in the denominator to prevent the division by zero.

Domain Knowledge based Validation

Whatever model we build, it has to be verified by the domain expert (or) we can validate it by ourselves, if we have strong domain knowledge. The simple way to do it is by taking the feature importance.

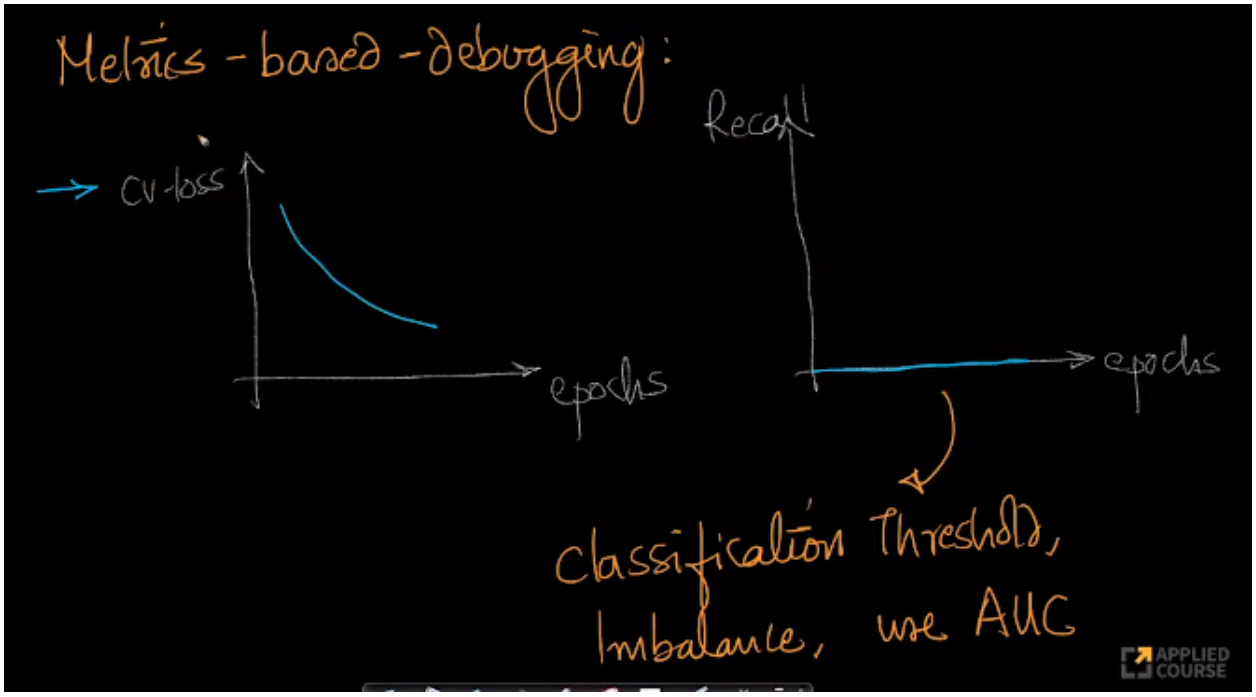
Model Failure Cases

Sometimes we do come across a few scenarios where our model fails. In such cases, we need to be able to explain the reasons for the failure. For this we need to check the input we are passing, the different layers the input is passing through, the intermediate layer outputs, and the final output that is different from the actual output.

Sometimes the changes in the output, changes in the preprocessing steps, changes in the feature transforms, changing the loss function, changes in the weighted models, etc could fix the failures.

Metrics based Debugging

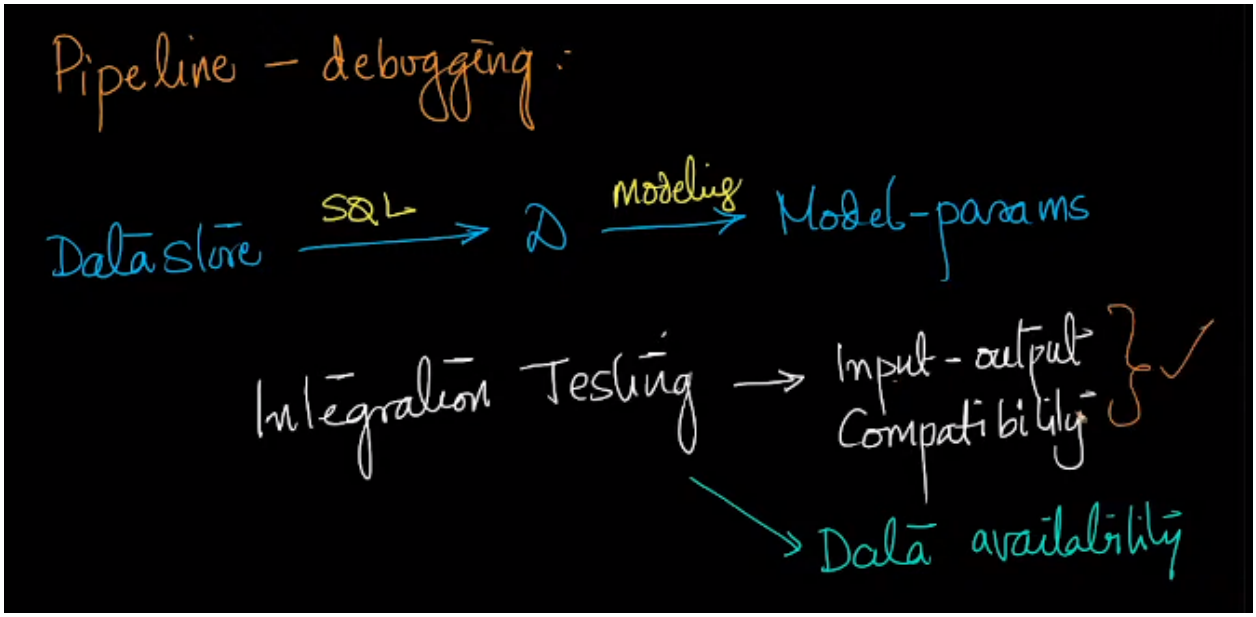
Let us take two metrics ‘Log Loss’ and ‘Recall’ into consideration. During the model training, the log loss decreases with increase in the number of epochs, whereas the Recall value remains constant. It is because Log Loss is a performance metric, and recall is a business metric.



One of the reasons for the occurrence of this scenario could be the value of $\mathbf{P(y_i|x_i)}$ being strictly less than 0.5. One other reason for this could be the imbalanced nature of the training dataset. One solution to handle this kind of issue is to apply Calibration(either with Platt’s scaling or Isotonic Regression) on top of the model. We also can change the metric from Recall to AUC. We also can use multiple techniques, but all those metrics should be sensible.

Pipelining-Debugging

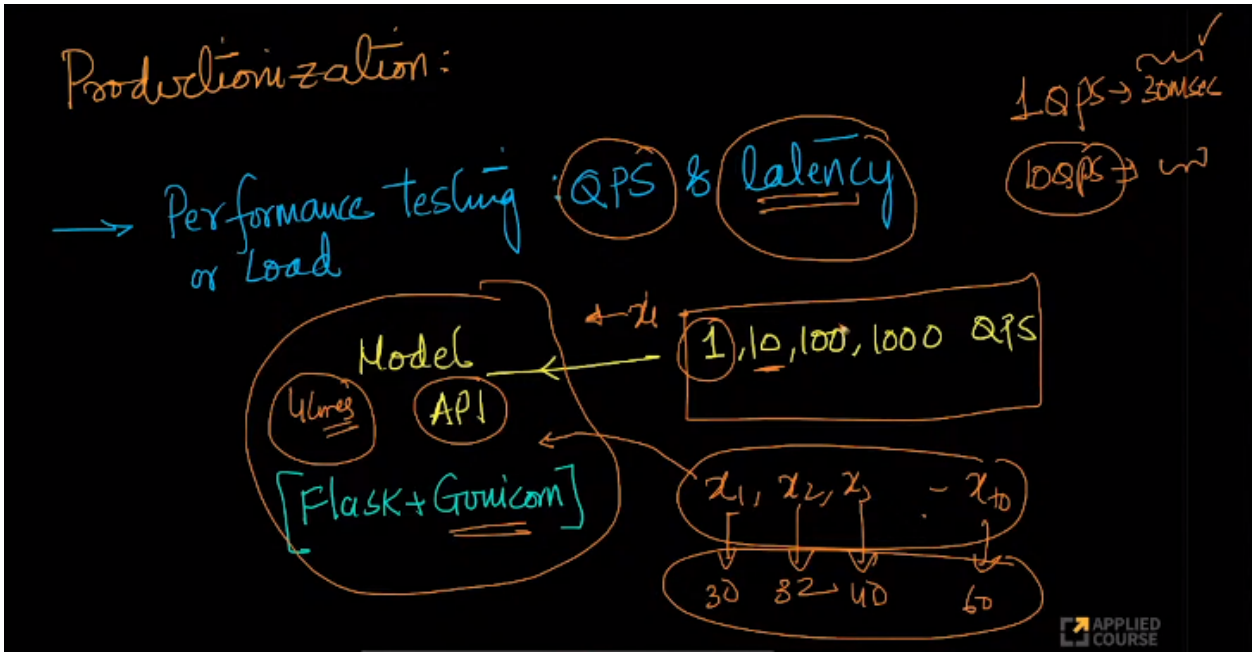
In general, we extract the data from the datastore using SQL, and use it for modelling and compute the model parameters. The major thing to be performed here is the **Integration Testing**. It is of two types. They are the **input-output compatibility** and the **Data availability**.



Input-Output compatibility is nothing but checking whether both the input and the output are of the same shape(dimensions), same data types,valid ranges, etc. Similarly when we schedule a model retraining at a particular point of time, then in case, the required data hasn't arrived by that time(either due to server failure or database failure), then we couldn't retrain the model. Such a situation is called **Data Unavailability**.

Productionization (QPS and Latency)

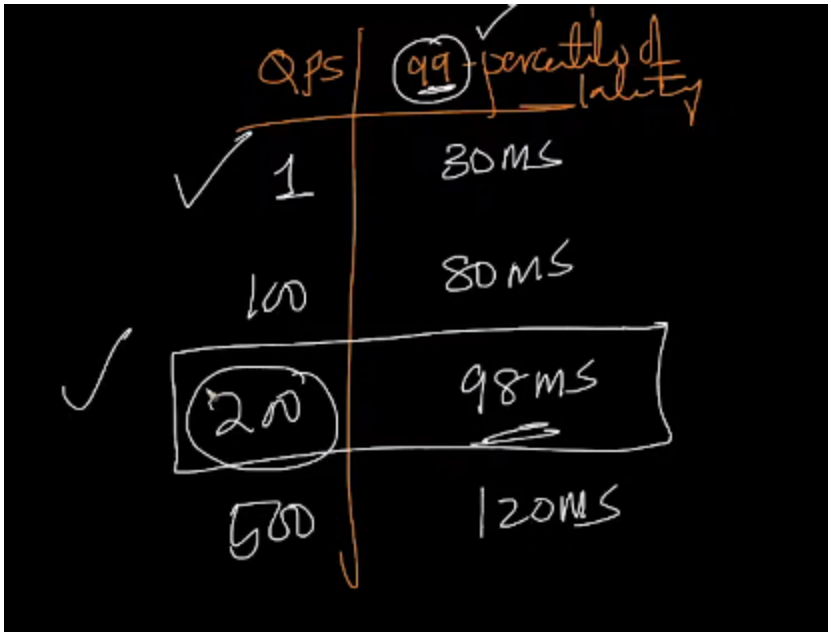
In Productionization in general, the software engineers do the load testing (or) performance testing. Even in machine learning, we have to perform this testing.



Here we have to process the first one query per second(1 QPS) and check the total time the model took to process it. Similarly we have to pass 10 queries per second, and compute the average time required to process all of them. We then have to increase it to 100. The average time required to process all these queries is called the Latency. We have to make sure the latency is as low as possible. This is a form of load testing to check how much load our model can take.

` Here each query has a different processing time. Different combinations of queries picked randomly (ie., though the number of queries is the same, the combinations are different) will have different processing times (latency). For different combinations of the same number of queries, we have to compute the average time.

Hence we need to build a plot with the number of queries as the value on the ‘X’ axis and the corresponding 99th percentile latency as the value on the ‘Y’ axis.



The software testing concept discussed so far is just an outline. For more information on it, please go through the below Wikipedia article.
https://en.wikipedia.org/wiki/Software_testing

A/B Testing

It deals with measuring the causality and improving the models. It is a part of the machine learning pipeline. Here if we already have a model deployed in the production (say 'M_{old}') and if we have a new model built (let us say 'M_{new}'), then we pass 50% of the data to 'M_{old}' for the predictions, and the remaining 50% to 'M_{new}'. If 'M_{new}' gives better results when compared to 'M_{old}' and if the business increases(ie., like the sales in e-commerce, etc), then we deploy the 'M_{new}' model in the production.
Note: In general, we directly do not go for a 50-50 split on 'M_{old}' and 'M_{new}'. We start with smaller percentages of 'Mnew' and increase it step by step, and finally, after we get the best results on the entire 100% data, then only we deploy the 'M_{new}' model. (In general, we go with 95%-5%, 90%-10%, 85%-15%, …., 50%-50%, 45%-55%,.....,0-100% ratios)

47.7 [Optional, but strongly recommended] : SageMaker Part 1

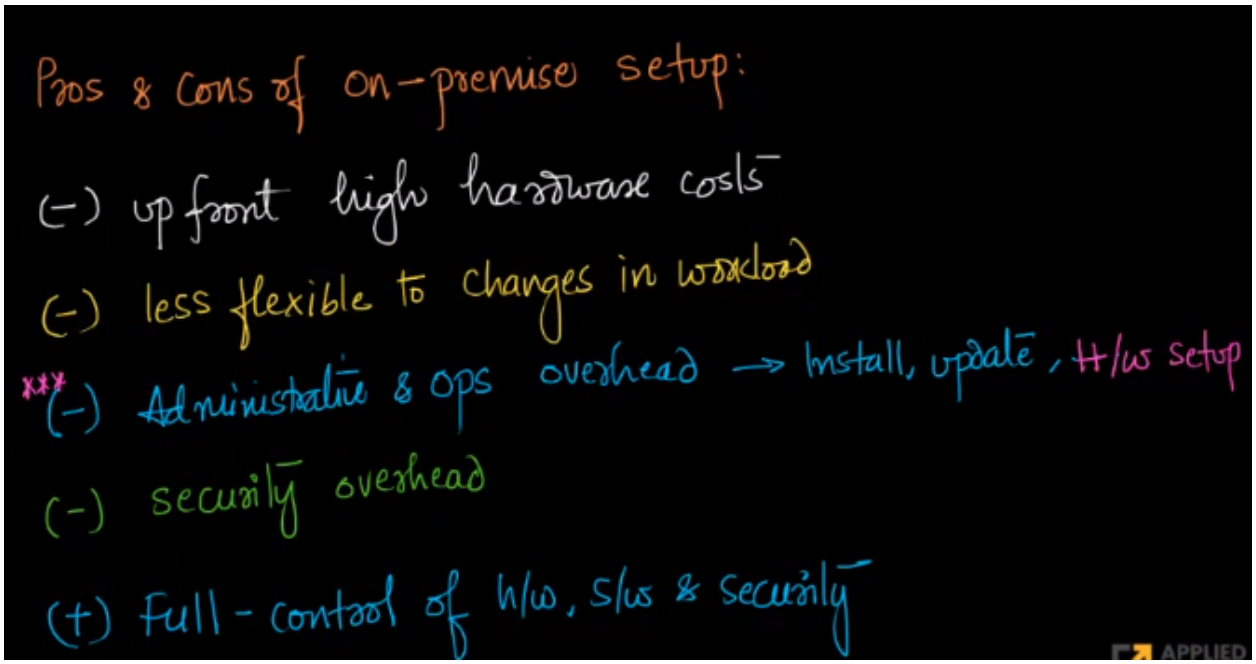
Sagemaker is an end-end machine learning platform on Amazon's cloud [AWS]. It helps you to perform everything on the cloud-based servers.

End-End ML Pipeline

On-Premise

Instead of using cloud-based modeling, typically large IT companies use on-premise Machine learning. On-premise learning refers to the installation of huge high-end servers. Some of them are used as database servers, Data warehousing, log servers. We can basically do all the ML problems end-end on the on-premise without using any cloud-based platform.

Pros and cons of On-premise:

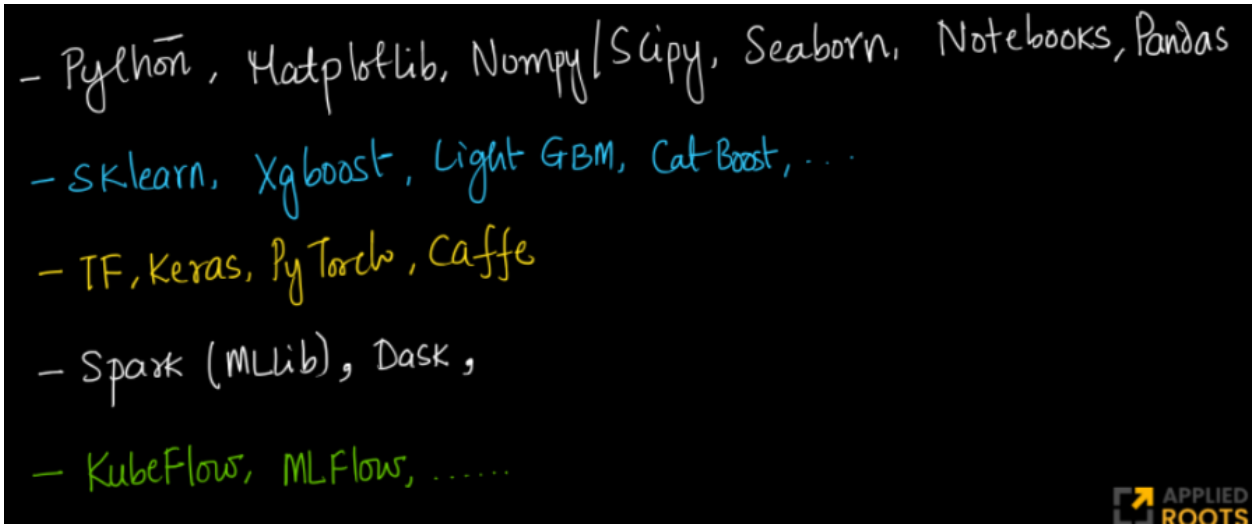


Cloud:

Cloud system offers to rent the hardware and software operations.
The largest cloud services are AWS [Amazon Web Services], Azure, GCP[Google computing Platform] followed by IBM and Oracle.
The way the cloud works is, they set up the huge data center at some locations, and offer to rent this hardware or operating systems like Linux, windows, or mac for anyone who needs it.
The advantage with this is, we can get everything installed. There will be minimal to no operational procedure like installing operating systems or updating anything.

Popular Open-source tools/libraries:

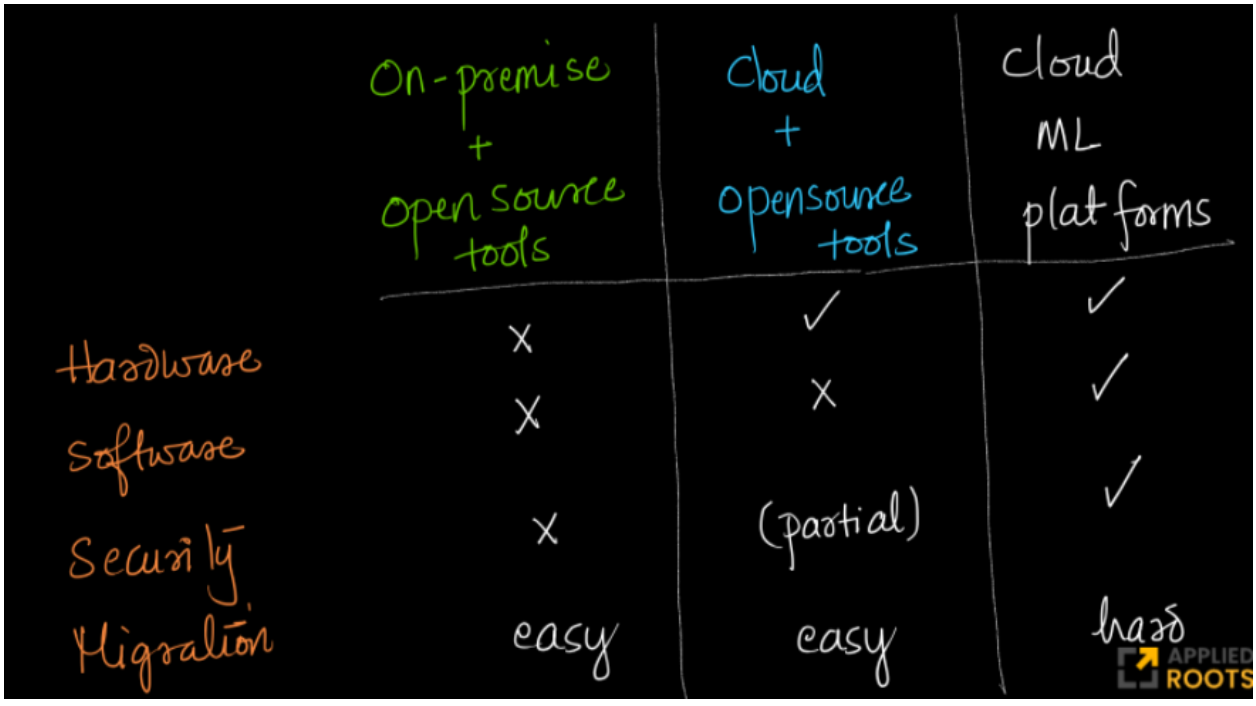
These are the most popular tools/libraries used in Data science whether it may be on cloud-based platforms or on-premise.



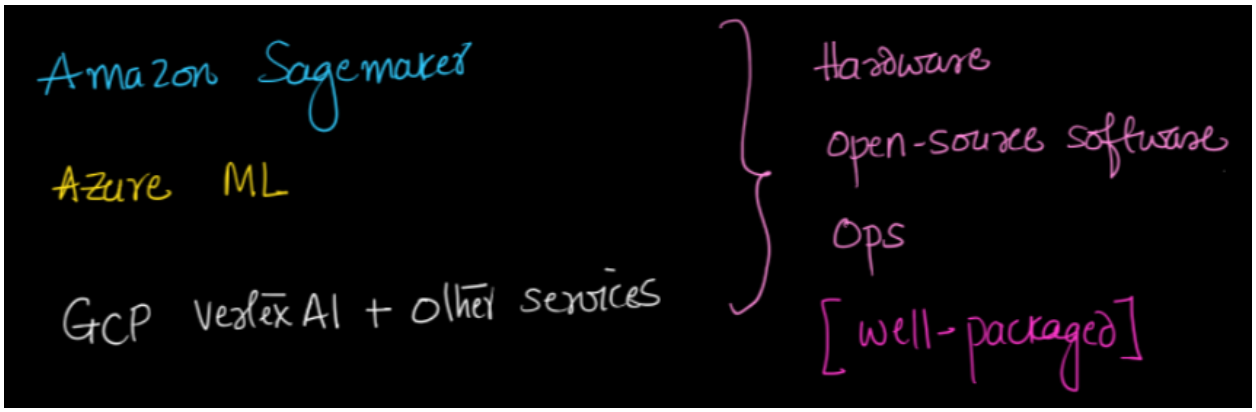
Building machine learning models can be done in any of the three ways.

1. Using open-source tools with on-premise machine learning.
2. Renting cloud systems instead of buying the high-end servers and installing open-source tools on top of it.
3. Using end-end Cloud-based Machine learning platform.

Comparison of three ways:

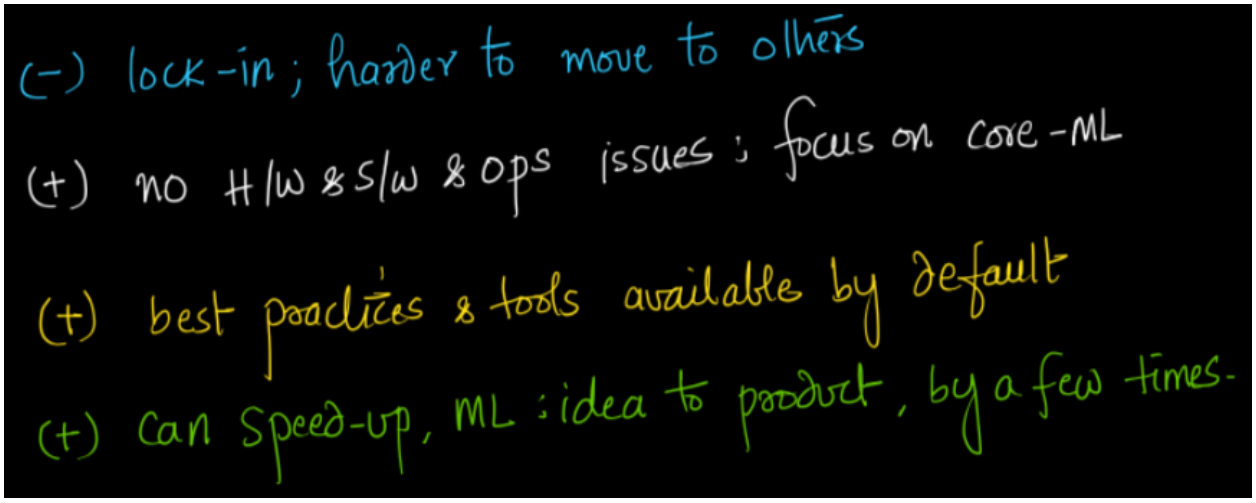


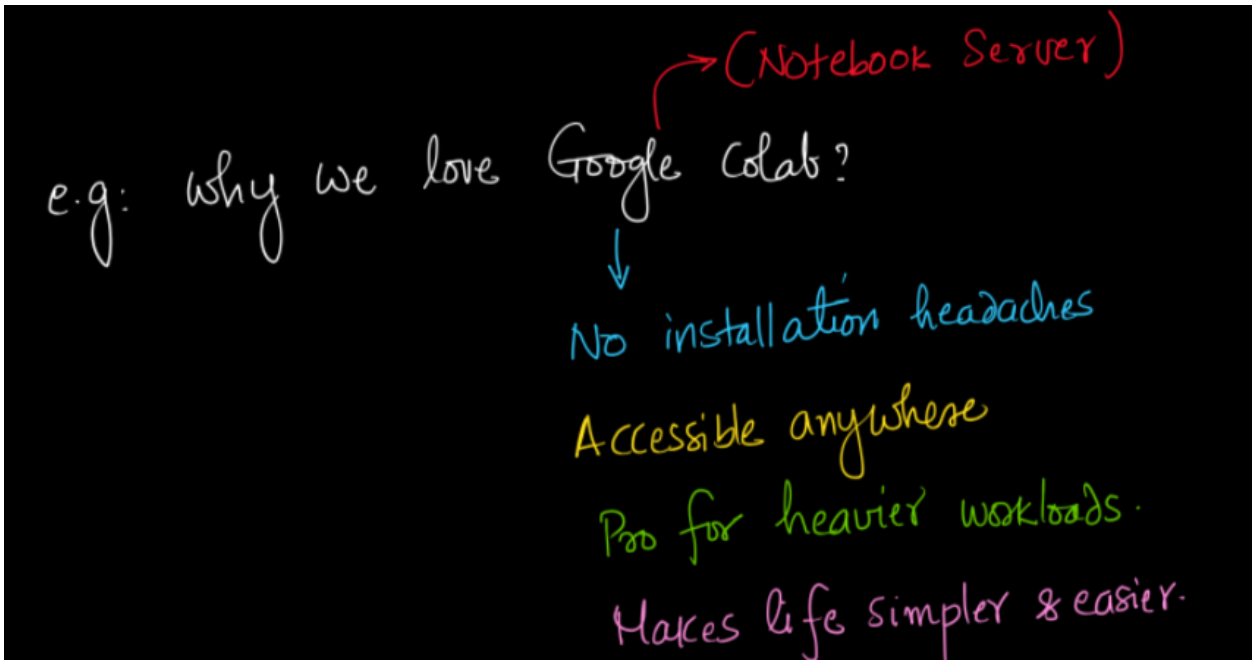
Cloud-based ML platforms:



The most mature Machine learning platform is Amazon Sagemaker closely followed by Azure.
Amazon Sagemaker offers most features and tools than any others.

Pros and cons of CLoud Ml platforms:



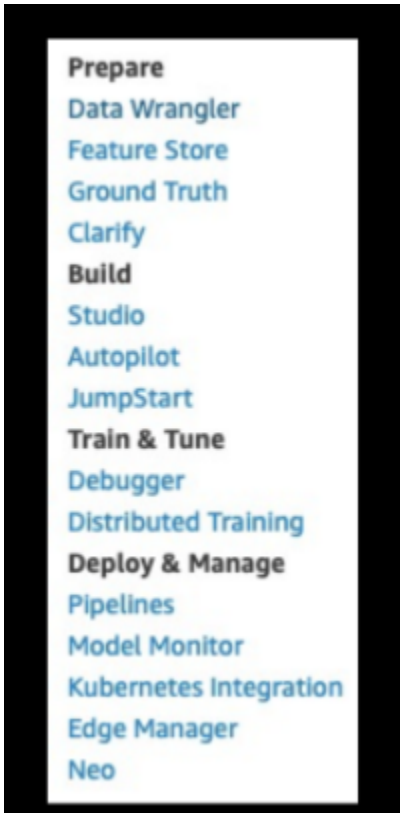


When to use Sagemaker?

1. Especially if your data is already on AWS
2. When a team has at least basic knowledge of AWS.
3. When there is no need to migrate to other platforms or on-premise in the future.
4. When the companies you work with have no issues regarding the usage of Sagemaker.

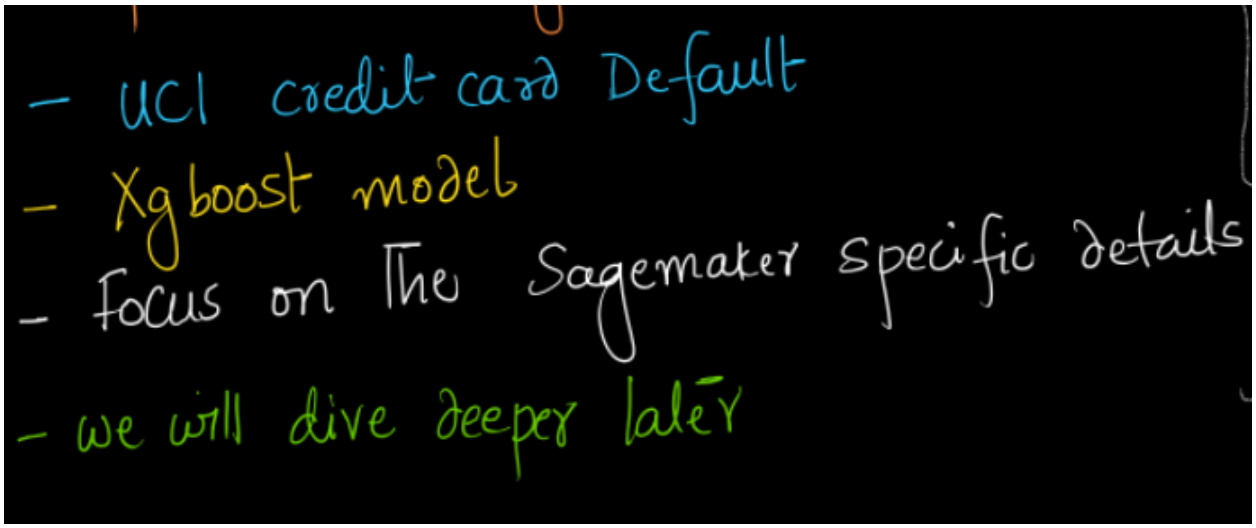
When Not to use Sagemaker ?

Key Features of Sagemaker:

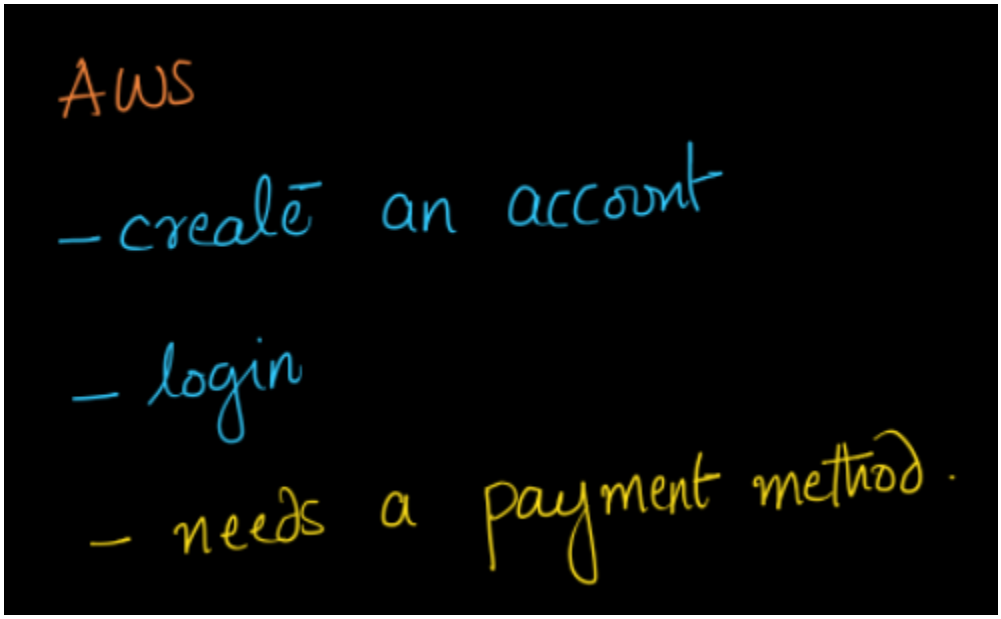


Data wrangler can help you import data from various sources like Data warehouse, Database, or any cloud storage.
We will go through the remaining features later in this session.

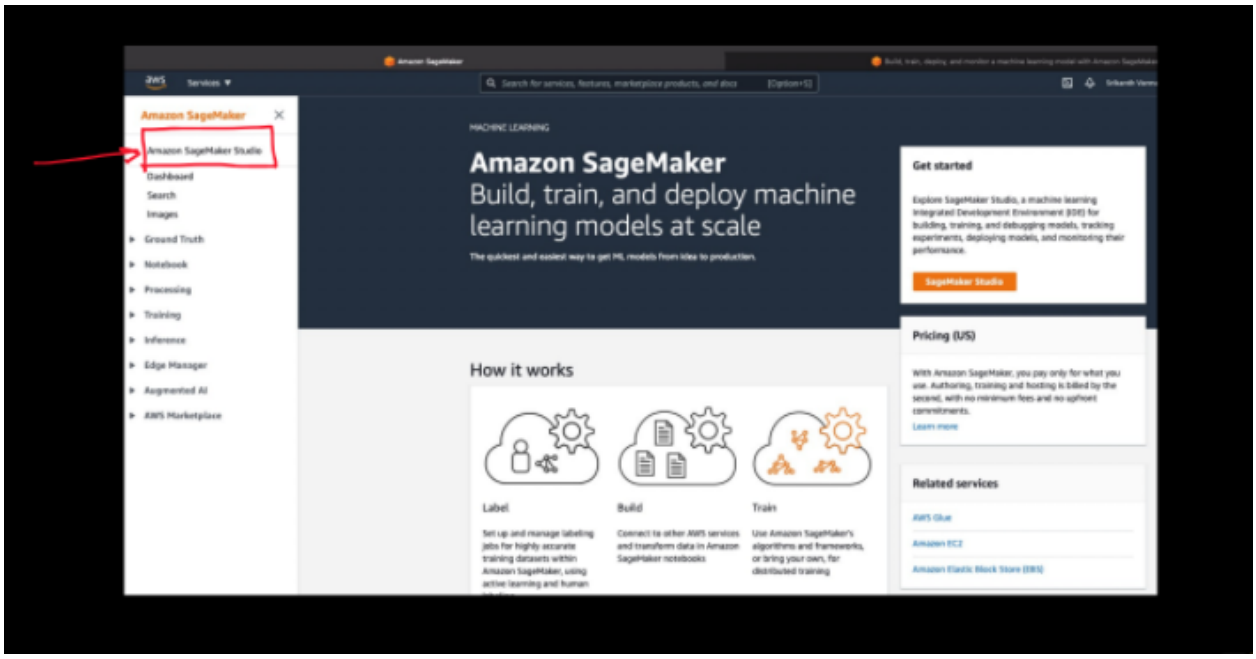
Simple end-end Machine Learning System:



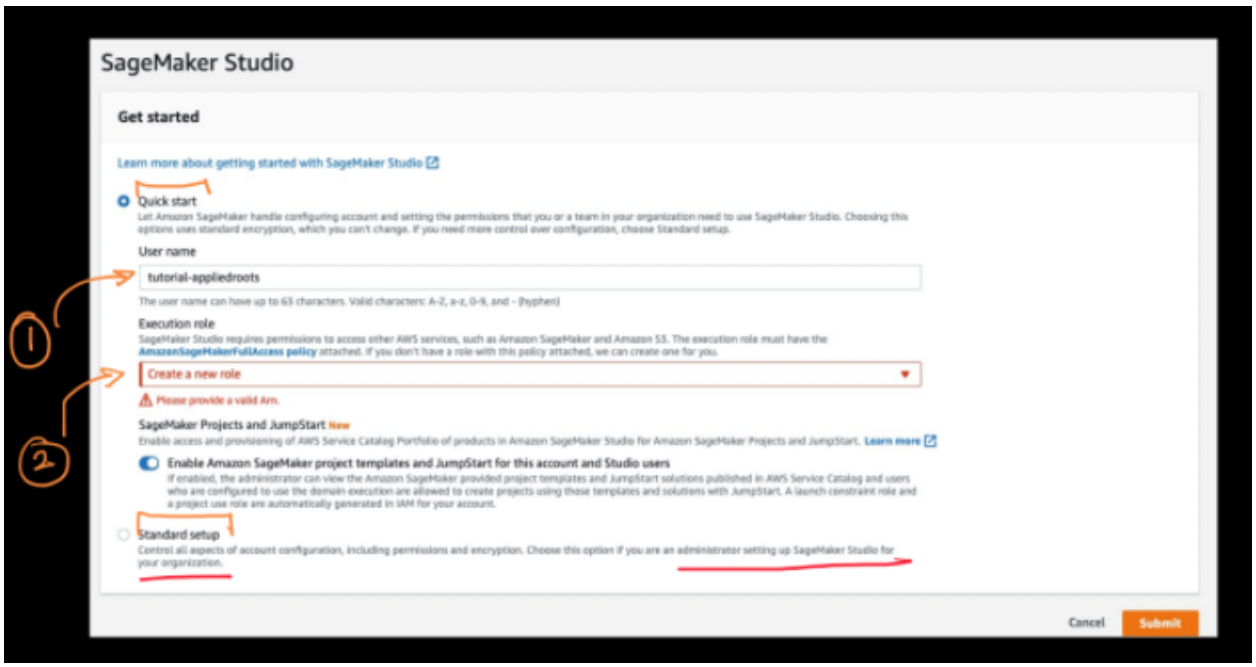
Please follow the documentation of AWS.



Go to Amazon sagemaker's Homepage > Sagmemaker studio > SetUp



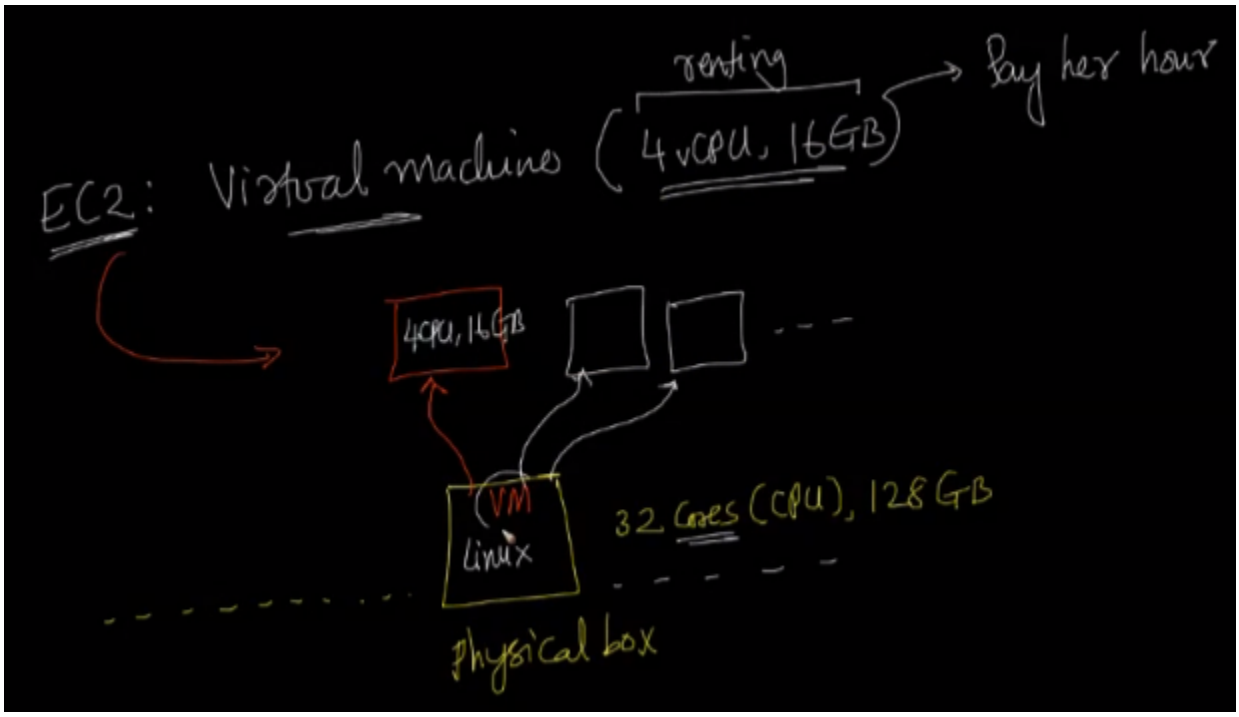
There are two options for this. You can either do either Quickstart [suitable for beginners] or standard setup [For Admistrator].



Sagemaker uses a lot of other AWS services internally like S3, EC2

S3[Smple Storage Service] is the first AWS service ever built. It is more like a cloud drive where we store all of our files/folders.
It is distributed disk/storage management system.

EC2[Elastic Compute Cloud] is a virtual machine that provides Computing power.



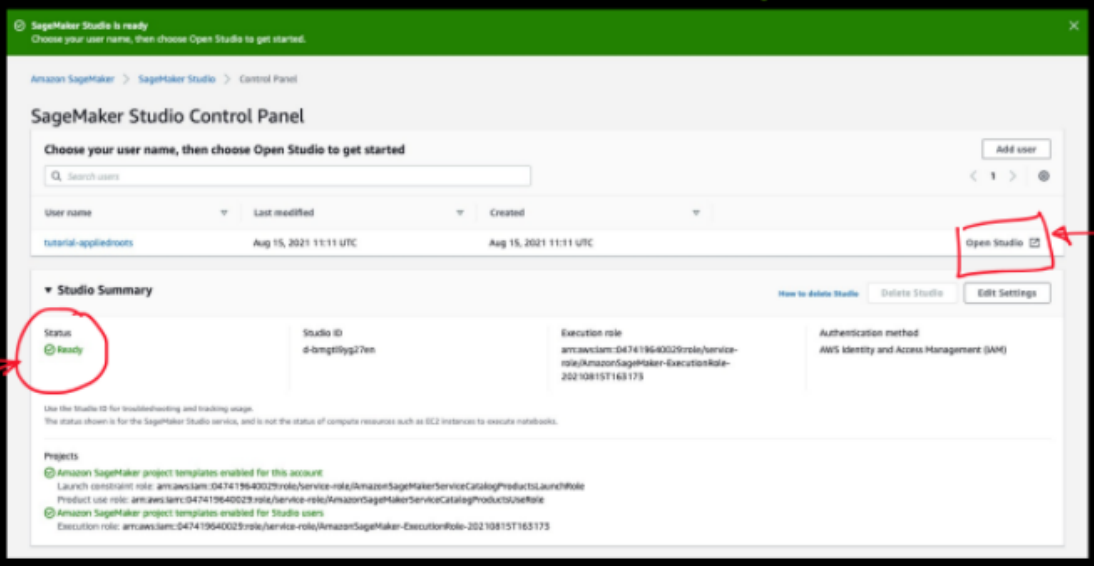
In the Execution role, we have to specify the S3 bucket we are accessing and click on submit. After few minutes, the Sagemaker studio will be done and through the studio, we can do a lot of stuff like building models, deployment.

Execution role:

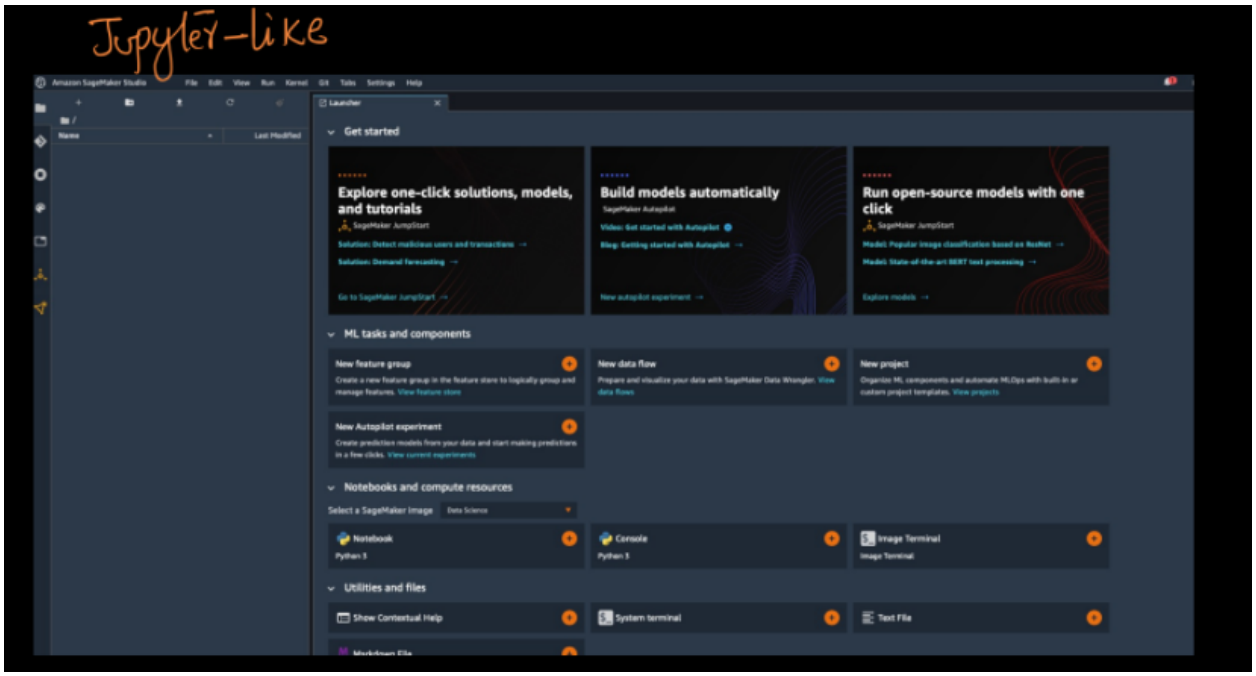
S3 : Simple Storage service

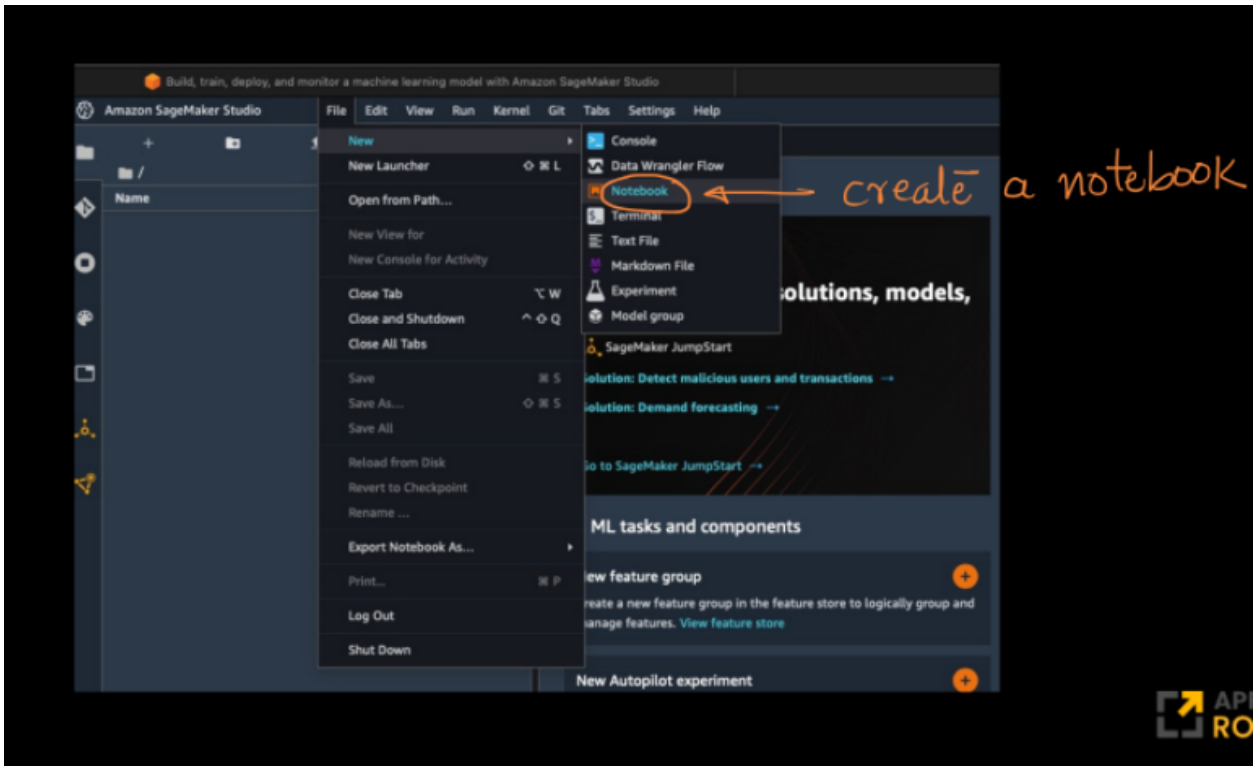
Data to S3

after few mins



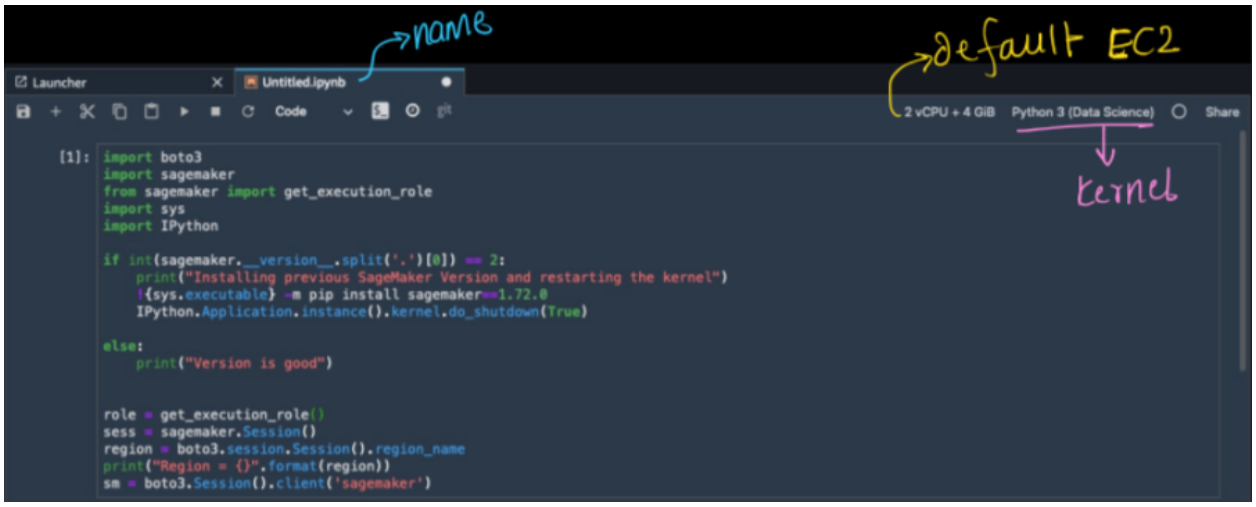
The Sagemaker studio is more like the Jupiter Notebook interface with some additional features and tools. As most data scientists are used to jupiter, it becomes easy for them to migrate to Sagemaker.





To create a new Notebook, Go to File > New > Notebook

After creating the new ipynb, we can choose a kernel from the various options. The preferred kernel for MI stuff will be Python3(Data Science). We can also select PySpark kernel where we can write Spark code in different programming languages like scala, python, Java,R



There are some specific libraries we need to import while using Sagemaker.
import Boto3 - It is an open-source python API to work on AWS admin work.
import sys - sys package helps you execute some system-specific stuff.
import IPython - It helps you to access the various parts of ipynb.

```
if int(sagemaker.__version__.split('.')[0]) == 2:
    print("Installing previous SageMaker Version and restarting the kernel")
    !{sys.executable} -m pip install sagemaker==1.72.0
    IPython.Application.instance().kernel.do_shutdown(True)

else:
    print("Version is good")
```

The above code checks for the Sagemaker version. And if the Sagemaker version is 2 it installs the version of 1.72.0 and shuts down the kernel.
And if the version is not equal to 2, it prints the “Version is good"
“!” executes the code on the terminal.

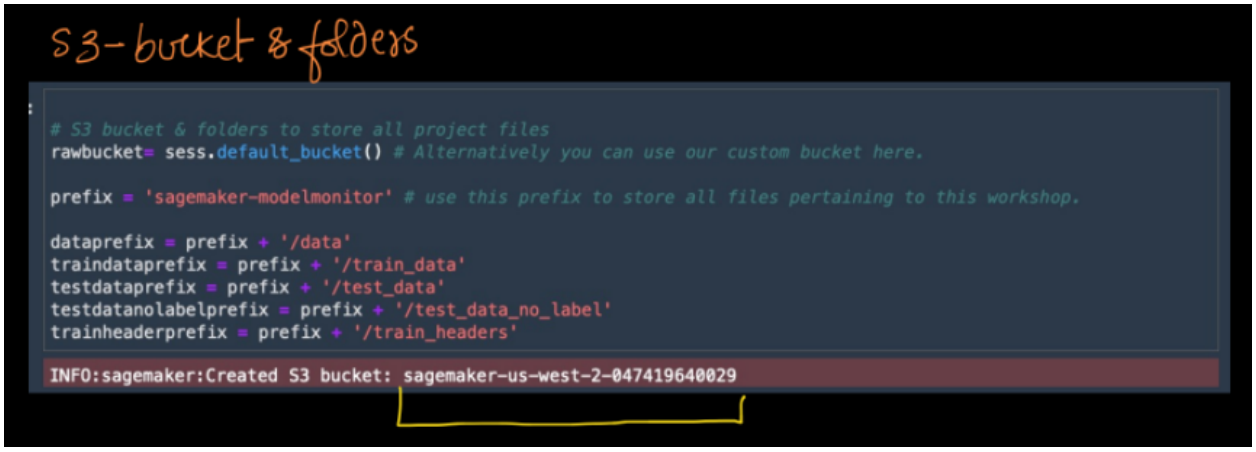
```
role = get_execution_role()
sess = sagemaker.Session()
region = boto3.session.Session().region_name
print("Region = {}".format(region))
sm = boto3.Session().client('sagemaker')
```

This code creates a Sagemaker session and displays the region of the data center.

We can import the libraries related to our ML problem. For our credit card problem, we are importing all these libraries

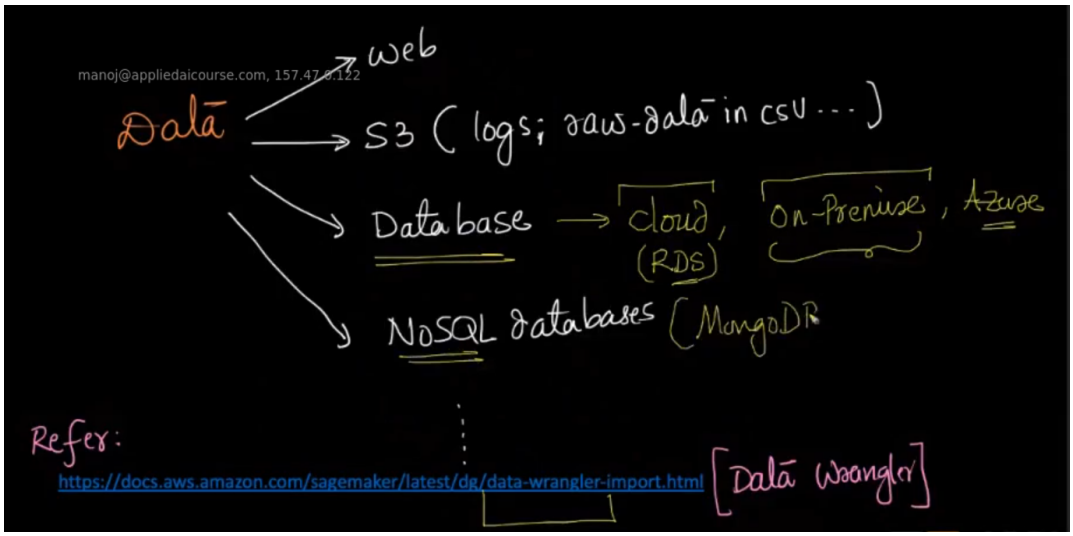
```
[1]: # import libraries
     import matplotlib.pyplot as plt
     import numpy as np
     import pandas as pd
     import os
     from time import sleep, gmtime, strftime
     import json
     import time

[2]: # Sagemaker-Experiments to track various training expeiments
     !pip install sagemaker-experiments
     from sagemaker.analytics import ExperimentAnalytics
     from smexperiments.experiment import Experiment
     from smexperiments.trial import Trial
     from smexperiments.trial_component import TrialComponent
     from smexperiments.tracker import Tracker
```

If we want to store any of our data permanently, we can do that in S3.
We can create a bucket and create some folders.
Here we have created an S3 bucket and created a folder under the name “modelmonitor” and within the folder, we have stored our data.



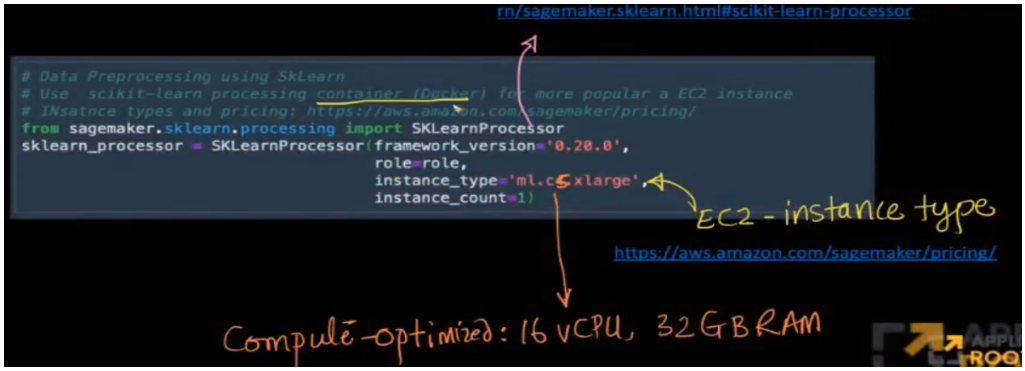
47.8 SageMaker Part 2

Data can come from various sources in various formats, this can be handled by data wrangler in Sagemaker. Some of the sources of data are listed below.



S3 is a storage service in aws. Even model data or training data which needs to be stored after the session should be pushed to an S3 instance if it is to be stored as the disk data gets wiped off after the jupyter notebook is closed.

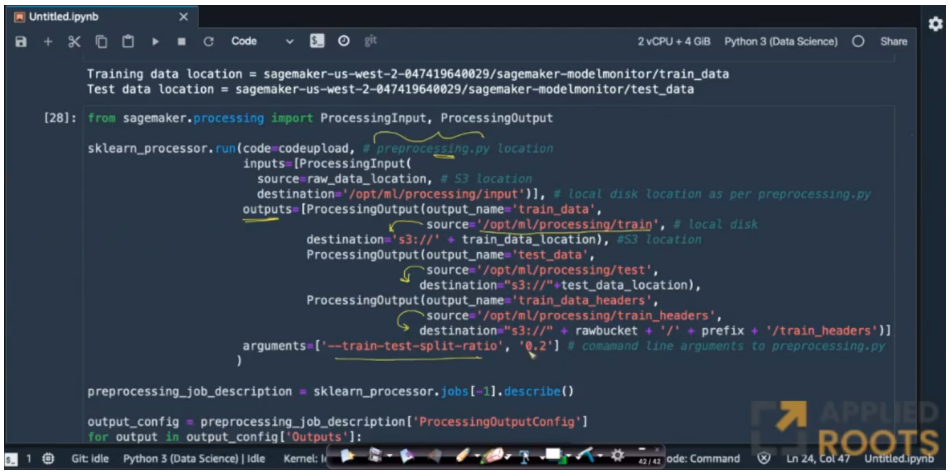
Compute intensive tasks can be done on other containers instead of the notebook and this can greatly reduce costs. A code snippet is shown below.



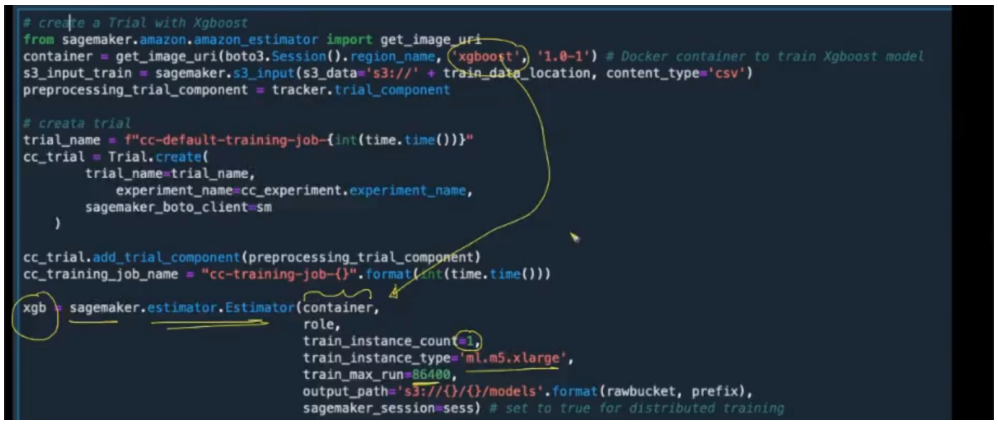
Here we are creating a heavy compute container just for preprocessing, with instance_type as ml.c5.xlarge. Sagemaker takes care of all the

dependencies installation. Building of instances with required packages is fast as Docker containerization is being used here. A config file can be used to install packages which are required.

For this newly created container the required code to execute can be sent to S3 then the container can access the code in S3. Similarly when this container is done with preprocessing the outputs of the process are to be pushed back to S3 which is accessed by notebook to get back the outputs of the job. The code below does this transfer from container to S3.



Training of models is dealt by Sagemaker Experiment. Keeping track of trails in these experiments is important to know which ideas work and which don't.

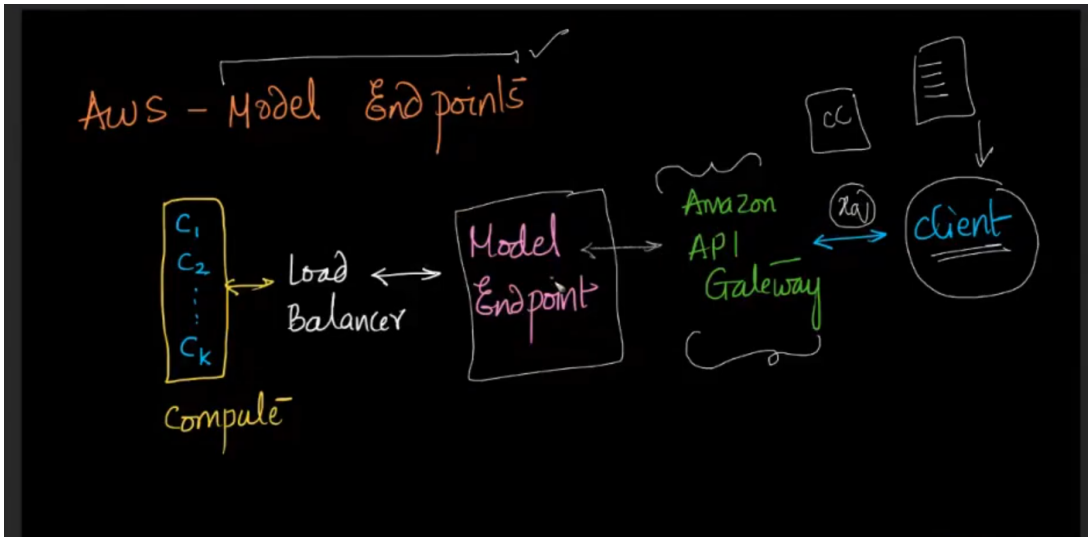


Here we are using Sagemaker Estimator to train our xgboost model again on a compute intensive container. Above is a glimpse of the code.

Batch processing means to predict using the model for whole test batch data in a single shot.

To run custom libraries in AWS we need to create a docker image with all the custom libraries and then it can be used.
Even these docker images are readily available in aws.

Real time inference is preferred to batch processing in many companies.
API can be thought of as calling a function on a remote system.



Model Endpoint here is responsible for the serving of models to the client.
Load Balancer is to even out the load to all the compute boxes so that a large number of requests can be handled effectively.

Monitoring of x_q,y_q done by the model endpoint is also possible.
Configuration of the traffic management can also be done in endpoint configuration.
Monitoring of CPU, memory usage can be helpful to know whether we are using the resources at hand efficiently or not.

```
[68]: # Call the EndPoint as if from a client

from sagemaker import RealTimePredictor
from sagemaker.predictor import csv_serializer

predictor = RealTimePredictor(endpoint=endpoint_name, content_type = 'text/csv')

with open('test_sample.csv', 'r') as f:
    for row in f:
        payload = row.rstrip('\n')
        response = predictor.predict(data=payload[2:])
        sleep(0.5)
print('done!')
```

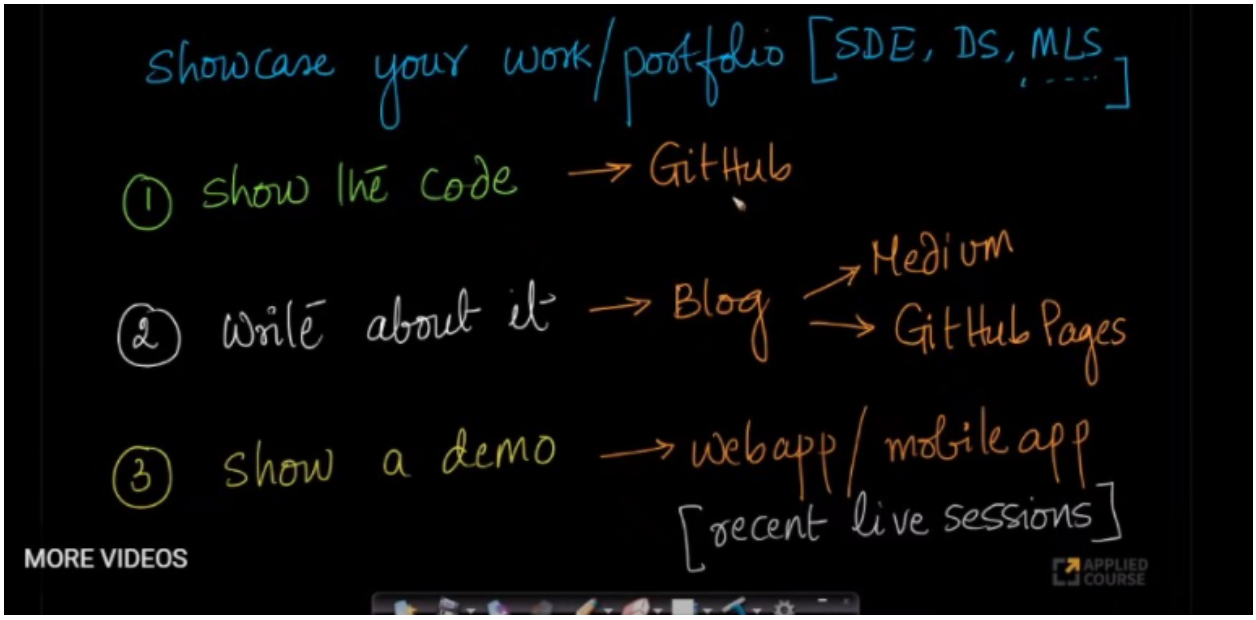
By this code we are calling the model on the data in test_sample.csv. Here RealTimePredictor means that we are calling the model on each datapoint one at a time.


```
{
  "captureData": {
    "endpointInput": {
      "data": "-0.34147611300851444,0.1932005252116958,50000.0,1.0,2.0,2.0,25.0,-1.0,3.0,2.0,0.0,0.0,0.0,10386.0,9
993.0,9993.0,15300.0,0.0,0.0,200.0,5307.0,0.0,0.0",
      "encoding": "CSV",
      "mode": "INPUT",
      "observedContentType": "text/csv"
    },
    "endpointOutput": {
      "data": "0.5108723044395447",
      "encoding": "CSV",
      "mode": "OUTPUT",
      "observedContentType": "text/csv; charset=utf-8"
    }
  },
  "eventMetadata": {
    "eventId": "5a669a54-0965-49ec-9fbd-41f4b5d5cdf8",
    "inferenceTime": "2021-08-29T12:35:14Z"
  },
  "eventVersion": "0"
}
```


Monitored data is stored in S3 in json format. A snapshot can be seen above.

For more on this topic refer to the developer guide on Amazon Sagemaker.

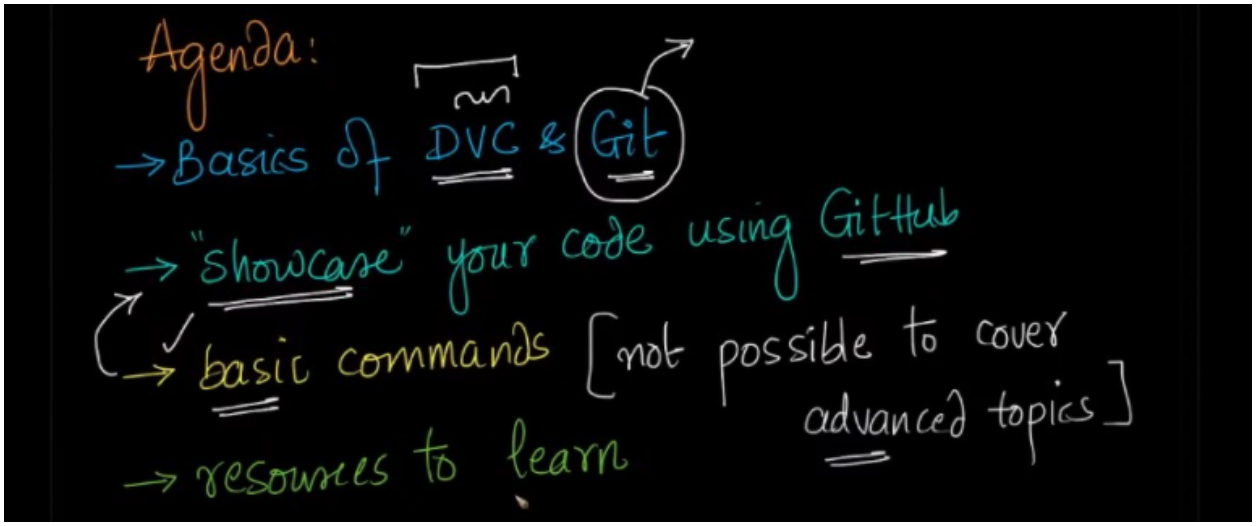
47.10 [Optional, but recommended] An introduction to Git & GitHub.



Timestamp : 02:23

In order to showcase our work on machine learning/deep learning or any field related to learning, we follow the three approaches listed below.

1.) Share the code on Github platform.
2.) Write a blog and share it on platforms like Medium,Github pages,etc.
3.) Show a demo. For example, building a web application or mobile application.



Timestamp : 06:57

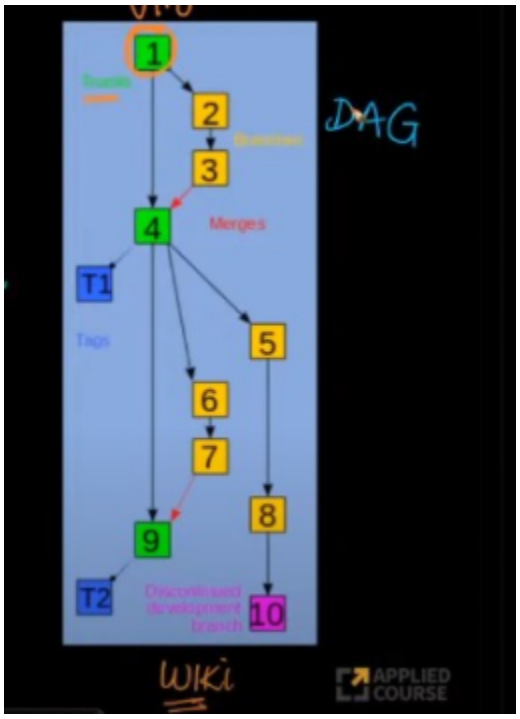
Agenda :

1.) Basics of DVC and Git . Here, we will discuss what is version control and then move on to distributed version control. We will also discuss Git.
2.) Showcase your code using Github. Here, we will discuss how to share our code on Github so the public or any restricted groups can access it and benefit from it.
3.) Basic commands. We will cover all the basic commands required to get started with github. We can't possibly cover all the commands. But, we will cover the important ones.
4.) We will point you to several resources where you can learn in depth about some of the commands or concepts.

Git : It's designed by Linus Torvalds in 2005. It was built to manage the huge code repository of the linux operating system. It's also used by many companies to manage their code repositories.

Next, we will learn about version control.

What is version control ?



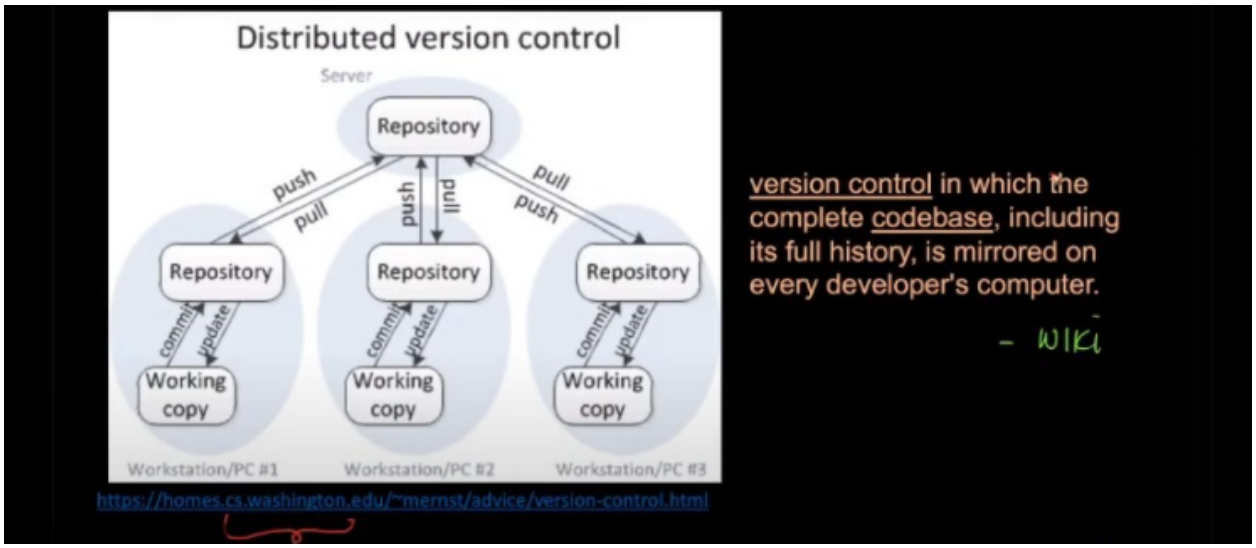
TImestamp : 10:12

Let’s say a team is going to build a software. Let’s also assume we have the software with version 1.0. (Highlighted in green color) . In the future, we may want to modify it to satisfy the requirements. So, other members in the team may work on improving the version 1.0. Here, what we will do is that we will clone the software with version 1.0 and then update the changes. If the changes reflected are validated by the community and if passed, then we make the changes to version 1.0 . If the tests are not passed, then we go back to the older version itself. The box with label 2 and 3 denotes the branch taken from the root or trunk which is the box with label 1. We have the root or trunk which is the base version of the software. To modify and update it, we clone and make some branches. Once these branches are validated, we update the root and create a new root which is the box labelled as 4. Again, we can make some changes to the new version i.e, by creating branches and then updating the root one. In some cases, we may not be able to update certain functionalities due to many reasons. So, we simply ignore that branch or discontinue from it. This is labelled as 10. Only, the branches which pass all the tests are used to update the root. The

other ones are simply discontinued. Finally, we create a new version which is labelled as 9. We have other two boxes which are labelled as T1 and T2. This simply denotes that it is the final one and it's not modified further.

Version control system simply helps the programmers community to collaborate and create softwares with the above mentioned functionalities. Git is one of those platforms which implements a version control system. As the name suggests, it simply controls the version of the software.

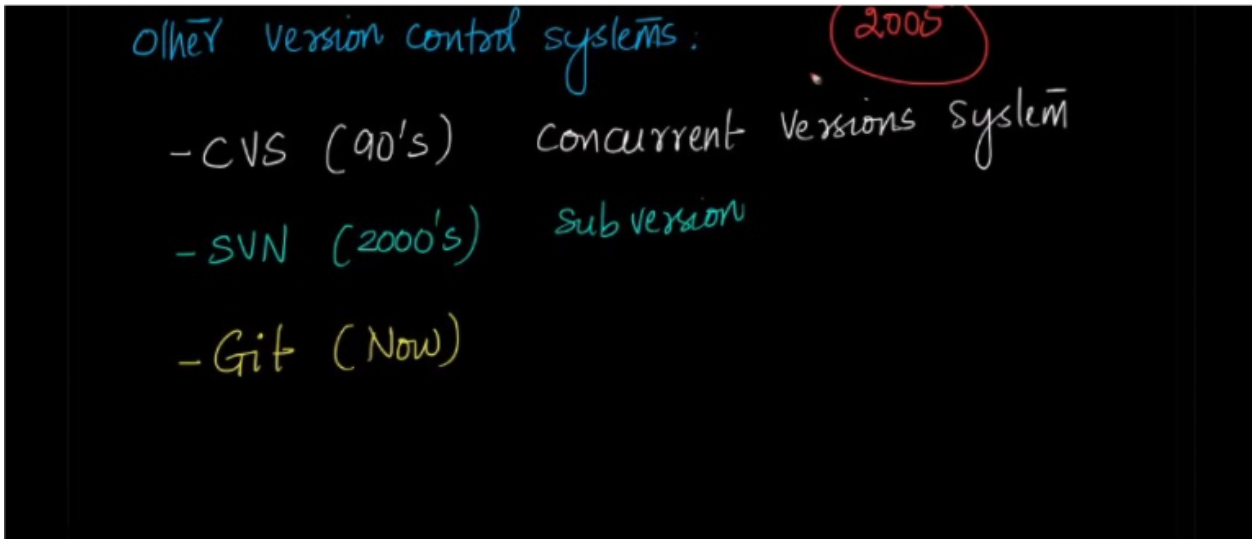
What is a distributed version control system ?



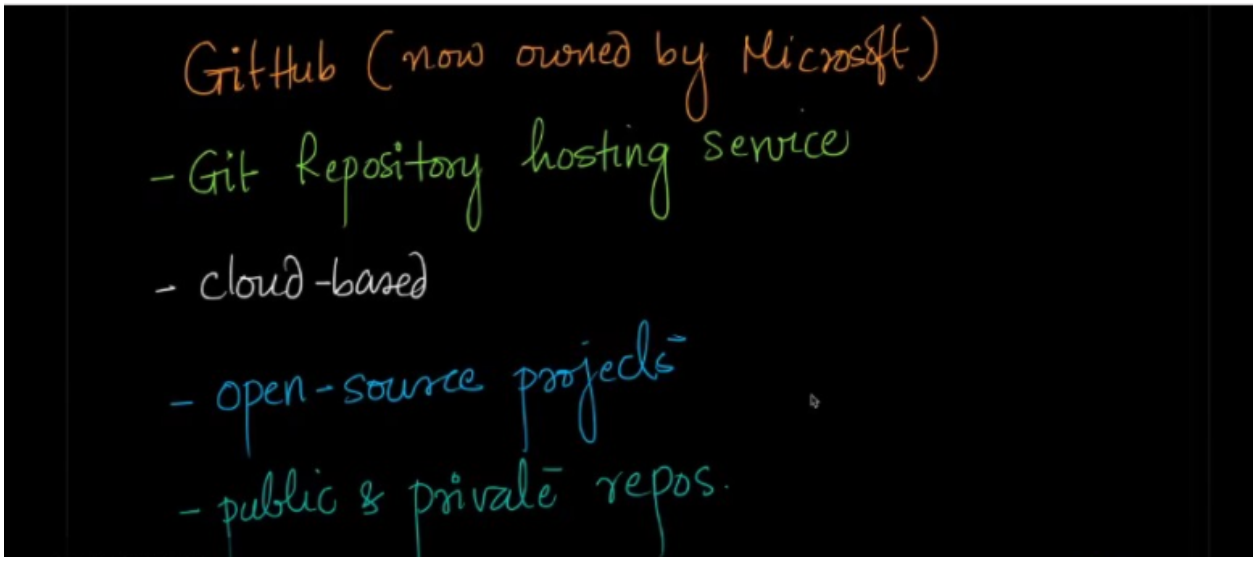
Timestamp : 15:03

Each developer is independently working on the software in their local machine. If a developer thinks that the changes can be updated, then he saves the changes by committing it. To reflect the changes on the other machines where other developers work, the changes are pushed to the server. By this, the other developer can see the changes and the codebase gets updated. The other developers can pull the changes to the codebase where the other developers pushed their changes. Then, they can update the changes with the update operation.

What are the other available version control systems before Git has been introduced ?



Timestamp : 17:36



Timestamp : 18:53

GitHub : Git is a technology or to be precise a version control system software. GitHub is a cloud-based platform where we can host github repositories. In a nutshell, the server in the previous block diagram is GitHub. GitHub is now owned by Microsoft. Mostly it's well-known for

hosting open-source projects. It also enables developers to create and host public and private repositories.

Note : The graph which we showed while discussing Git is actually a directed acyclic graph.

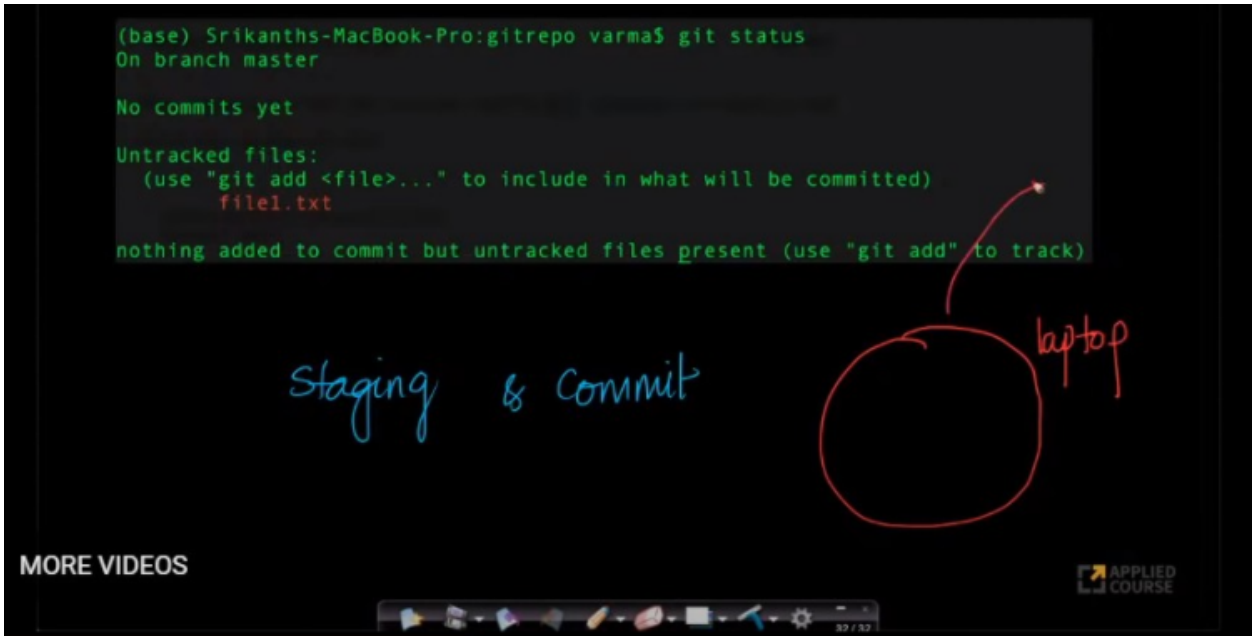
How to use GitHub ?

1.) By using github desktop software. It's a GUI based tool. To know more about this, please refer to this
2.) Command line tool.

In this session, we will focus on command line tools.

How to set up the command line system ?

1.) Please visit this link.
2.) We will discuss the commands in the context of the linux operating system.
3.) First, open the terminal.
4.) Type "mkdir gitrepo" and press Enter. (The string which is enclosed with " " is the command). It will create a directory named gitrepo. In this directory we will store all the code related files.
5.) Then, type "cd gitrepo". This command will change the current directory to gitrepo folder.
6.) Type "git init" . This command will initialize the current directory as a git repository. Now, it's not a normal directory anymore.
7.) Next, we will create a file named file1.txt in the current directory using the command "touch file1.txt" .
8.) To check whether the file has been successfully created, we can use the "ls -lrt" command to view all the files in the current directory.
9.) Then type "git status".



Timestamp : 28:59

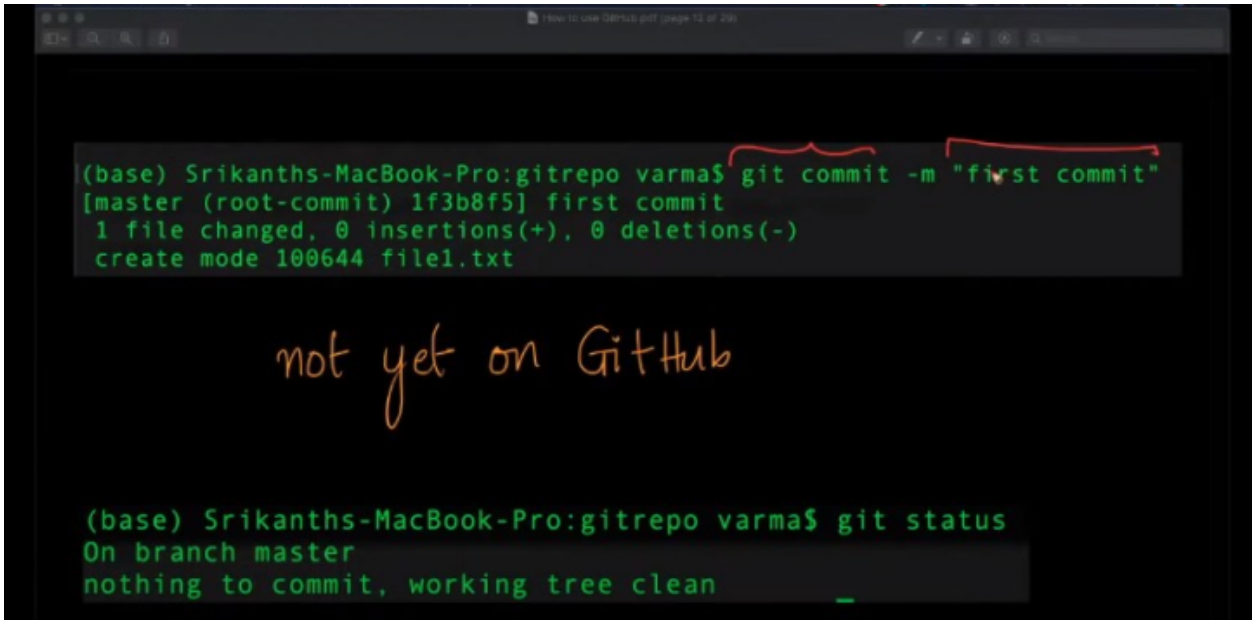
This command simply shows the status. It says “No commits yet”. This means, we didn’t commit any changes we made. It’s on the branch master. This is the root one. As we already discussed, we have a root branch and then from it, we can create as many branches we want. We also added a file named file1.txt in this master branch. But, we didn’t commit to it. So, it’s untracked.

Staging : We are first informing the git that we have some files in our directory which we are going to commit.

Commit : Simply committing to the local system.

To perform the staging operation, type “git add” . This command will simply add all the files in the current directory to the repository.

To perform the commit operation, type “git commit -m “first commit” . This command will commit all the files which were added previously. -m denotes a message.



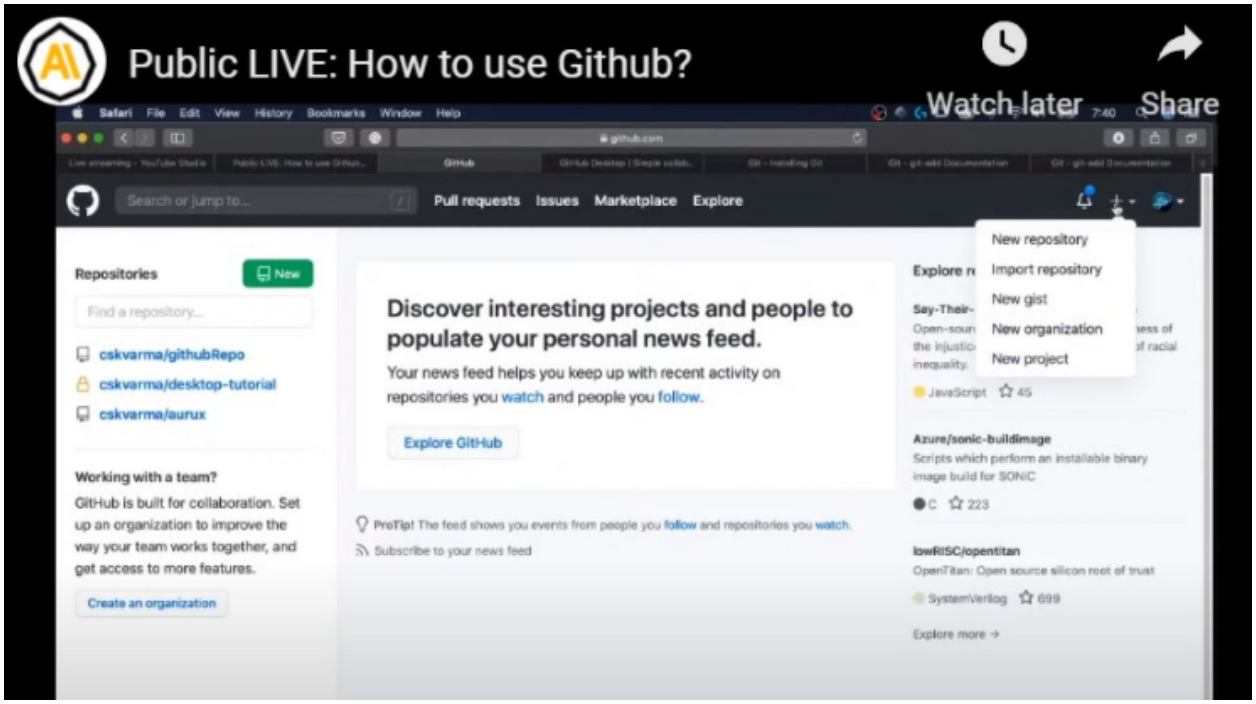
Timestamp : 32:28

We still didn't push anything to github. Everything lies in the local repository.

Till now, we had only a local repository i.e, all the files still reside in our local machine. To make it public or for some restricted community, we need to push it to github.

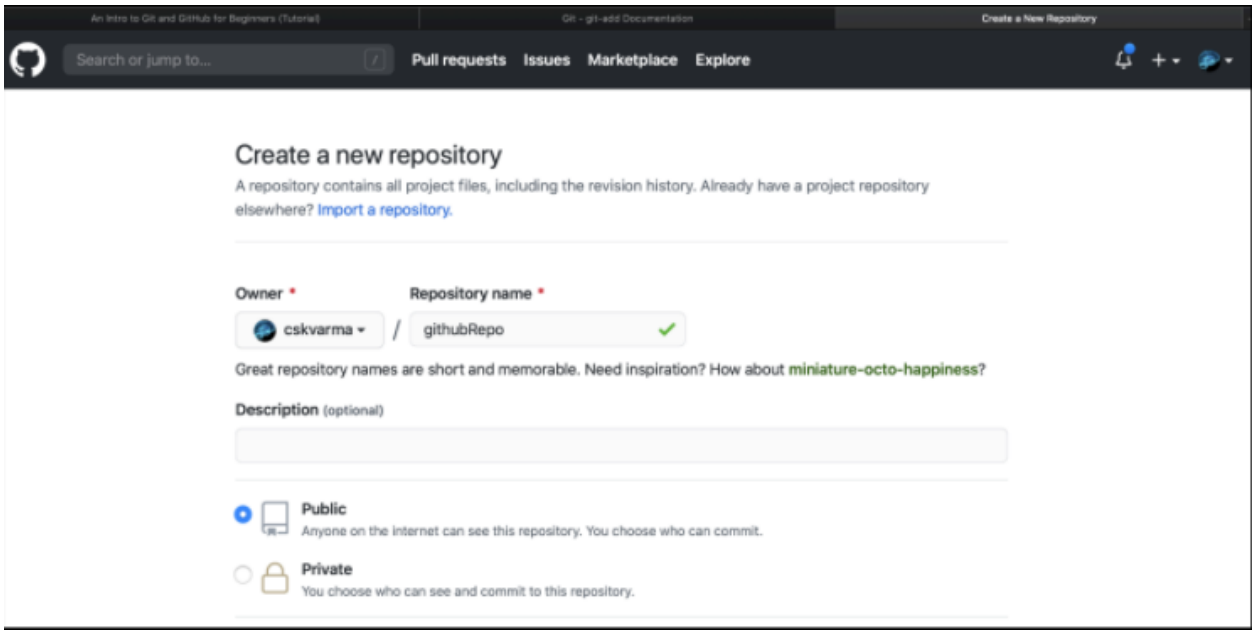
Steps to get started.

1.) Visit this website and create a free account.



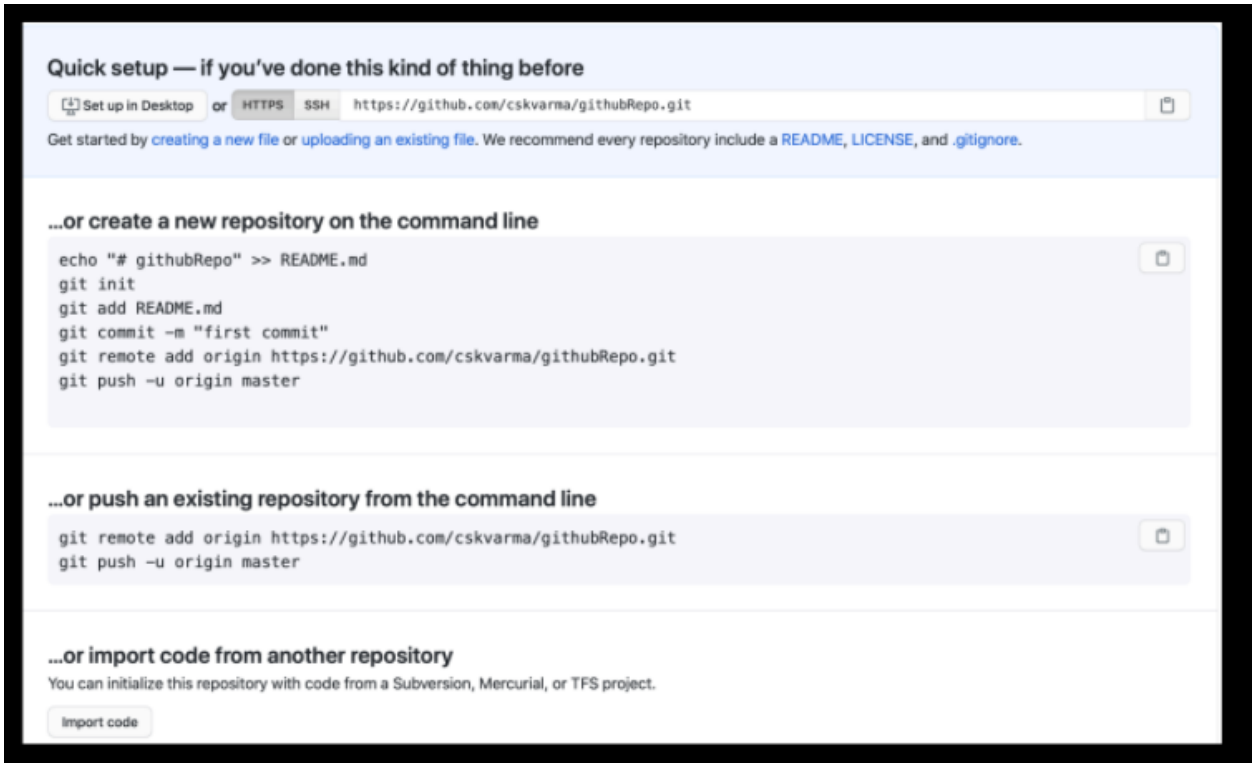
Timestamp : 34:26

2.) Then, click on the icon with “+” and choose “New repository”.



Timestamp : 35:03

Then , fill the fields. If you choose the option which is labelled as Public, then the repository created will be available to the public and anyone can access it. If you go with Private, then only a restricted community can access it.



Timestamp : 35:16

Here, we want to perform the second thing mentioned above. This is because we already have a local repository as we have created it before.

Next, simply type the command mentioned above in the terminal in sequence.

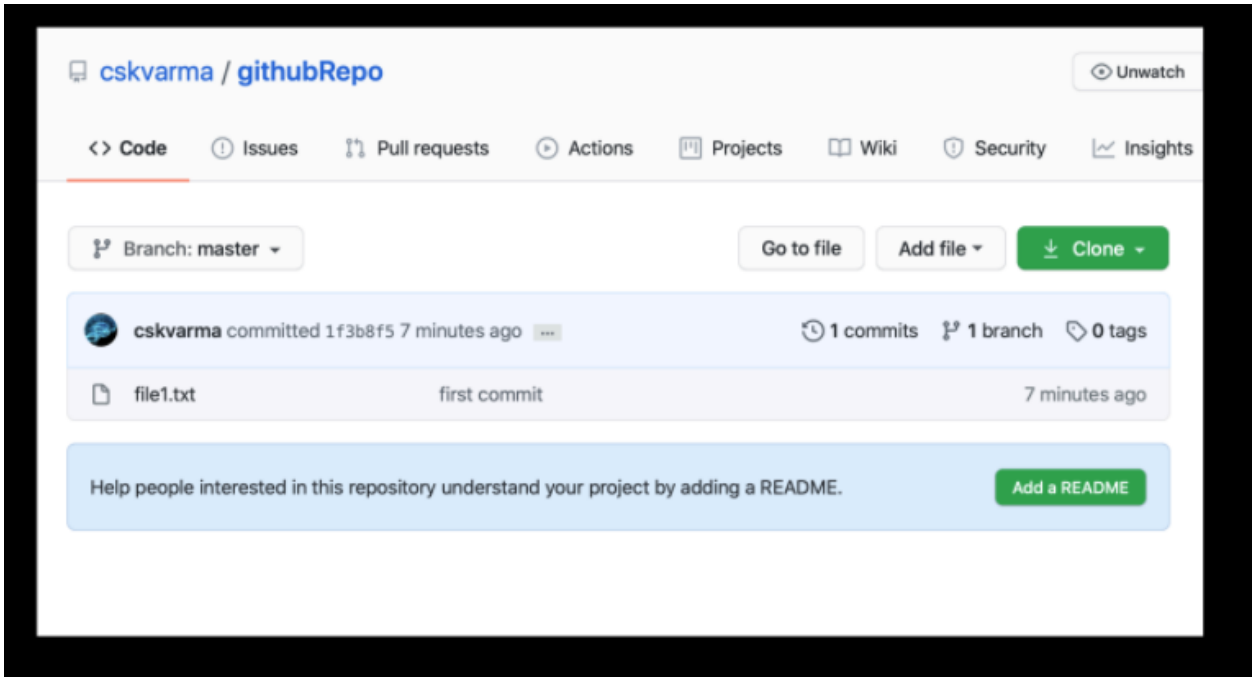
```
(base) Srikanths-MacBook-Pro:gitrepo varma$ git remote add origin https://github.com/cskvarma/
githubRepo.git
(base) Srikanths-MacBook-Pro:gitrepo varma$ git push -u origin master
Username for 'https://github.com': cskvarma
Password for 'https://cskvarma@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 207 bytes | 207.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/cskvarma/githubRepo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
(base) Srikanths-MacBook-Pro:gitrepo varma$
```

Timestamp : 37:16

Remote simply means, we are accessing a remote server which is the github server.

Origin is the alias to the url.

Next, we push all the files to the remote repository by the command “git push -u origin master”.



Then, we can see all the files in the remote repository.

Suppose, we want to create a new file.

We do this by typing the command “touch file2.txt”.

Then, we push the file to the remote repository by typing the command “git push” . We can also do this by using the command “git push origin master”.

Timestamp : 40:34

We can also do the same as mentioned below.

Timestamp : 41:42

We are simply replacing the origin with the entire url. We already mentioned that origin is the alias to the url.

To know what origin really is , type the command “git remote -v” . Here -v stands for verbose. This means that we are saying to provide information about the remote repository. We can see that origin is the alias to the above-mentioned url.

Let’s say a developer wants to pull some files from the server to the local machine. This operation is named as fetch. This is called branching too. The developer wants to make some changes to the existing files residing in the remote repository. So, the developer can branch the main root and make some changes. If the changes are passed by the testing team, then the changes get reflected in the main branch.

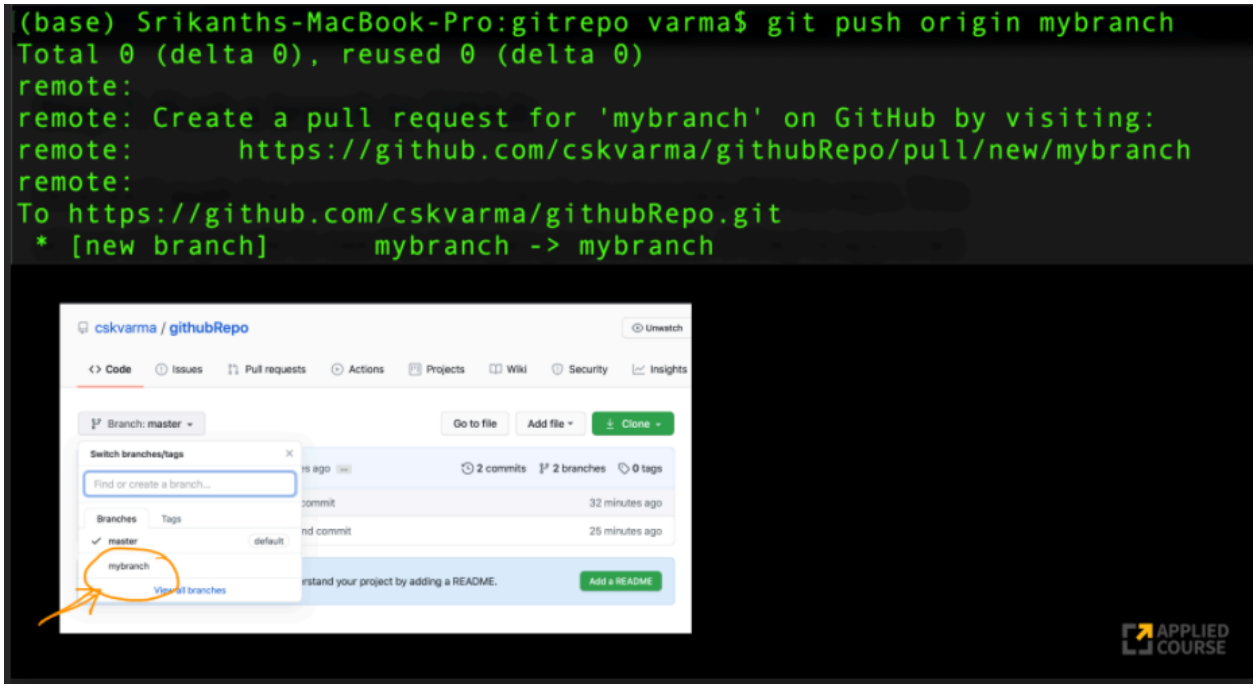
How to do it ?

Timestamp : 43:19

Type the command “git fetch”

Then , type “git checkout -b mybranch” . This command simply creates a new branch named mybranch under the master. So, changes made to the file are not reflected in the master branch as we created a new branch. Only if we commit it, it will get reflected. To be simple, we are cloning the file and making some changes.

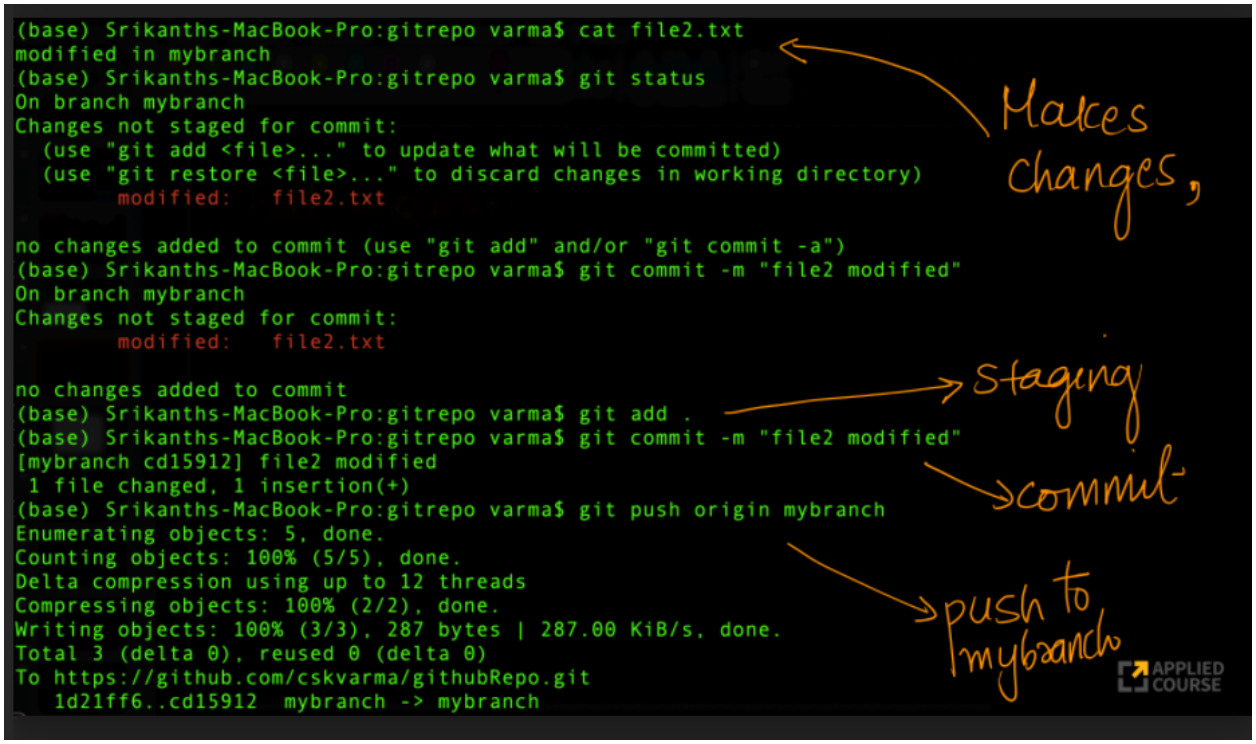
To know about the branches we created, type “git branch”.



Timestamp : 45:38

When you type “git push origin mybranch” , the branch named “mybranch” will get reflected in the remote repository. In the previous command, we created it on our local machine only. With this command, we are also making the changes in the remote repository.

We can see in the highlighted figure that the branch has successfully pushed in the remote repository.



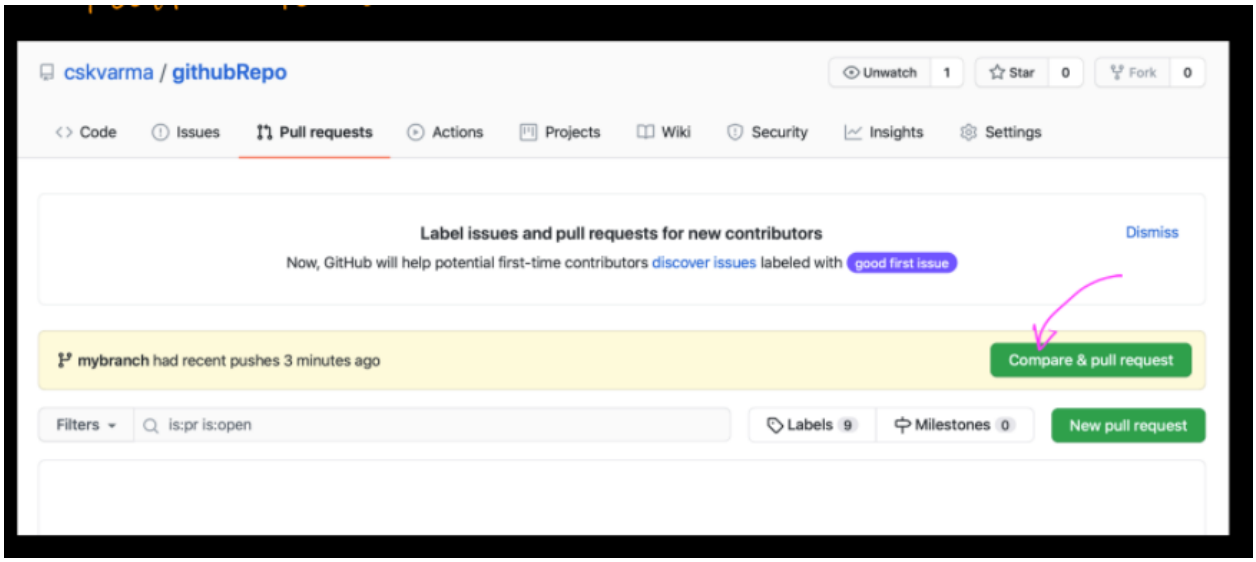
Timestamp : 46:51

1.) Open the file named file2.txt and make some changes like adding some text. cat command is used to view the file.
2.) Then type “git status” to check the status. Since we made some changes and not yet committed it, it shows the corresponding message.
3.) Then, type “git add” to add the file or to perform the staging operation.
4.) Then, type “git commit -m “file 2 modified” . This will commit the changes.
5.) These changes only get reflected in the local repository. It’s not yet been reflected in the remote repository.
6.) Then type “git push origin mybranch” to push the branch to the remote repository.
7.) We did four steps right ?

1.) Make changes
2.) Staging
3.) Commit

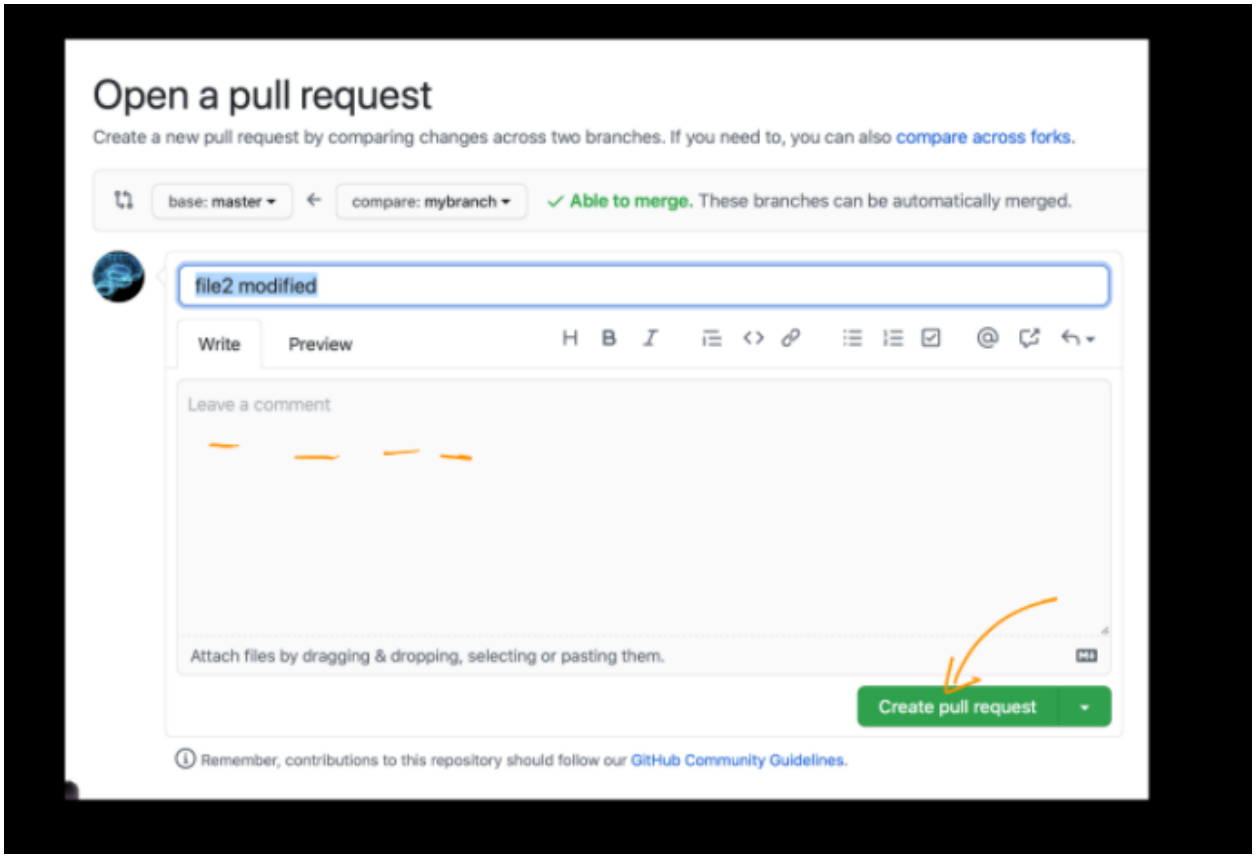
4.) Push

Pull Request : We already made some changes to the mybranch which is a branch we created from the master. But, since the changes are passed, we need to update the master branch. This is called pull request.



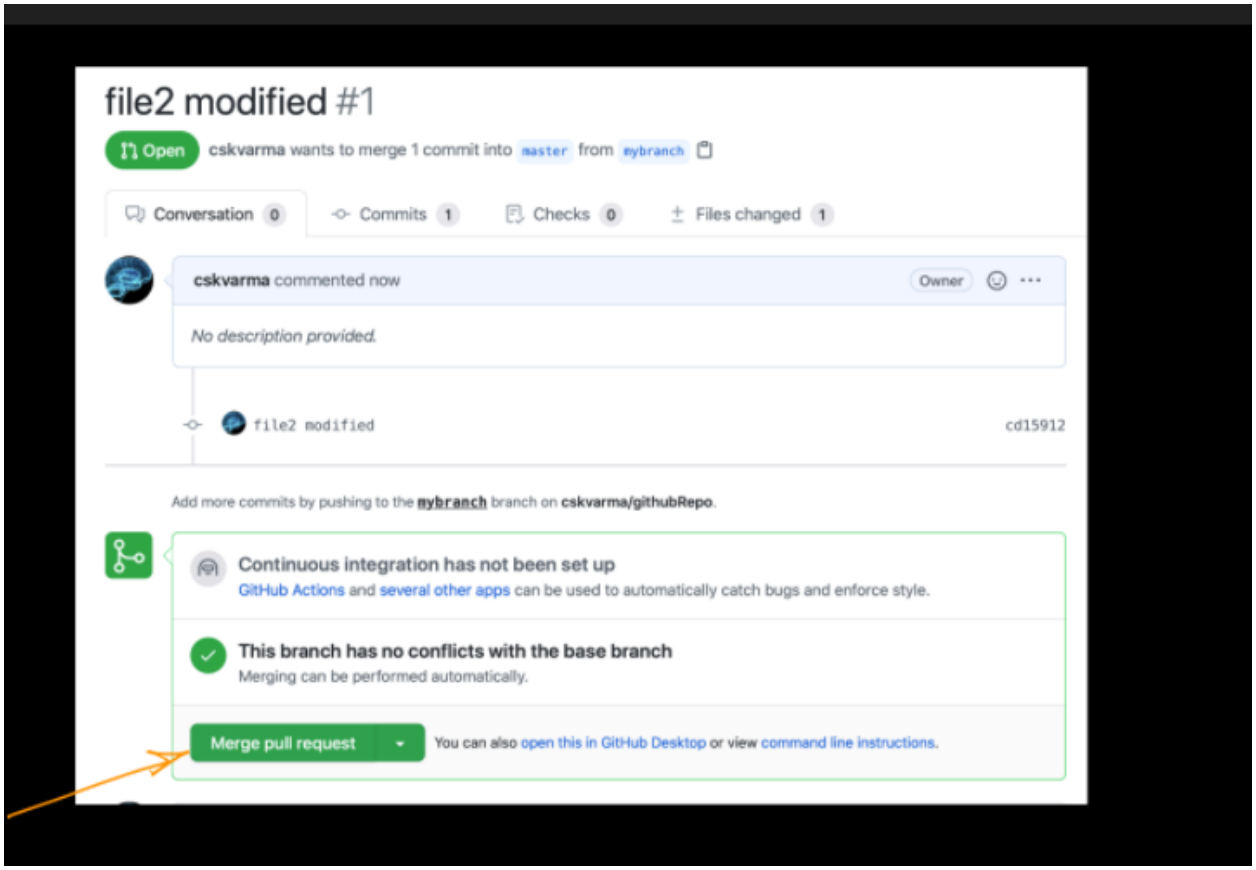
Timestamp : 50:13

Click on “Compare and pull request”.



Timestamp : 50:15

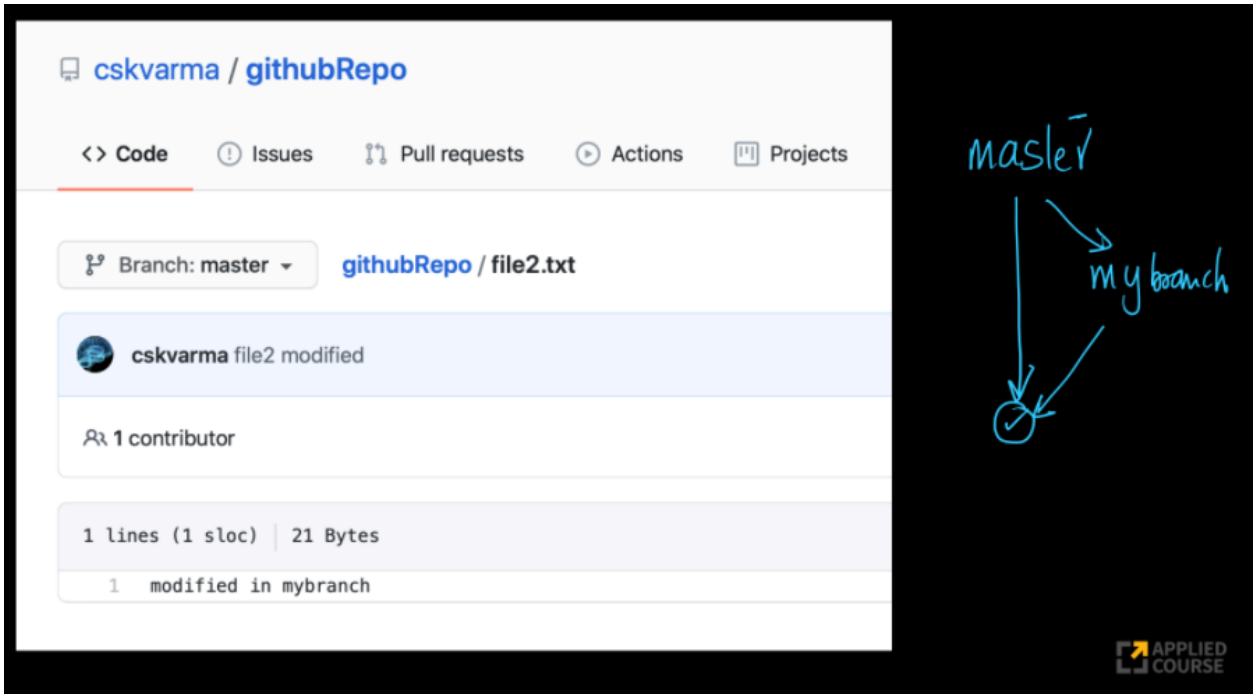
Then, click on “Create pull request” . We are simply merging the branch created by one of the developers from the master branch to itself.



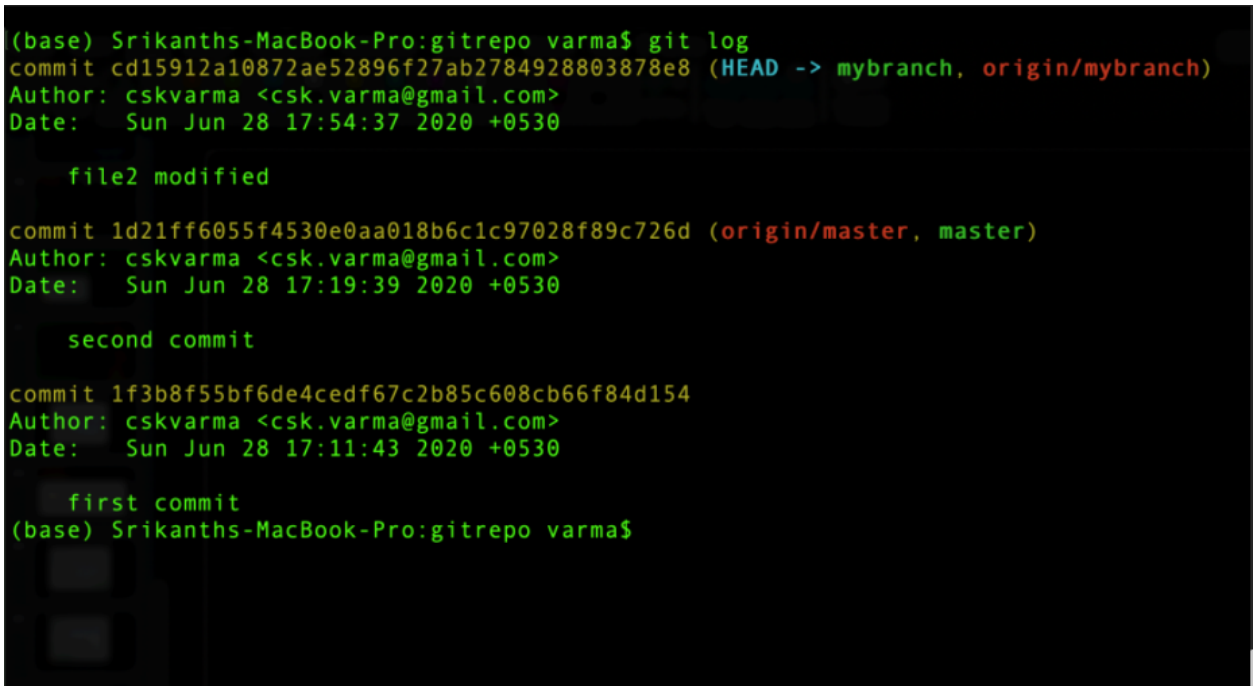
Timestamp : 50:54

Then, the other developers working on the same thing also need to acknowledge the changes.It can be done by clicking on the “Merge pull request” button.

Now, if we visit the master branch, then we can see the changes made to the file2.txt.

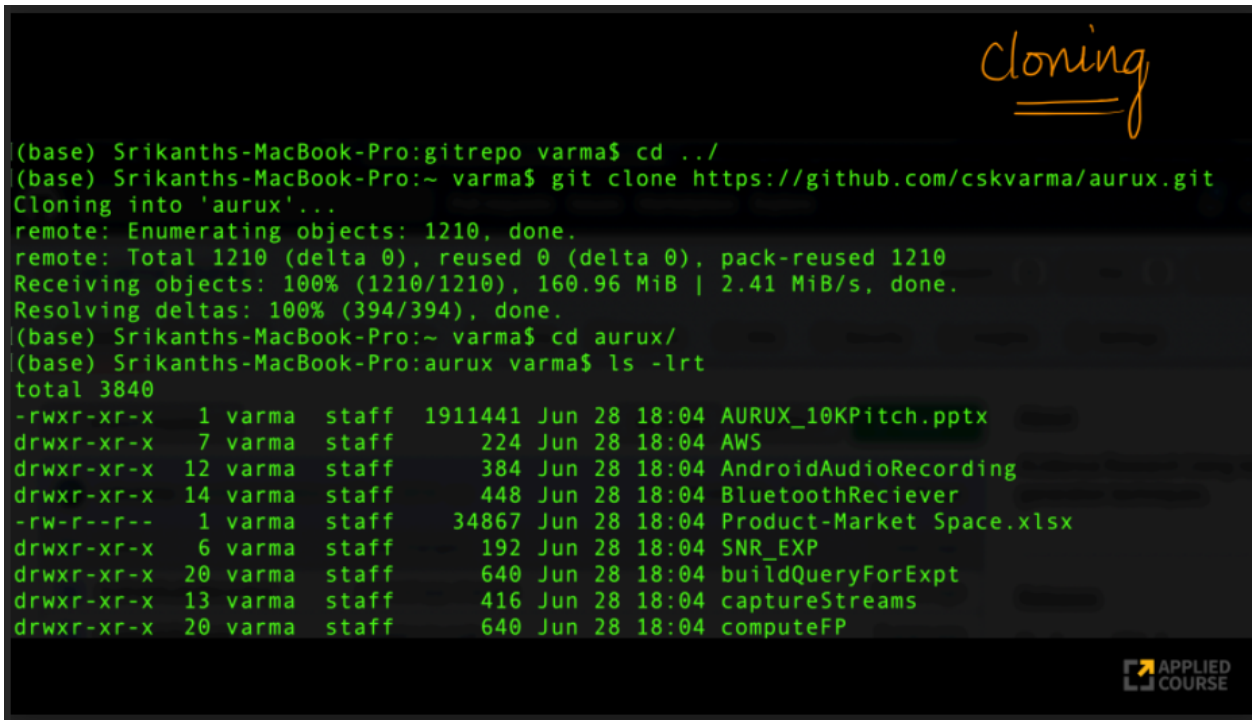


Timestamp : 51:09



Timestamp : 51:42

You can view all the things we have done to our repository by typing the command “git log”.



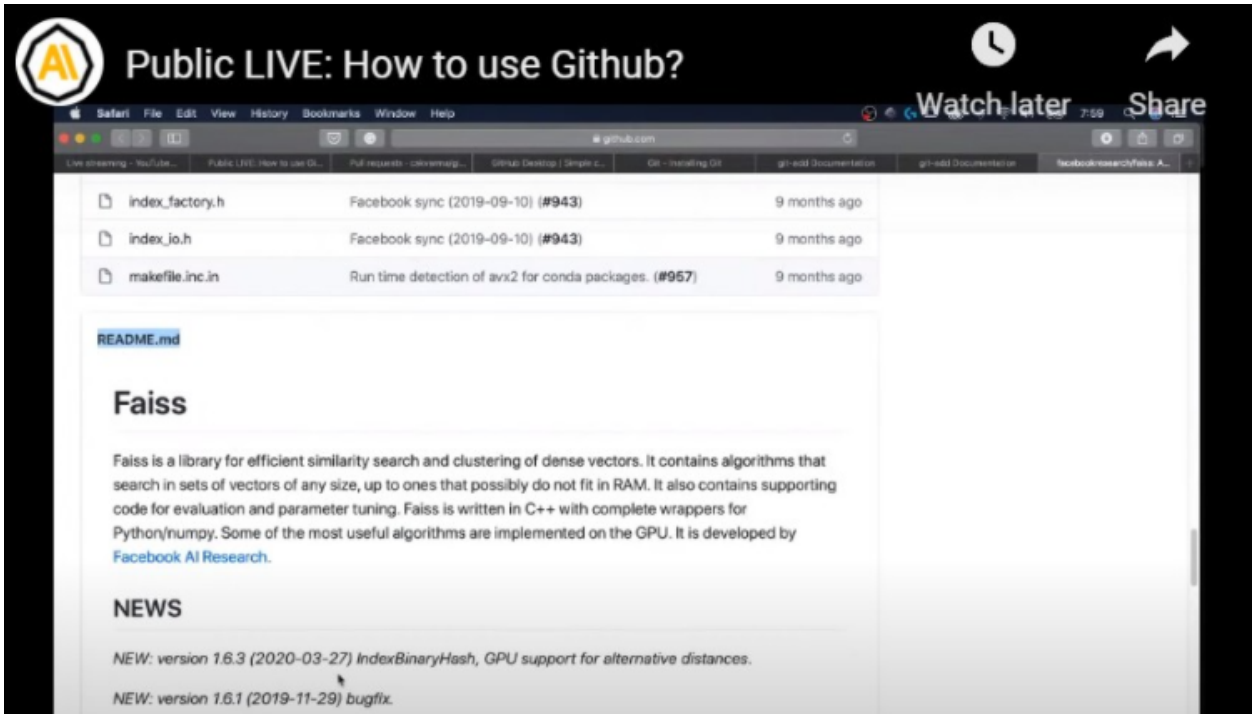
Timestamp : 52:10

Let’s say we are already having a remote repository. If we want to have a copy of it in our local machine, we need to clone it right ?

To perform this, we type the command “git clone url. Here, url depends on your remote repository.

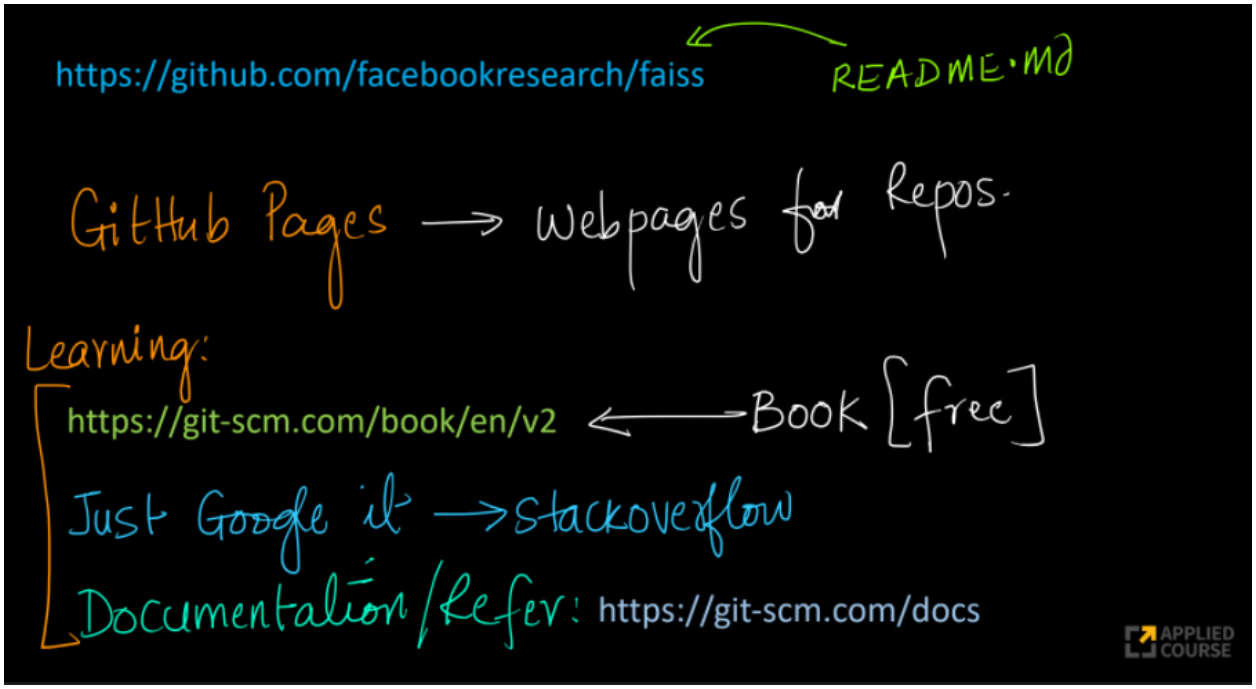
README.md :

In this section, we can give an introduction about our work.



Timestamp : 54:15

Github Pages : We can build static websites using this tool to showcase our work.



Timestamp : 56:14