2022

Final Project - AWS Machine Learning

BDM 1034 APPLICATION DESIGN FOR BIG DATA

GROUP E

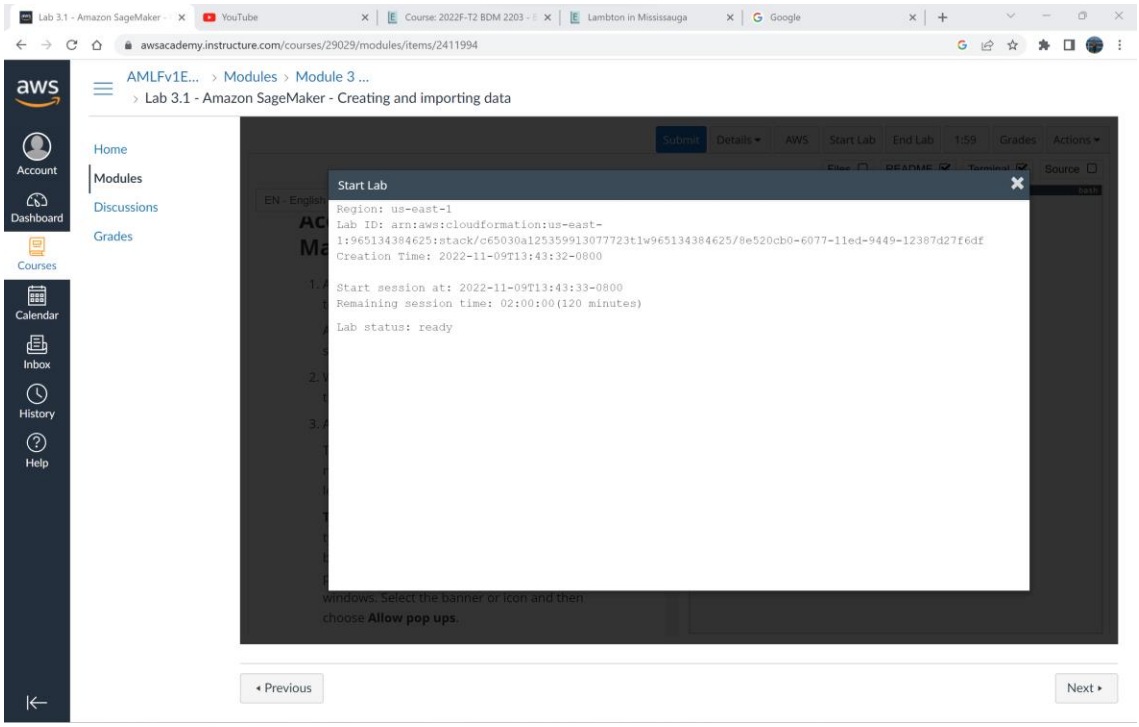
Table of Contents

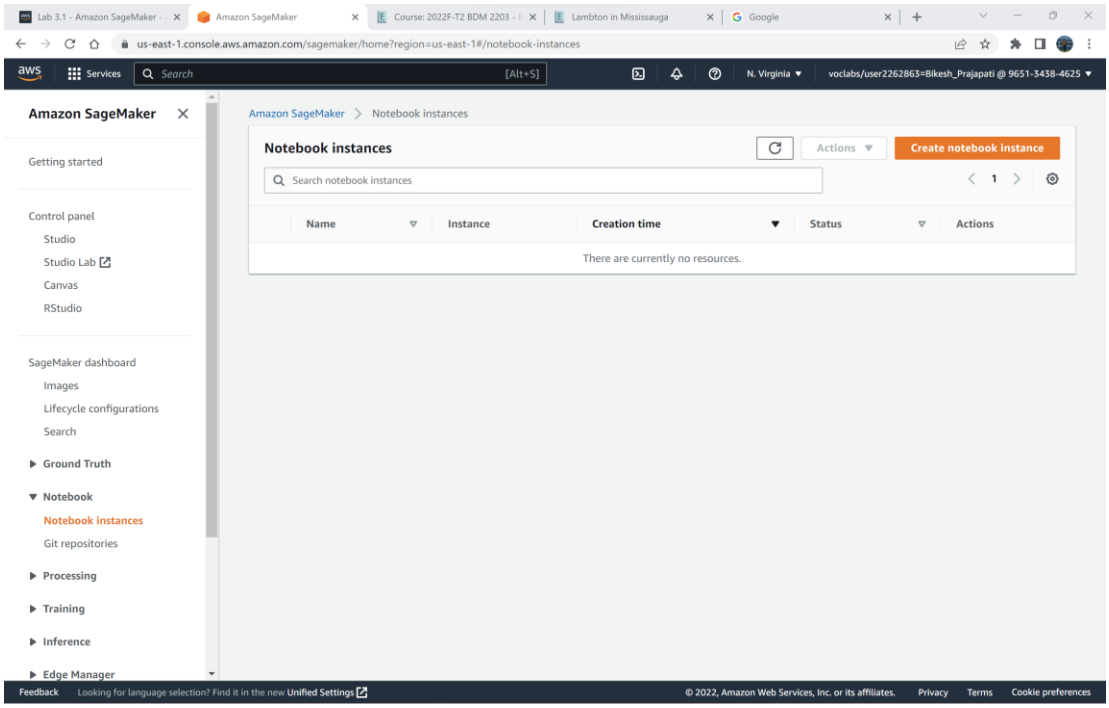
Group Members who have shown participation

Student Name	Student ID
Bikesh Prajapati	C0859472
Padam Regmi	C0858265
Indrani Das	C0847084
Dolsy Arora	C0859753
Kripa Rachel Thomas	C0858713

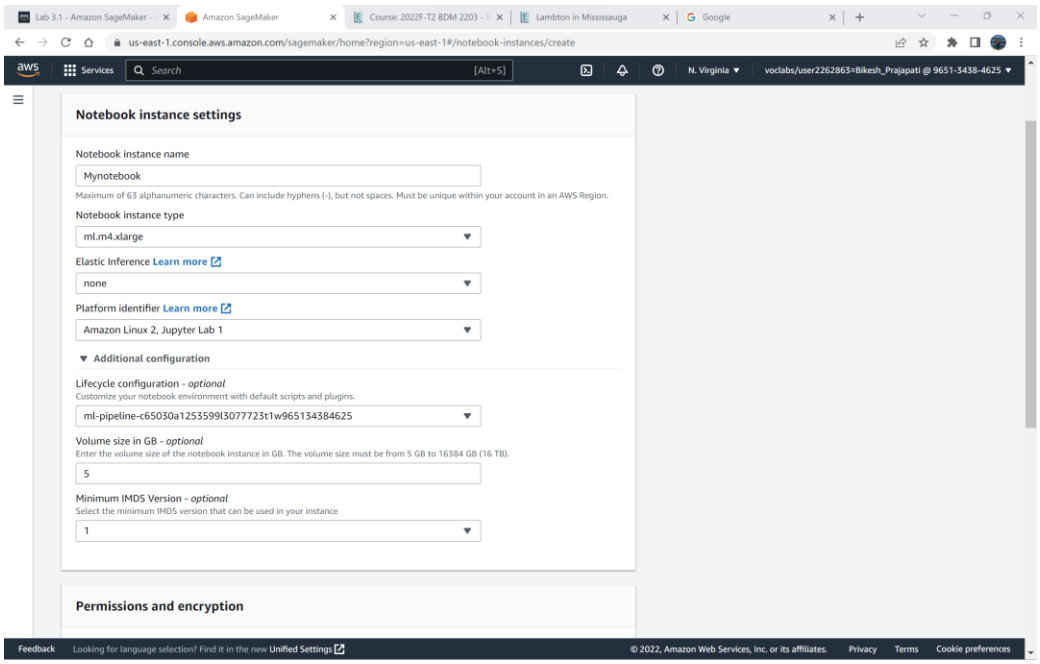
LAB 3

Lab 3.1 - Amazon Sage Maker - Creating and importing data

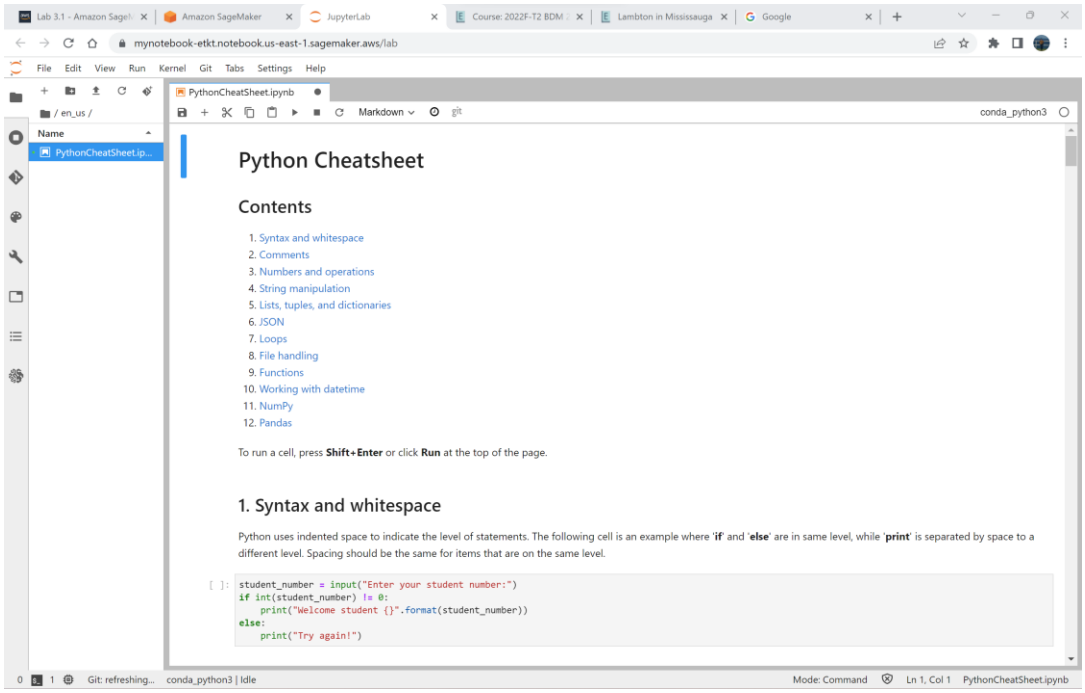




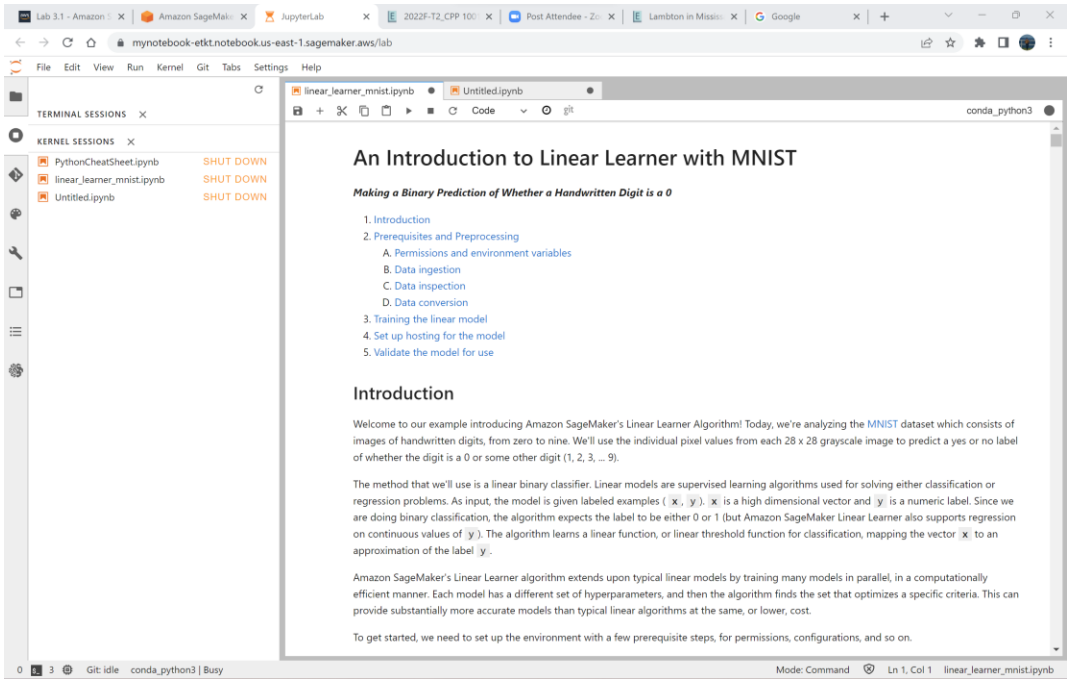
First of all, for this lab we are importing and creating the instances for the notebook on Amazon sage maker.

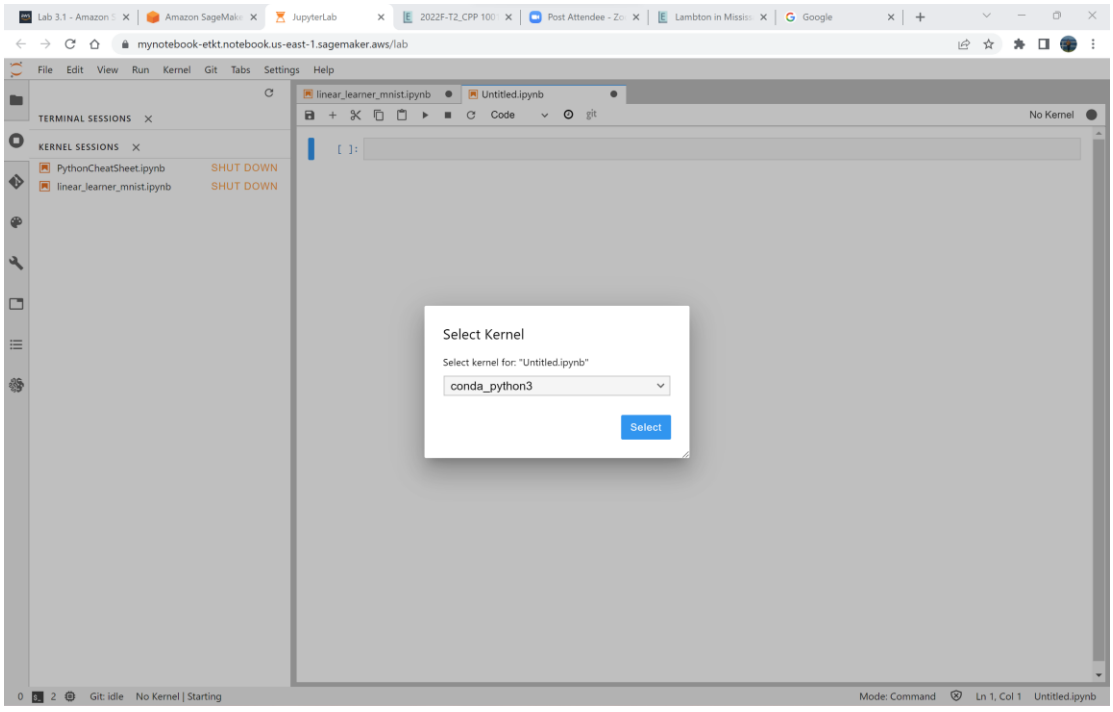


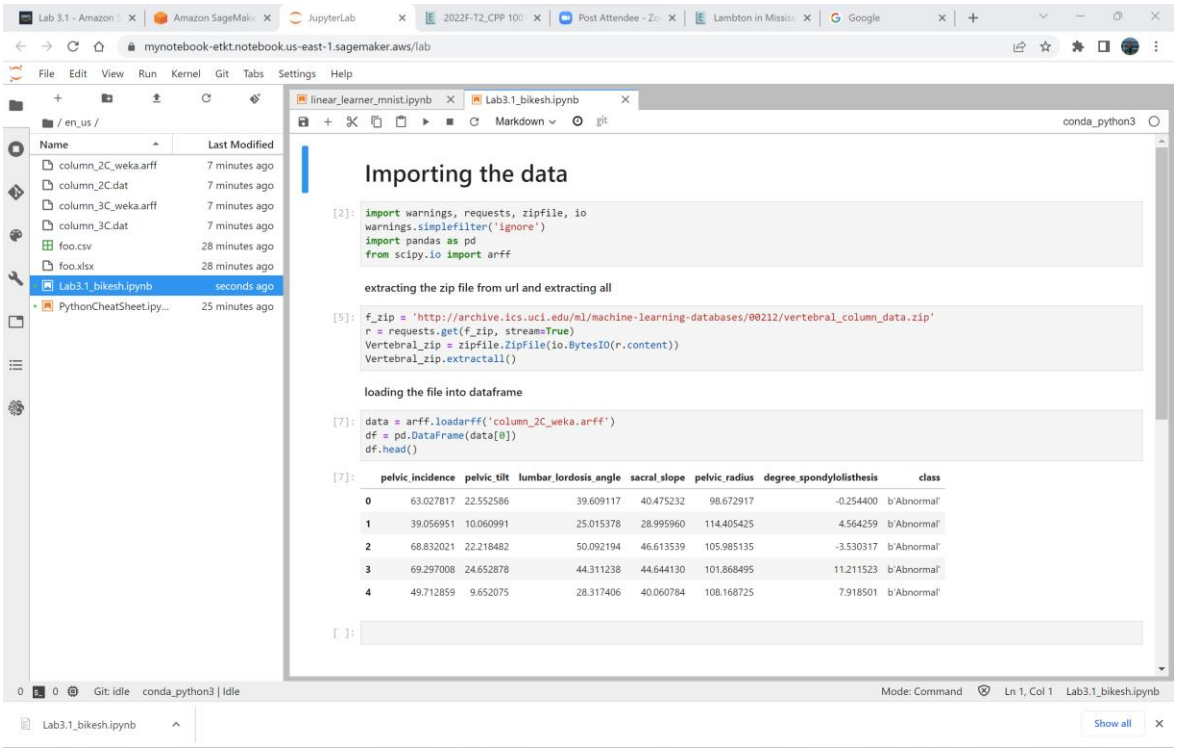
In this section, we fill out all the necessary requirements that have been mentioned on the guideline of lab. Such as naming, choose ml.m4.xlarge as notebook type, configure the pipe line and identify the platform.



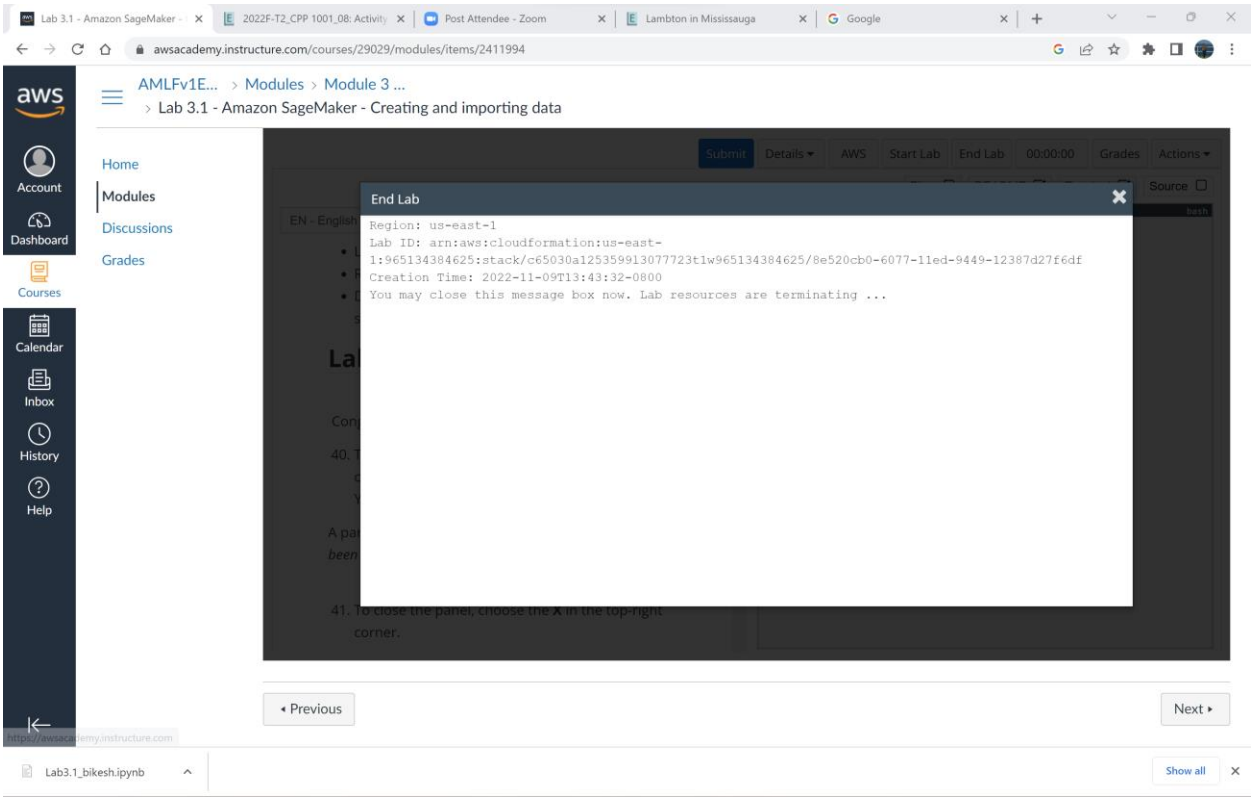
After that, we full up our first lab python code and start to deploy the code. Since the code are already prepare we just need to run the code and see the output.


mynotebook-etkt.notebook.us-east-1.sagemaker.aws/lab
File Edit View Run Kernel Git Tabs Settings Help
TERMINAL SESSIONS
KERNEL SESSIONS
PythonCheatSheet.ipynb SHUT DOWN
linear_learner_mnist.ipynb SHUT DOWN
Untitled.ipynb SHUT DOWN
linear_learner_mnist.ipynb
Untitled.ipynb
Code
conda_python3
An Introduction to Linear Learner with MNIST
Making a Binary Prediction Whether a Handwritten Digit is a 0
1. Introduction
2. Prerequisites and Preprocessing
A. Permissions and environment variables
B. Data ingestion
C. Data inspection
D. Data conversion
3. Training the linear model
4. Set up hosting for the model
5. Validate the model for use
Introduction
Welcome to our example introducing Amazon SageMaker's Linear Learner Algorithm! Today, we're analyzing the MNIST dataset which consists of images of handwritten digits, from zero to nine. We'll use the individual pixel values from each 28 x 28 grayscale image to predict a yes or no label of whether the digit is a 0 or some other digit (1, 2, 3, ... 9).
The method that we'll use is a linear binary classifier. Linear models are supervised learning algorithms used for solving either classification or regression problems. As input, the model is given labeled examples (x , y). x is a high dimensional vector and y is a numeric label. Since we are doing binary classification, the algorithm expects the label to be either 0 or 1 (but Amazon SageMaker Linear Learner also supports regression on continuous values of y). The algorithm learns a linear function, or linear threshold function for classification, mapping the vector x to an approximation of the label y .
Amazon SageMaker's Linear Learner algorithm extends upon typical linear models by training many models in parallel, in a computationally efficient manner. Each model has a different set of hyperparameters, and then the algorithm finds the set that optimizes a specific criteria. This can provide substantially more accurate models than typical linear algorithms at the same, or lower, cost.
To get started, we need to set up the environment with a few prerequisite steps, for permissions, configurations, and so on.
Git: idle conda_python3 | Busy
Mode: Command Ln 1, Col 1 linear_learner_mnist.ipynb

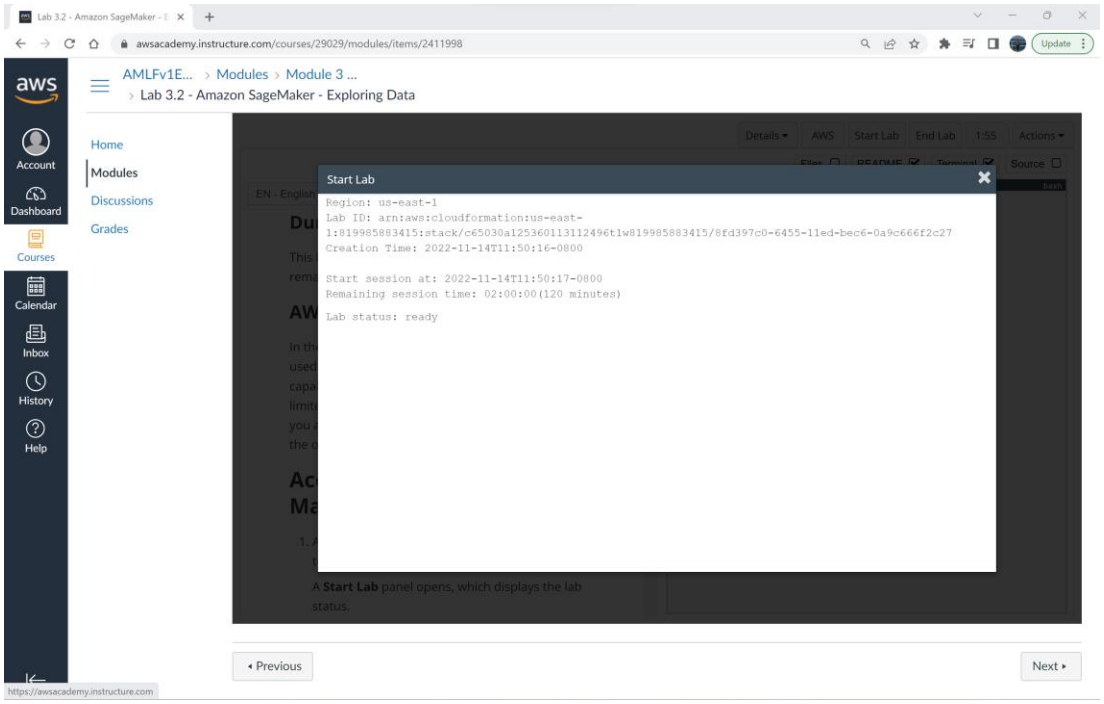

mynotebook-etkt.notebook.us-east-1.sagemaker.aws/lab
File Edit View Run Kernel Git Tabs Settings Help
TERMINAL SESSIONS
KERNEL SESSIONS
PythonCheatSheet.ipynb SHUT DOWN
linear_learner_mnist.ipynb SHUT DOWN
linear_learner_mnist.ipynb
Untitled.ipynb
Code
No Kernel
[ ]:
Select Kernel
Select kernel for: "Untitled.ipynb"
conda_python3
Select
Git: idle No Kernel | Starting
Mode: Command Ln 1, Col 1 Untitled.ipynb

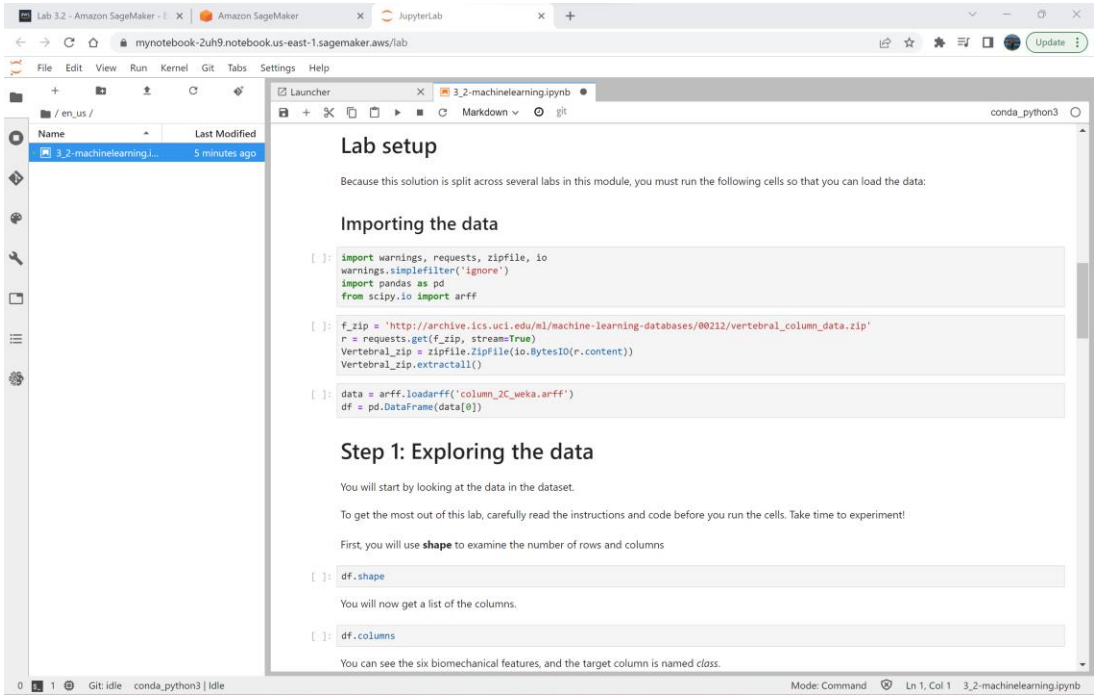


Our first task it to importing the data. we are extracting all form the zip files that are located in inside the database of this module. After that we store the pandas data frame as df and view top five using head . That's all for out first lab and we completed our lab successfully.

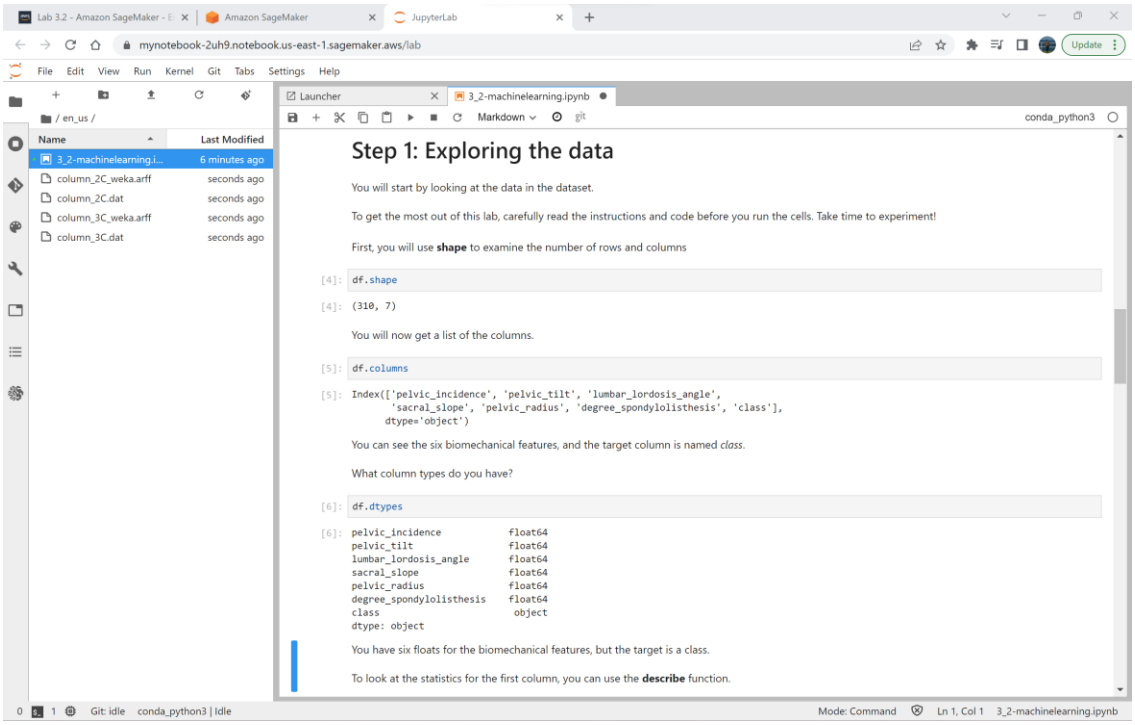


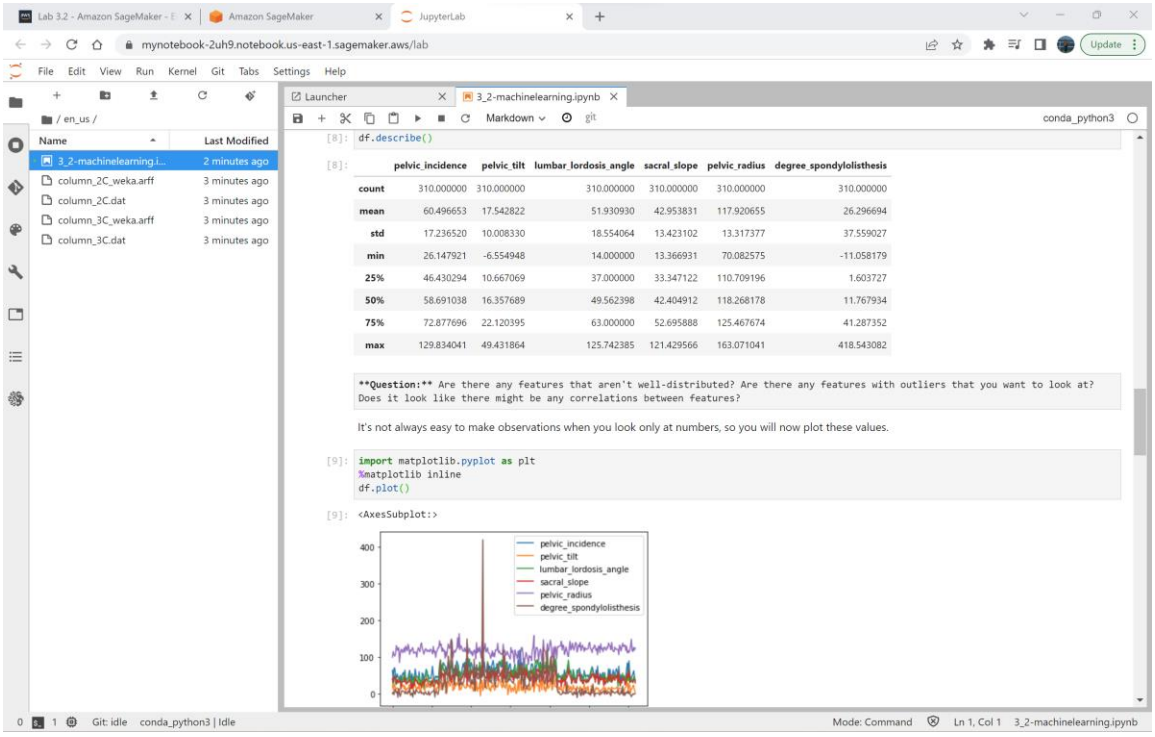
Lab 3.2 Evaluate the data



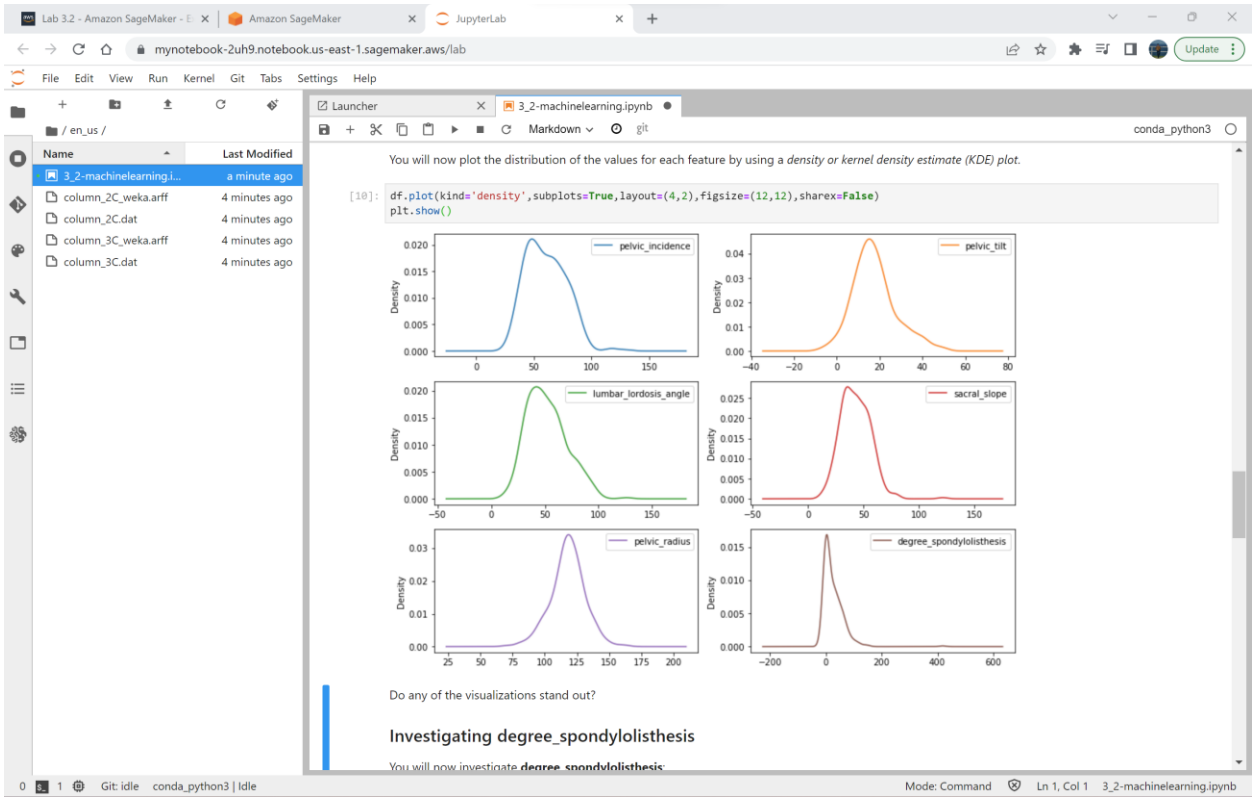


Here, we explore the lab 3.2 code and first thing is to import the data. After importing we are exploring the date by using different syntax such as shape to look into how many column we got and columns to see list of columns on that particular dataset.



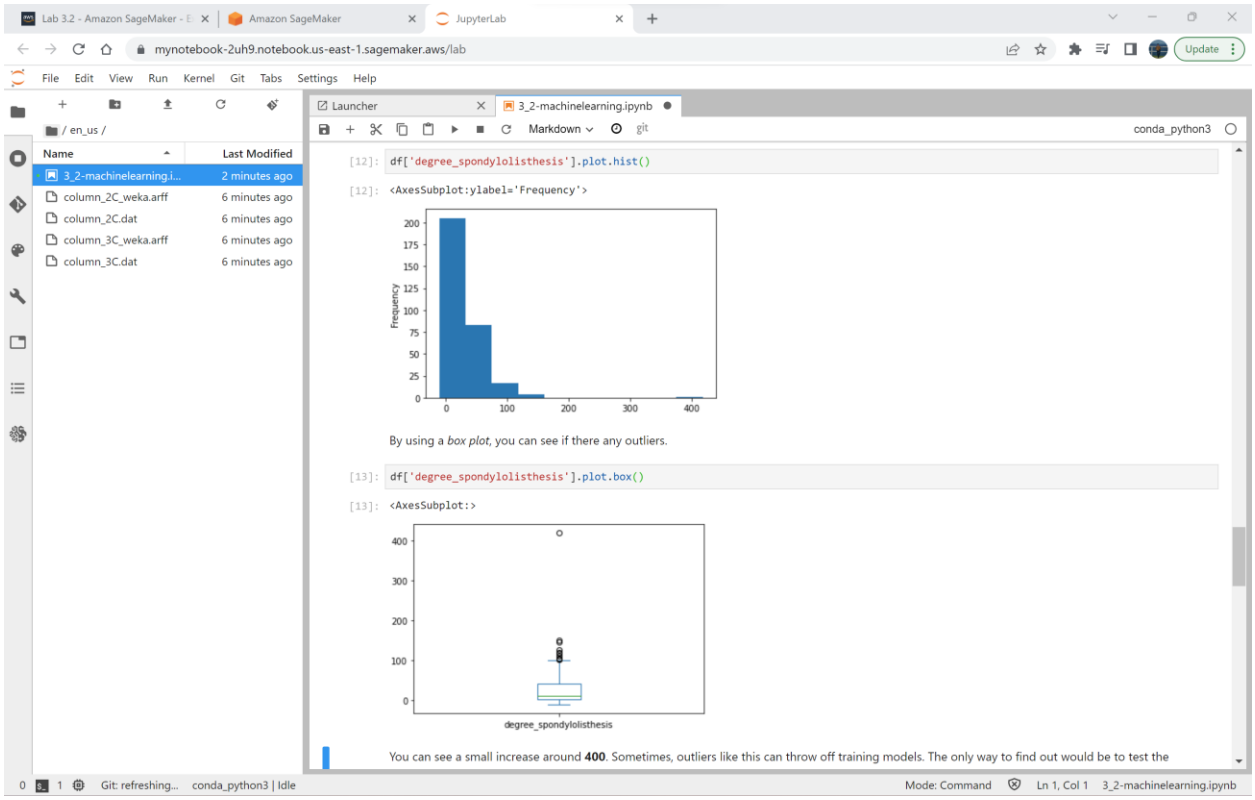


Here we are using describe to look into whether feature is well distributed. We are looking into outlier that we want to look into. We plot all the feature into line graph to see the distribution of data.

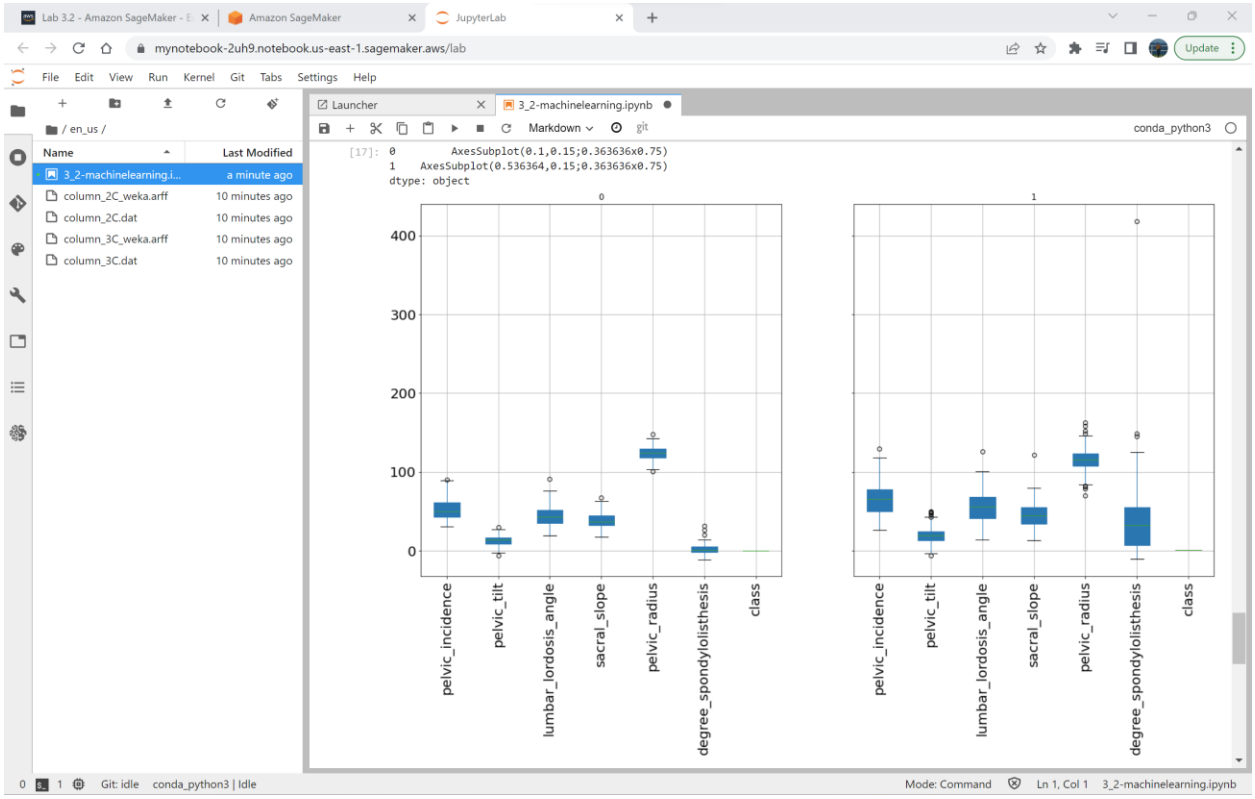


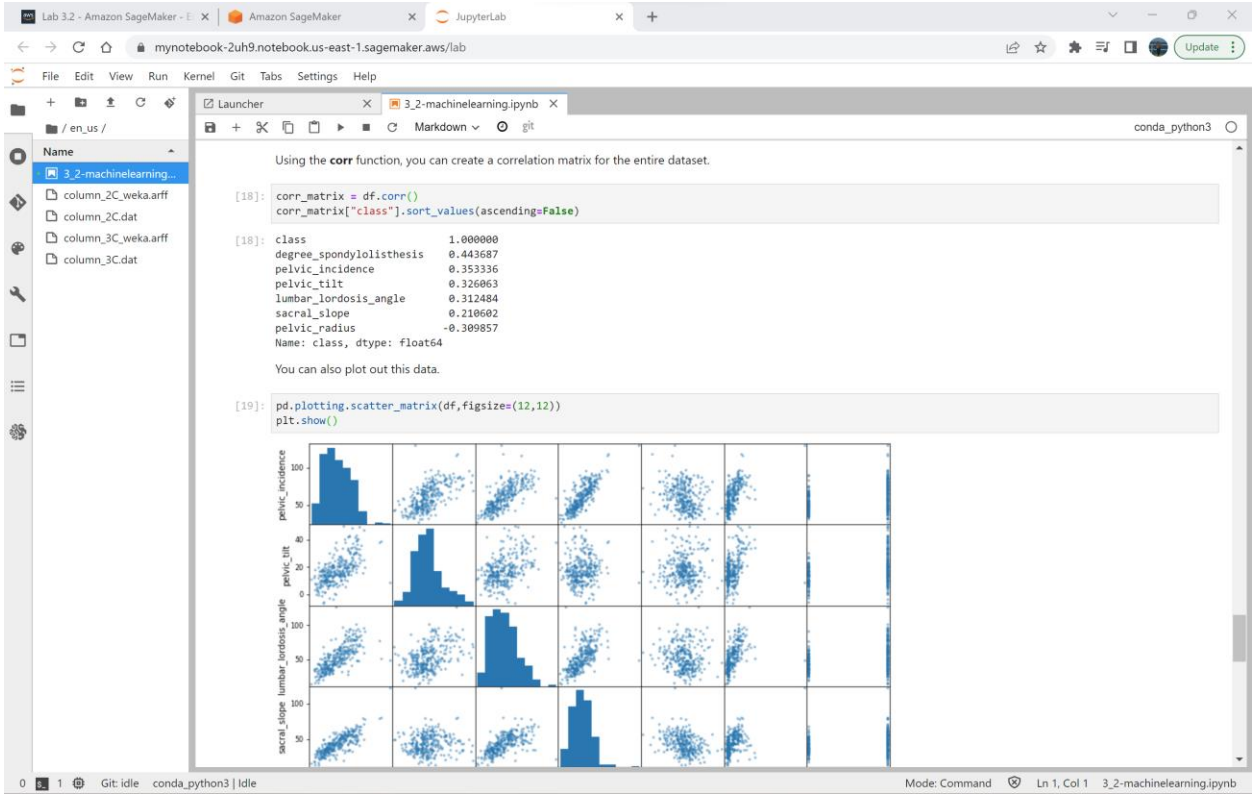
Yes, the visualization are stand out.

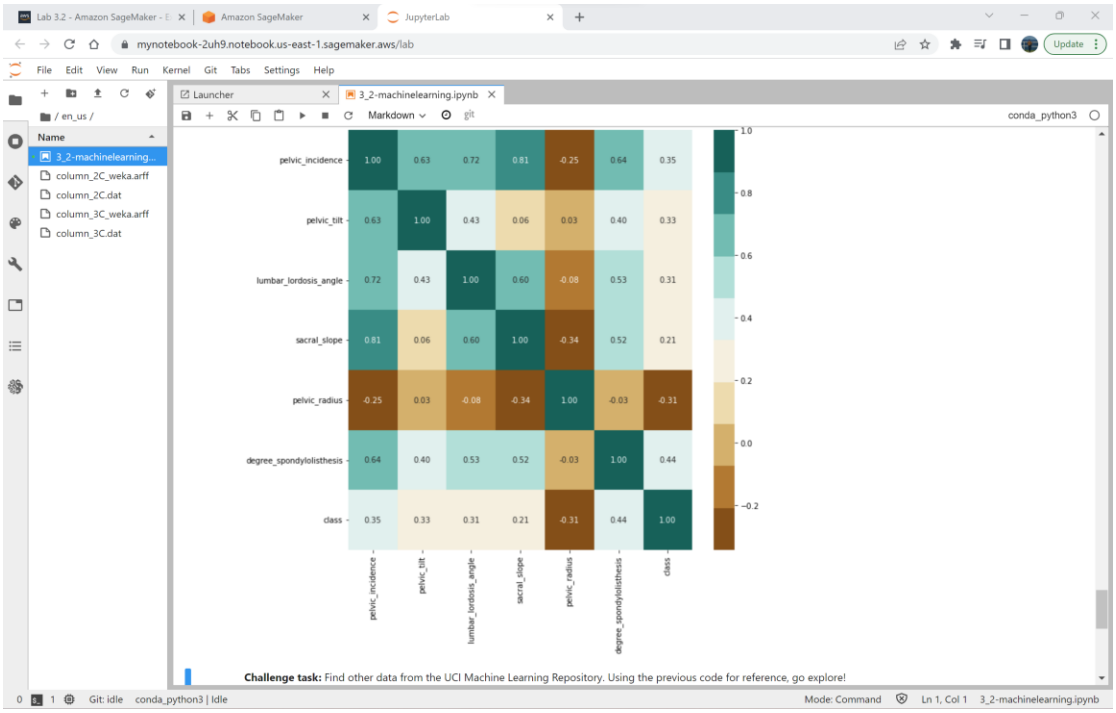
Yes, there is an outlier in degree.



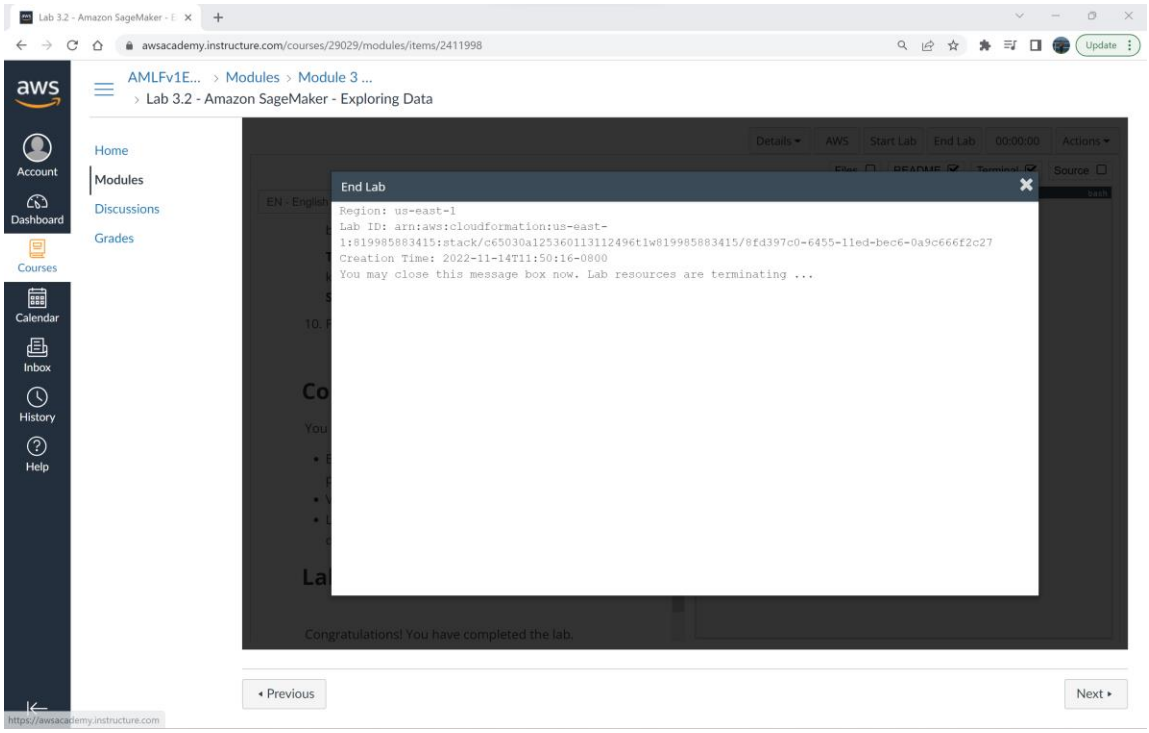
Here are some visualization that we have create to see the outlier and using box plot. A we can see, we can find some outlier as some data are out side the box plot.



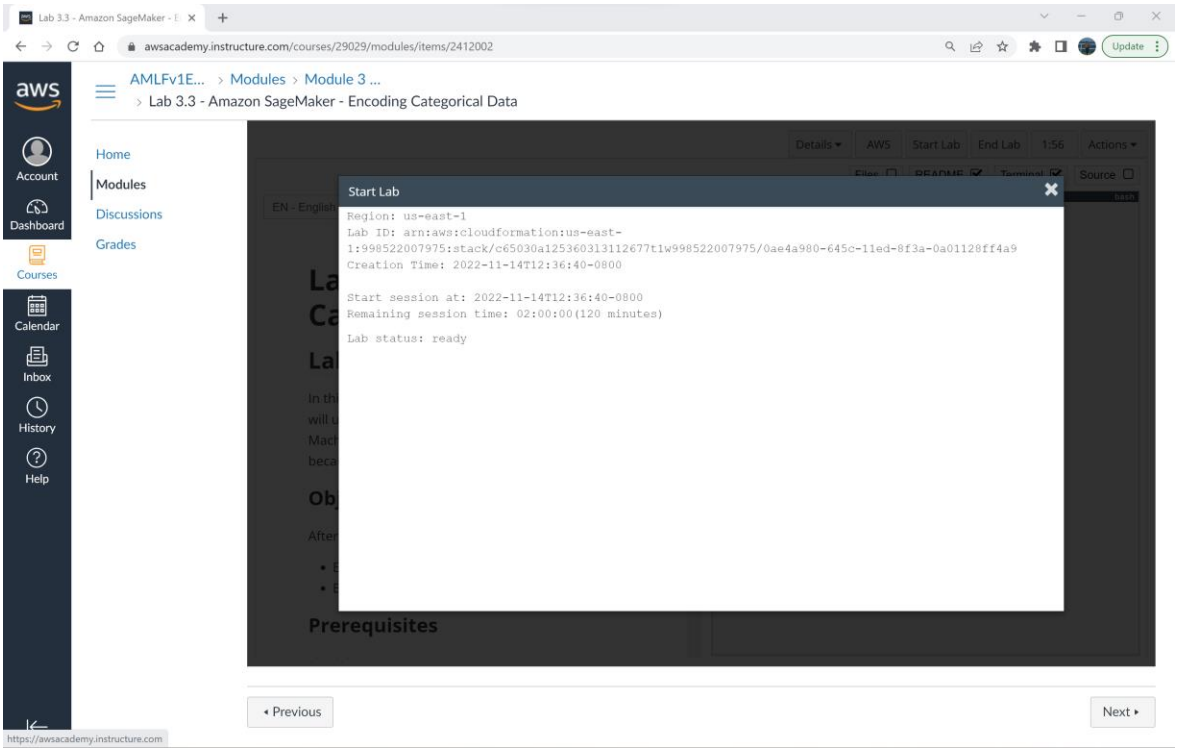


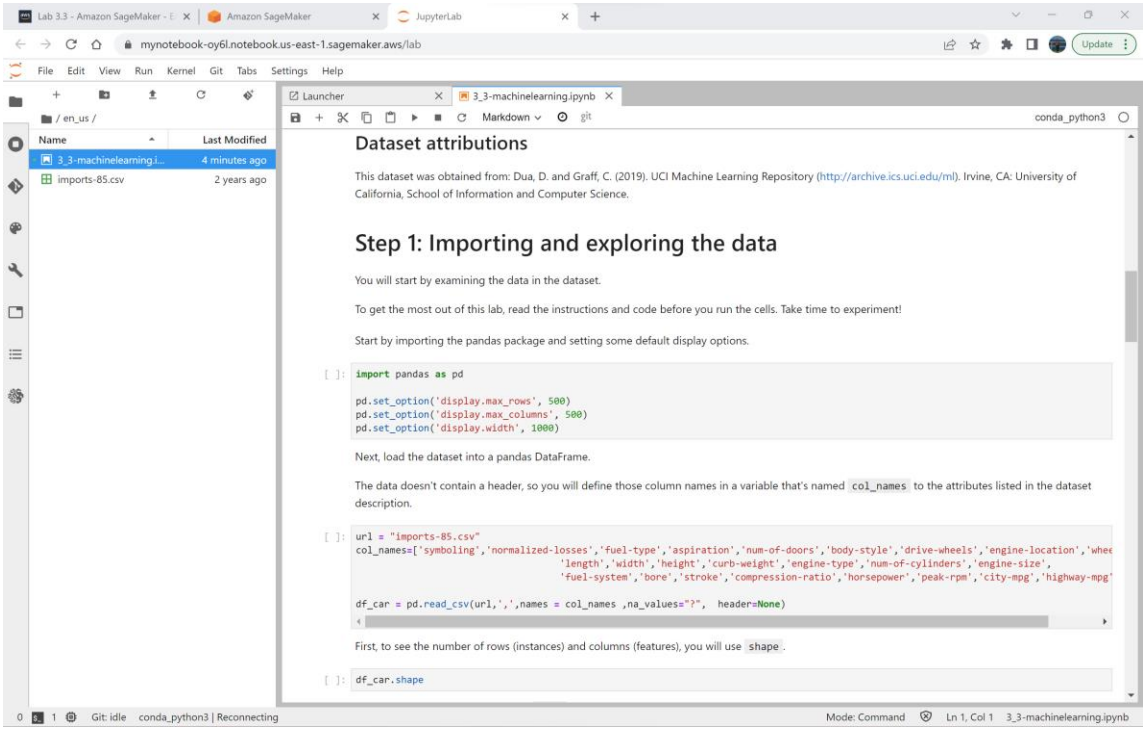


Can check the correlation using seaborn heatmap as well. Some features are highly correlated to each other. Some are not. We have color distribution to distinguish between high and low which represents green to brown. Green being high positive and brown being low negative. That's end out lab and we completed all the step .

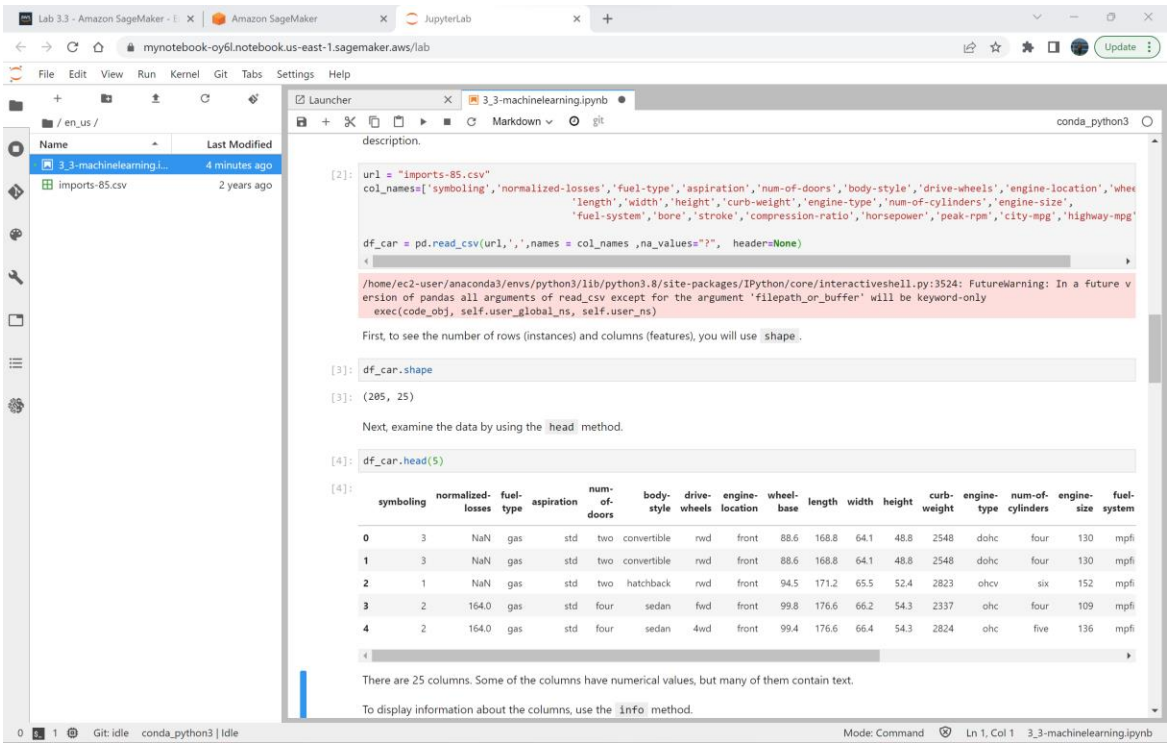


Lab 3.3 Amazon Sage Maker - Encoding Categorical Data





First of all we are importing the data for our lab 3.3. since the data doesn't contain a header, we will define columns names in a variables col_names to the attributes listed in the dataset description.



description.

```
[2]: url = "imports-85.csv"
col_names=['symboling','normalized-losses','fuel-type','aspiration','num-of-doors','body-style','drive-wheels','engine-location','whee
                                        'length','width','height','curb-weight','engine-type','num-of-cylinders','engine-size',
                                        'fuel-system','bore','stroke','compression-ratio','horsepower','peak-rpm','city-mpg','highway-mpg'

df_car = pd.read_csv(url,',',names = col_names ,na_values="?",  header=None)
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3524: FutureWarning: In a future v
ersion of pandas all arguments of read_csv except for the argument 'filepath_or_buffer' will be keyword-only
  exec(code_obj, self.user_global_ns, self.user_ns)
```

First, to see the number of rows (instances) and columns (features), you will use `shape`.

```
[3]: df_car.shape
[3]: (205, 25)
```

Next, examine the data by using the `head` method.

```
[4]: df_car.head(5)
```

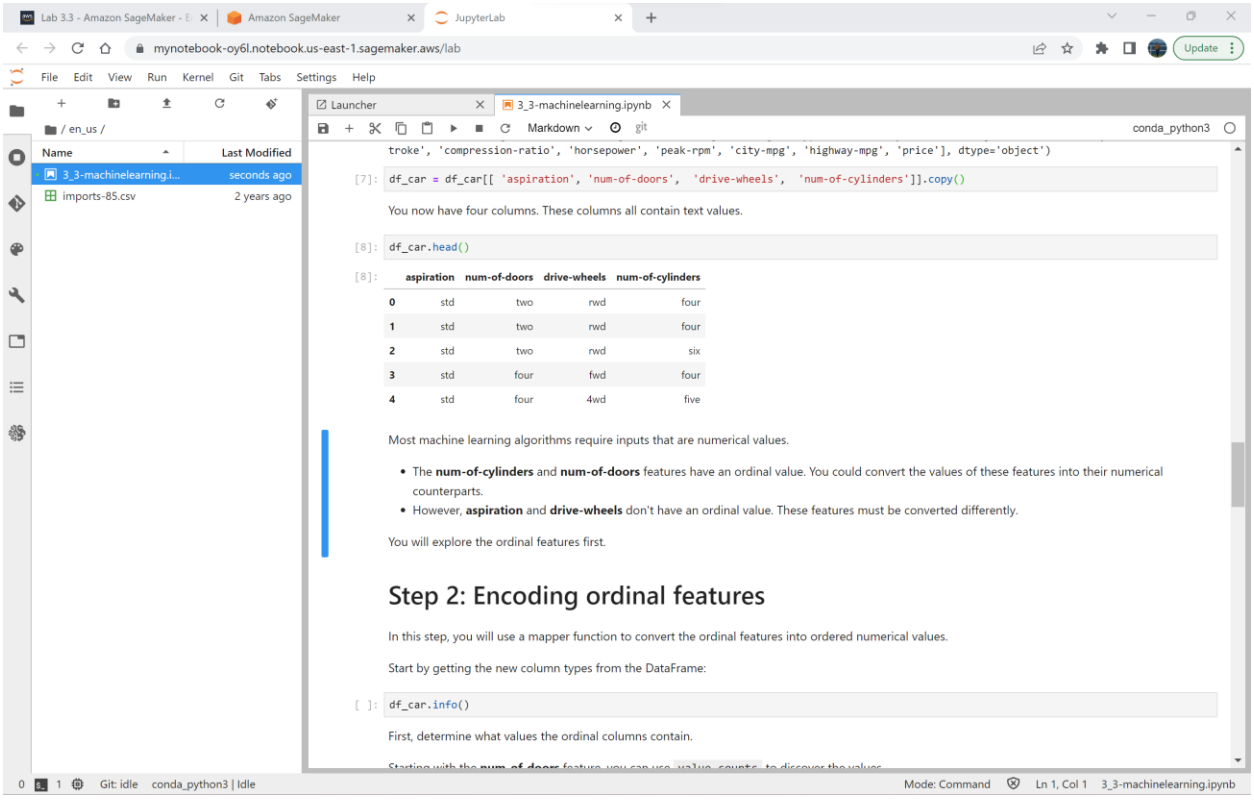
[4]:

	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight	engine-type	num-of-cylinders	engine-size	fuel-system
0	3	NaN	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi
1	3	NaN	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi
2	1	NaN	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi
3	2	164.0	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi
4	2	164.0	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi

There are 25 columns. Some of the columns have numerical values, but many of them contain text.

To display information about the columns, use the `info` method.

Data does not have header so we pass the list of columns and then display the data frame using head.



```
troke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price'], dtype='object')
[7]: df_car = df_car[[ 'aspiration', 'num-of-doors',  'drive-wheels',  'num-of-cylinders']].copy()
```

You now have four columns. These columns all contain text values.

```
[8]: df_car.head()
```

[8]:

	aspiration	num-of-doors	drive-wheels	num-of-cylinders
0	std	two	rwd	four
1	std	two	rwd	four
2	std	two	rwd	six
3	std	four	fwd	four
4	std	four	4wd	five

Most machine learning algorithms require inputs that are numerical values.

- The **num-of-cylinders** and **num-of-doors** features have an ordinal value. You could convert the values of these features into their numerical counterparts.
- However, **aspiration** and **drive-wheels** don't have an ordinal value. These features must be converted differently.

You will explore the ordinal features first.

Step 2: Encoding ordinal features

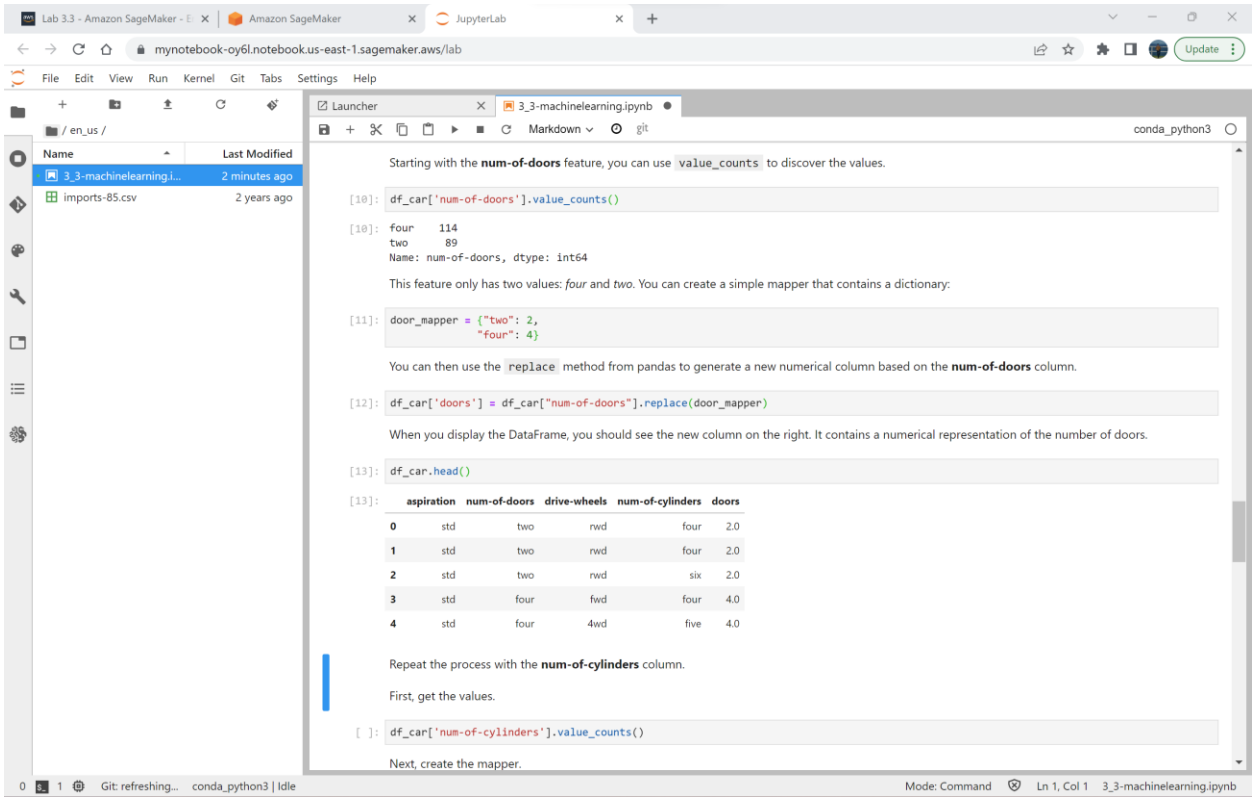
In this step, you will use a mapper function to convert the ordinal features into ordered numerical values.

Start by getting the new column types from the DataFrame:

```
[ ]: df_car.info()
```

First, determine what values the ordinal columns contain.

Her we copy the four column which contain the text data so that we can encode those categorical data into numerical data.



Starting with the **num-of-doors** feature, you can use `value_counts` to discover the values.

```
[10]: df_car['num-of-doors'].value_counts()
```

```
[10]: four    114
      two      89
      Name: num-of-doors, dtype: int64
```

This feature only has two values: *four* and *two*. You can create a simple mapper that contains a dictionary:

```
[11]: door_mapper = {"two": 2,
                     "four": 4}
```

You can then use the `replace` method from pandas to generate a new numerical column based on the **num-of-doors** column.

```
[12]: df_car['doors'] = df_car["num-of-doors"].replace(door_mapper)
```

When you display the DataFrame, you should see the new column on the right. It contains a numerical representation of the number of doors.

```
[13]: df_car.head()
```

[13]:

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors
0	std	two	rwd	four	2.0
1	std	two	rwd	four	2.0
2	std	two	rwd	six	2.0
3	std	four	fwd	four	4.0
4	std	four	4wd	five	4.0

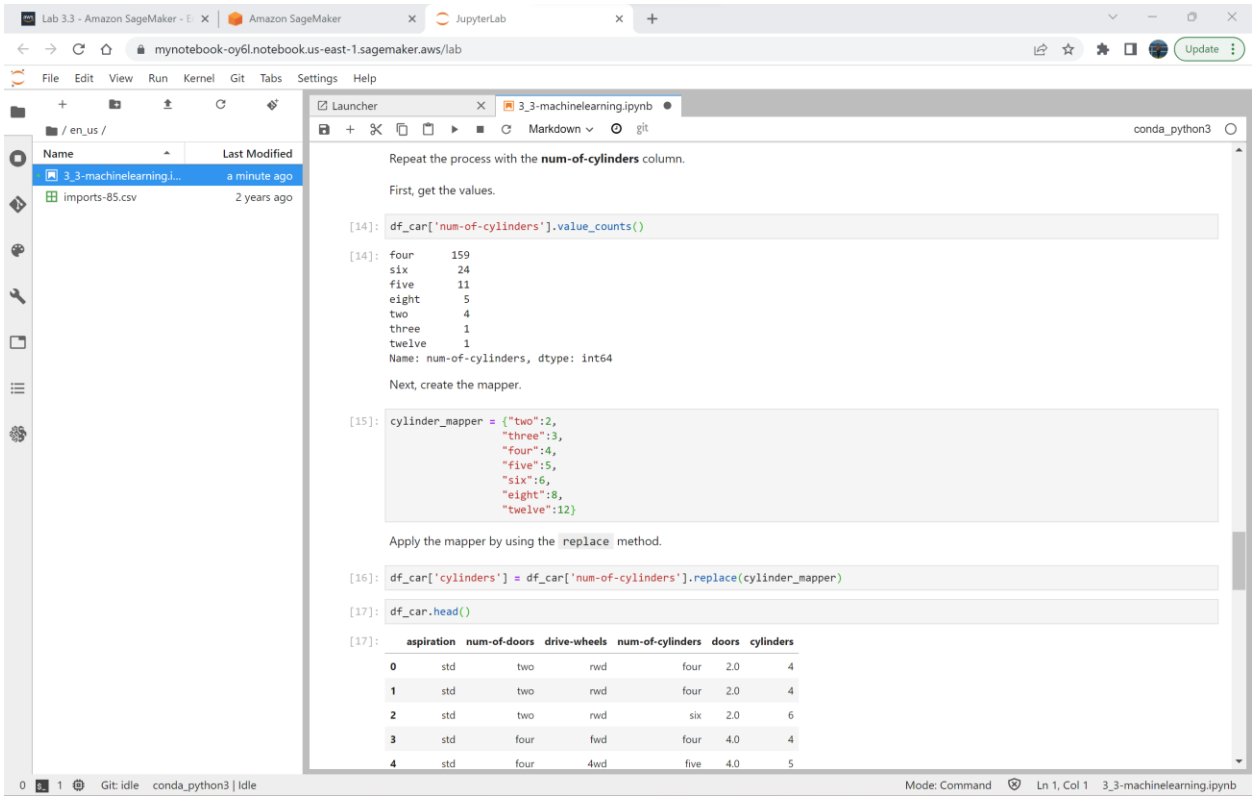
Repeat the process with the **num-of-cylinders** column.

First, get the values.

```
[ ]: df_car['num-of-cylinders'].value_counts()
```

Next, create the mapper.

First we work on num of doors columns, where data are store as two four, we convert that into 2,4 using the mapper function and replace the with given dictionary.



Repeat the process with the **num-of-cylinders** column.

First, get the values.

```
[14]: df_car['num-of-cylinders'].value_counts()
```

```
[14]: four      159
      six        24
      five       11
      eight       5
      two         4
      three       1
      twelve      1
      Name: num-of-cylinders, dtype: int64
```

Next, create the mapper.

```
[15]: cylinder_mapper = {"two":2,
                         "three":3,
                         "four":4,
                         "five":5,
                         "six":6,
                         "eight":8,
                         "twelve":12}
```

Apply the mapper by using the `replace` method.

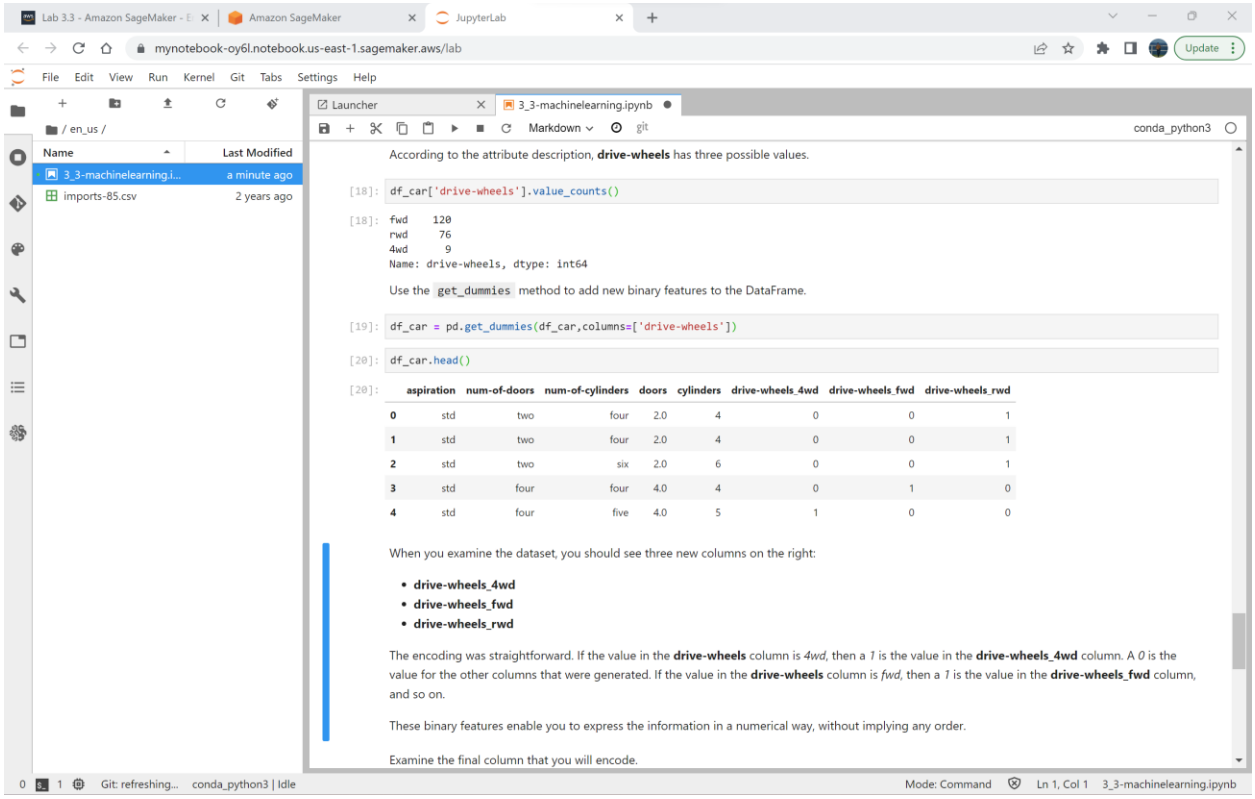
```
[16]: df_car['cylinders'] = df_car['num-of-cylinders'].replace(cylinder_mapper)
```

```
[17]: df_car.head()
```

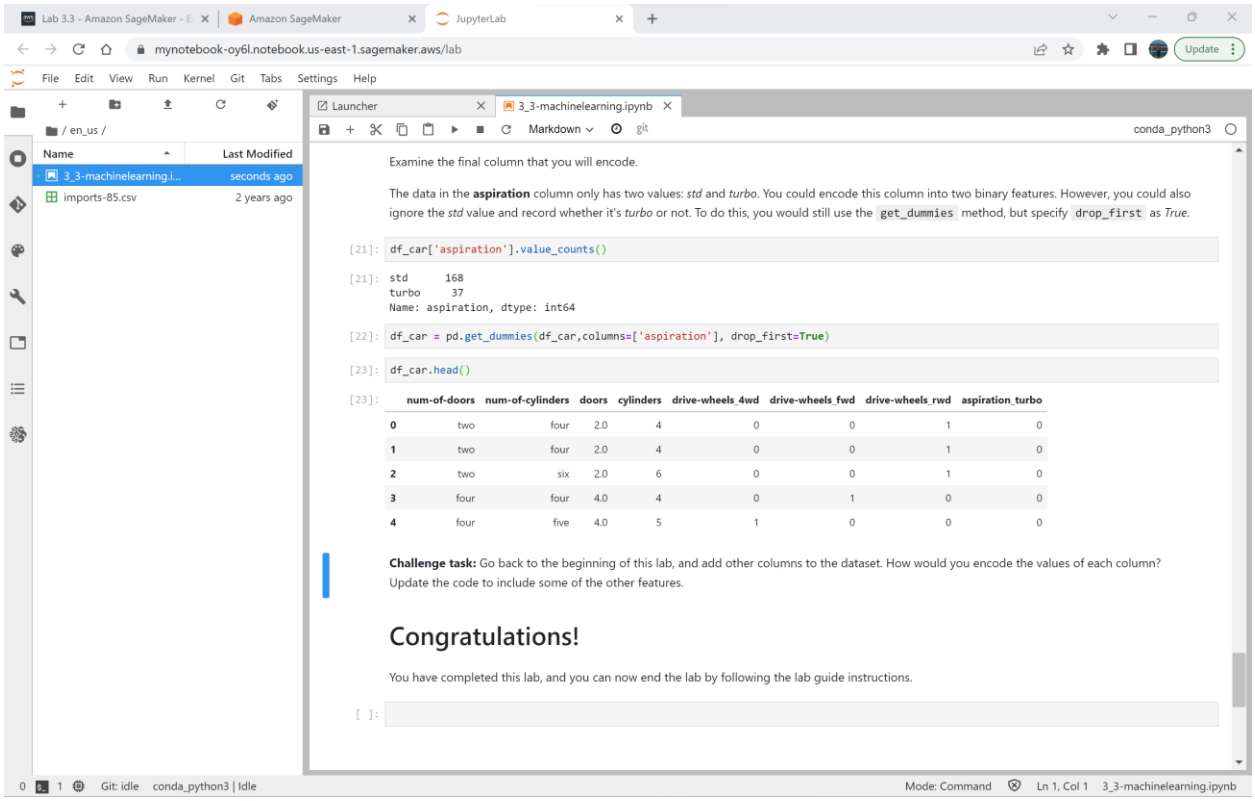
[17]:

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors	cylinders
0	std	two	rwd	four	2.0	4
1	std	two	rwd	four	2.0	4
2	std	two	rwd	six	2.0	6
3	std	four	fwd	four	4.0	4
4	std	four	4wd	five	4.0	5

Here, we repeate the same method as before to replace the categorical data for the num of cylinder where words are convert into numerical data like 2,3,4,5,5,6,8,12.



For drive wheel column , text data are not in numerical word so that we use encoding to covert that into binary features. By using *get.dummies()* method we can get new column according to the data we have in that column.



Examine the final column that you will encode.

The data in the **aspiration** column only has two values: *std* and *turbo*. You could encode this column into two binary features. However, you could also ignore the *std* value and record whether it's *turbo* or not. To do this, you would still use the `get_dummies` method, but specify `drop_first` as *True*.

```
[21]: df_car['aspiration'].value_counts()

[21]: std      168
      turbo     37
      Name: aspiration, dtype: int64

[22]: df_car = pd.get_dummies(df_car,columns=['aspiration'], drop_first=True)

[23]: df_car.head()
```

[23]:

	num-of-doors	num-of-cylinders	doors	cylinders	drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd	aspiration_turbo
0	two	four	2.0	4	0	0	1	0
1	two	four	2.0	4	0	0	1	0
2	two	six	2.0	6	0	0	1	0
3	four	four	4.0	4	0	1	0	0
4	four	five	4.0	5	1	0	0	0

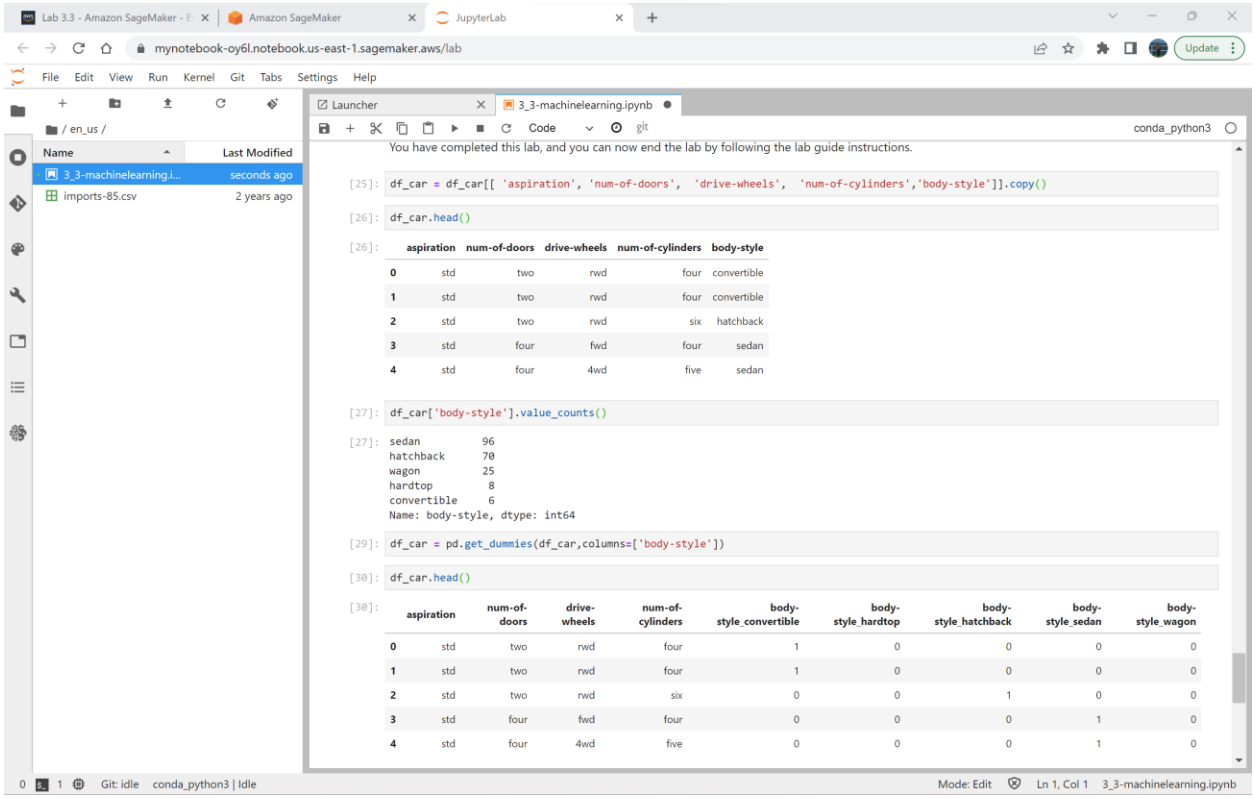
Challenge task: Go back to the beginning of this lab, and add other columns to the dataset. How would you encode the values of each column? Update the code to include some of the other features.

Congratulations!

You have completed this lab, and you can now end the lab by following the lab guide instructions.

```
[ ]:
```

Last column, we drop one feature and just covert one data to check weather that is inside the dataset or not.



You have completed this lab, and you can now end the lab by following the lab guide instructions.

```
[25]: df_car = df_car[[ 'aspiration', 'num-of-doors',  'drive-wheels',  'num-of-cylinders','body-style']].copy()

[26]: df_car.head()
```

[26]:

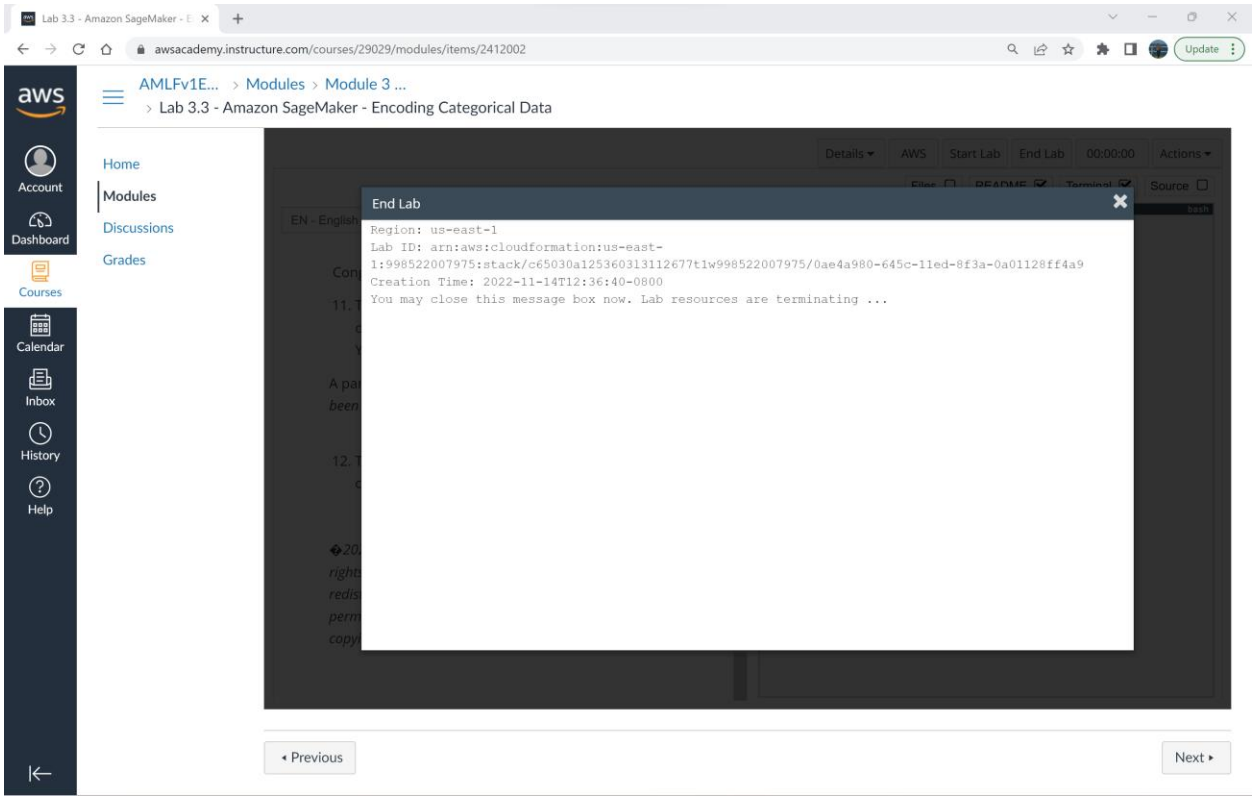
	aspiration	num-of-doors	drive-wheels	num-of-cylinders	body-style
0	std	two	rwd	four	convertible
1	std	two	rwd	four	convertible
2	std	two	rwd	six	hatchback
3	std	four	fwd	four	sedan
4	std	four	4wd	five	sedan

```
[27]: df_car['body-style'].value_counts()

[27]: sedan          96
      hatchback      70
      wagon          25
      hardtop         8
      convertible     6
      Name: body-style, dtype: int64

[29]: df_car = pd.get_dummies(df_car,columns=['body-style'])

[30]: df_car.head()
```

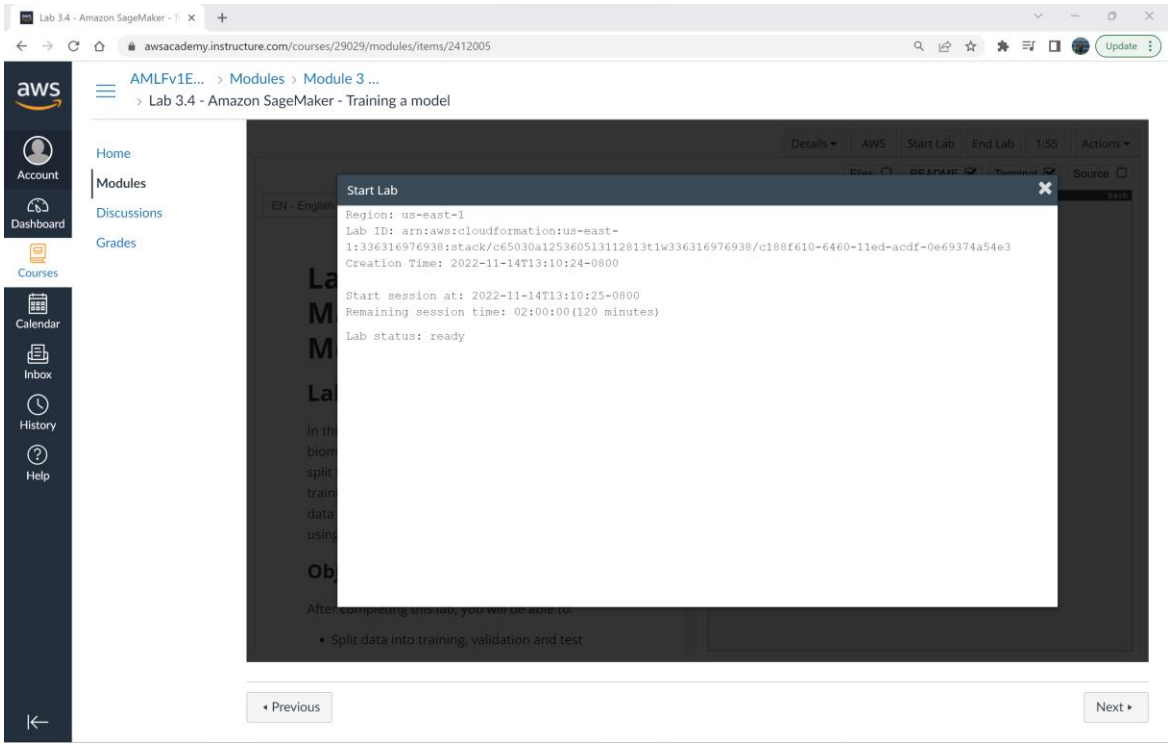
[30]:

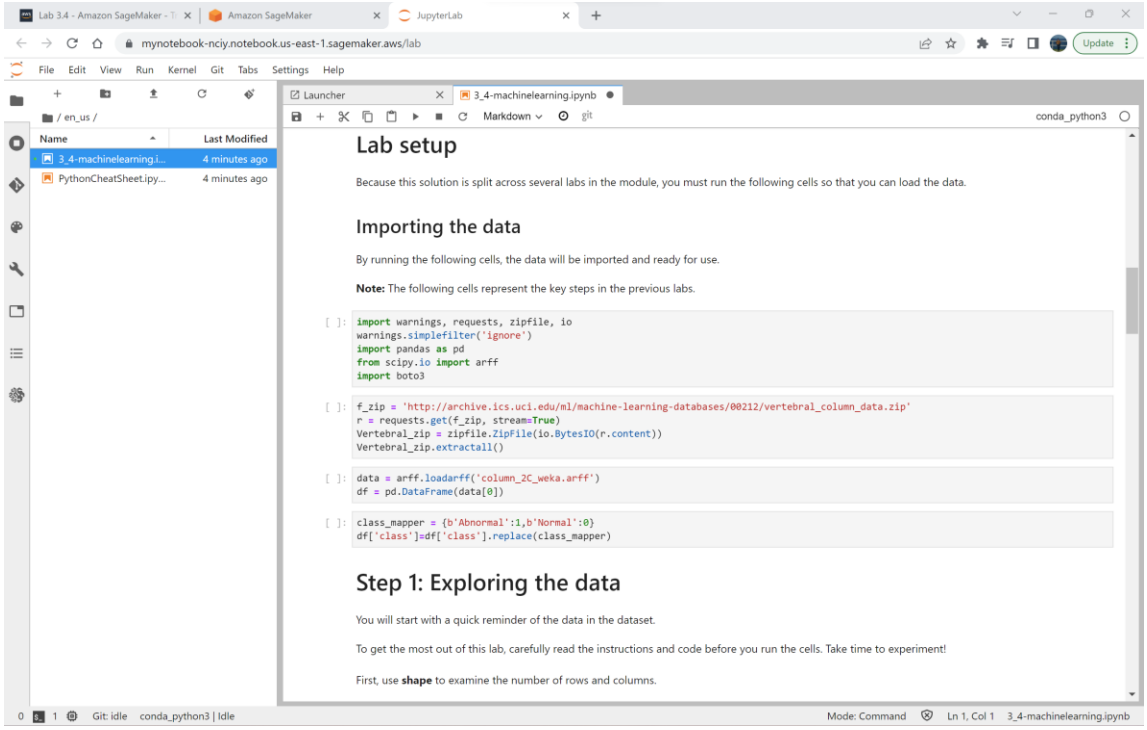
	aspiration	num-of-doors	drive-wheels	num-of-cylinders	body-style_convertible	body-style_hardtop	body-style_hatchback	body-style_sedan	body-style_wagon
0	std	two	rwd	four	1	0	0	0	0
1	std	two	rwd	four	1	0	0	0	0
2	std	two	rwd	six	0	0	1	0	0
3	std	four	fwd	four	0	0	0	1	0
4	std	four	4wd	five	0	0	0	1	0

For challenge task, I have added body style column into new copied dataset to covert into numerical data. We found out that it has 5 different types. We use *same get_dummies()* method to covert into binary features and add new column for each features. That's all for this lab and we completed are the required step for this lab.

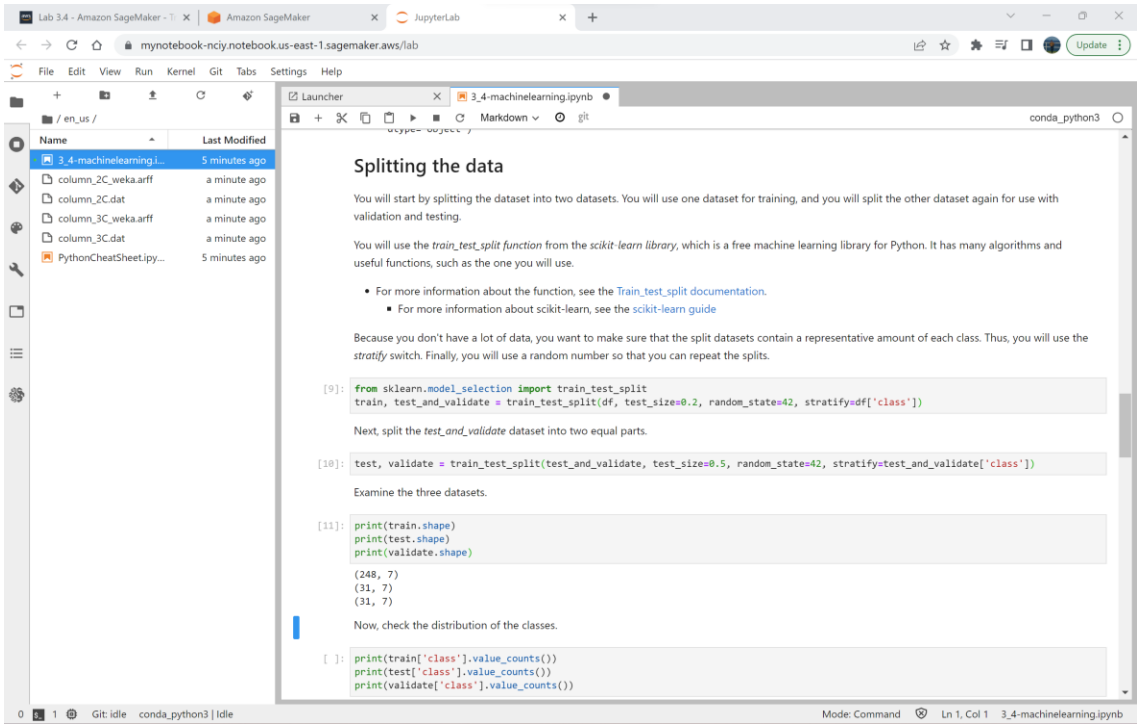


Lab 3.4 Amazon Sage Maker - Training a Machine Learning Model

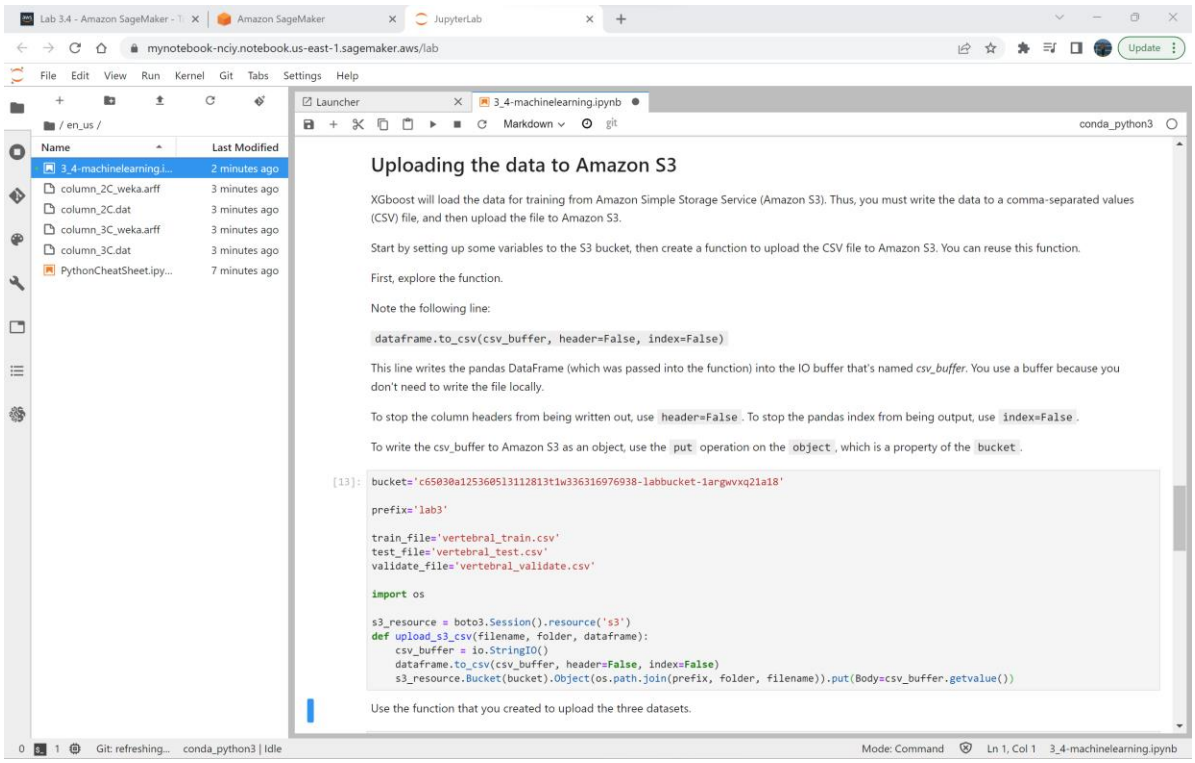




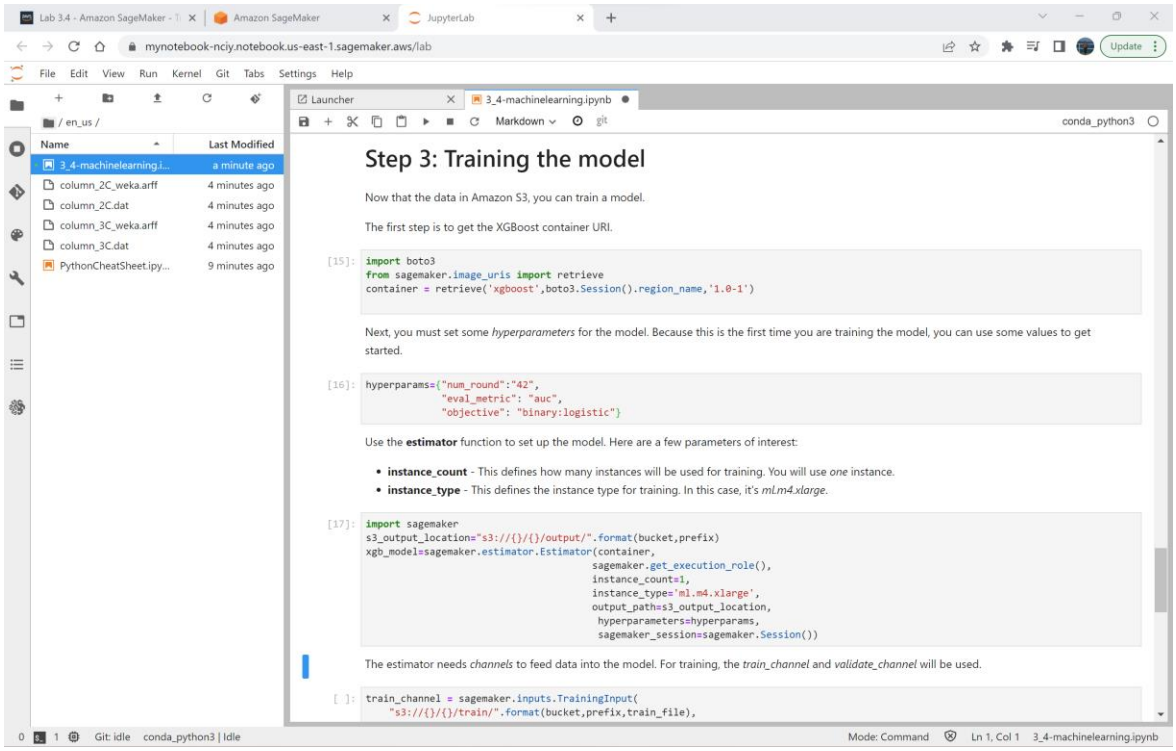
Loading the file form en_us -> 3_4 lab. And importing all the libraries and dataset. Here we change the b'abnormal and b'normal to 1,0 using mapper so that we can use that as features for further analysis.



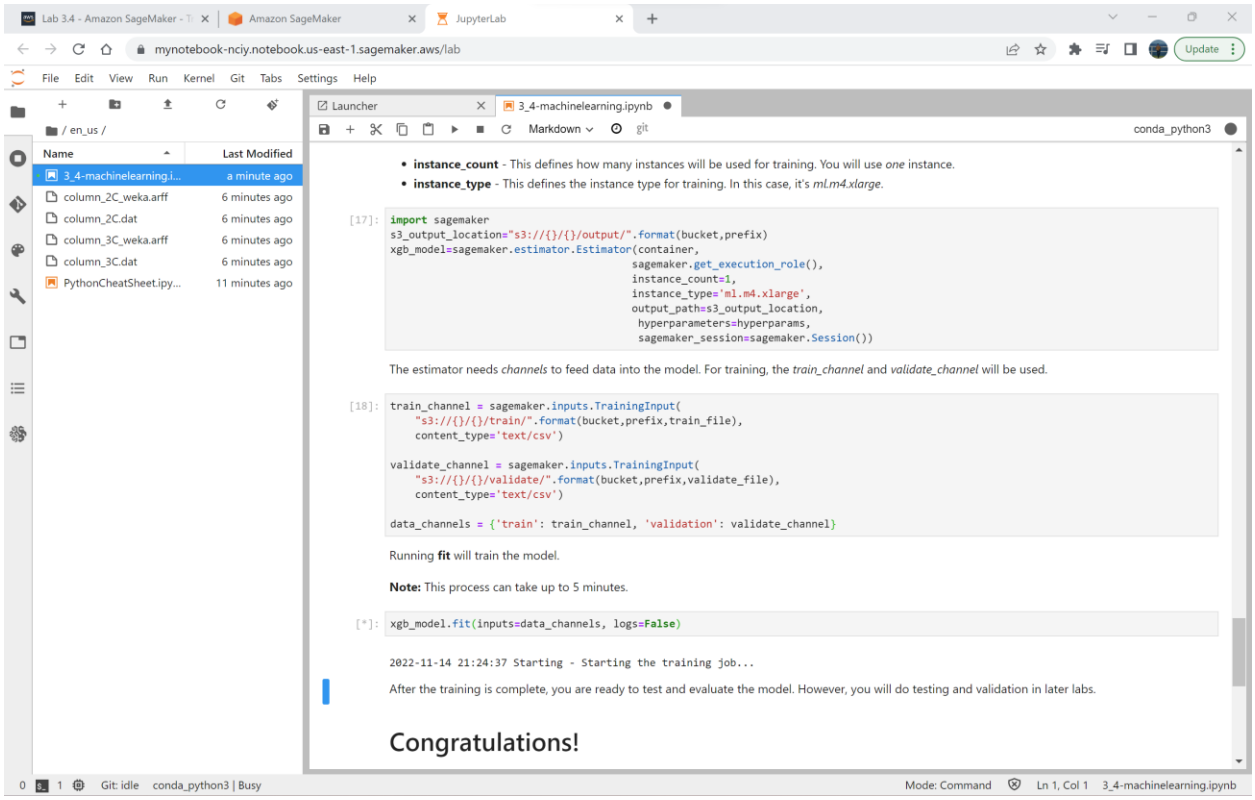
Here, we are splitting the data into test and validate, train. Using the sklean.model_selection and importing train_test_split. We use 80 |20 train test for this model.



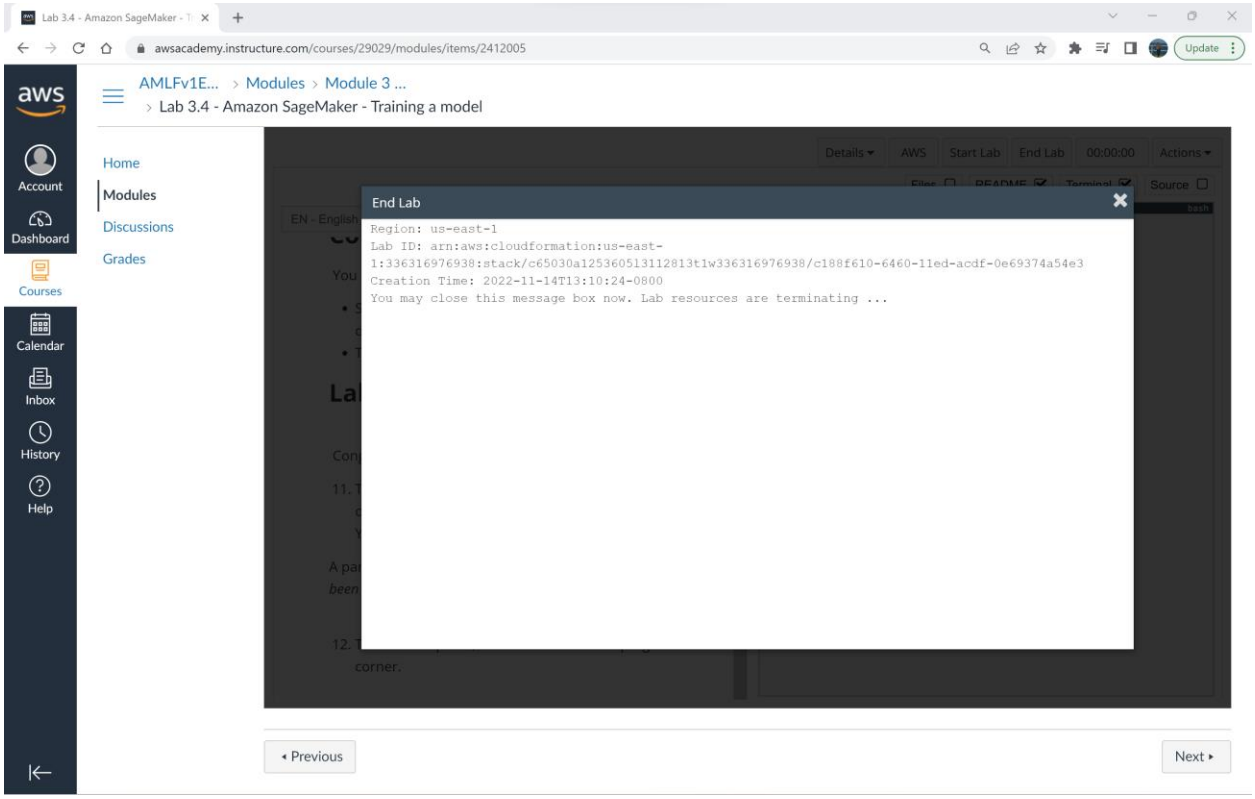
Now, we are uploading the data into Amazon S3 by creating the csv files of recent data frame of each train, test, and validate files.



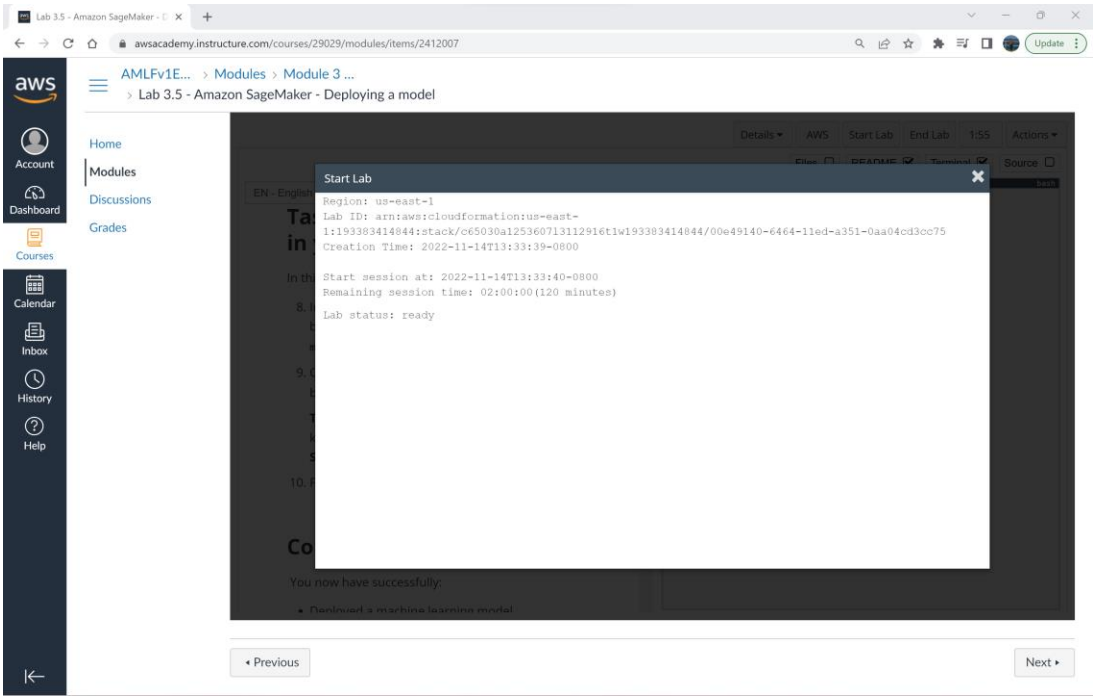
Now, we train the model where our data are in Amazon S3.

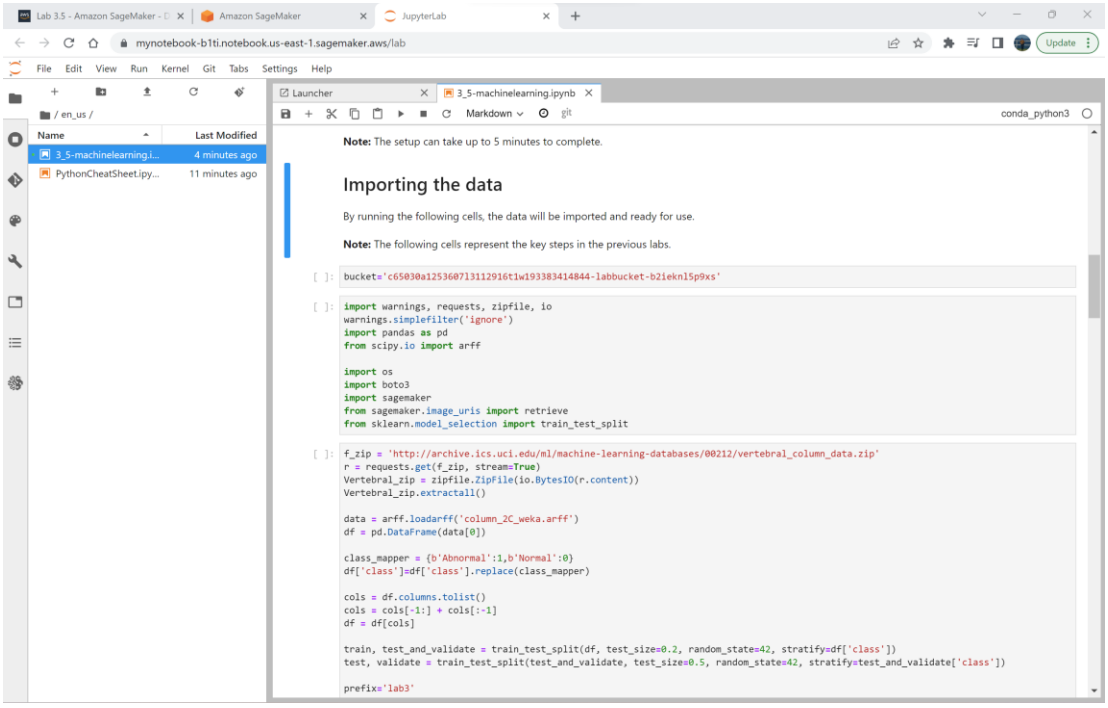


At the end we finally mange to train out job into amazon sage maker. That's all for this lab and we have completed all the step required for this lab.

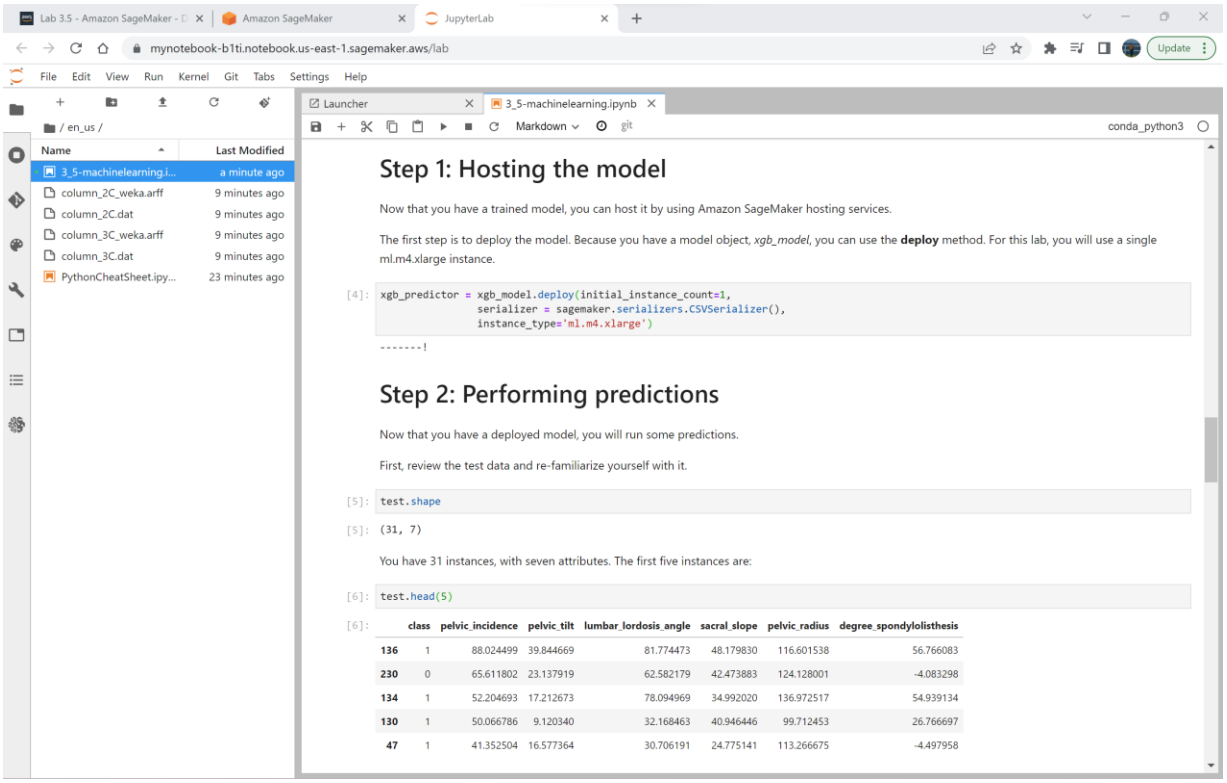


Lab 3.5 Machine Learning Pipeline with Amazon Sage Maker





Here, we load the data from sage maker. Run the code file inside the en_us/ 3_5 machine learning. After importing we do train split for the data with 80- 20 split.



Step 1: Hosting the model

Now that you have a trained model, you can host it by using Amazon SageMaker hosting services.

The first step is to deploy the model. Because you have a model object, *xgb_model*, you can use the **deploy** method. For this lab, you will use a single ml.m4.xlarge instance.

```
[4]: xgb_predictor = xgb_model.deploy(initial_instance_count=1,
                                      serializer = sagemaker.serializers.CSVSerializer(),
                                      instance_type='ml.m4.xlarge')
-------!
```

Step 2: Performing predictions

Now that you have a deployed model, you will run some predictions.

First, review the test data and re-familiarize yourself with it.

```
[5]: test.shape
[5]: (31, 7)
```

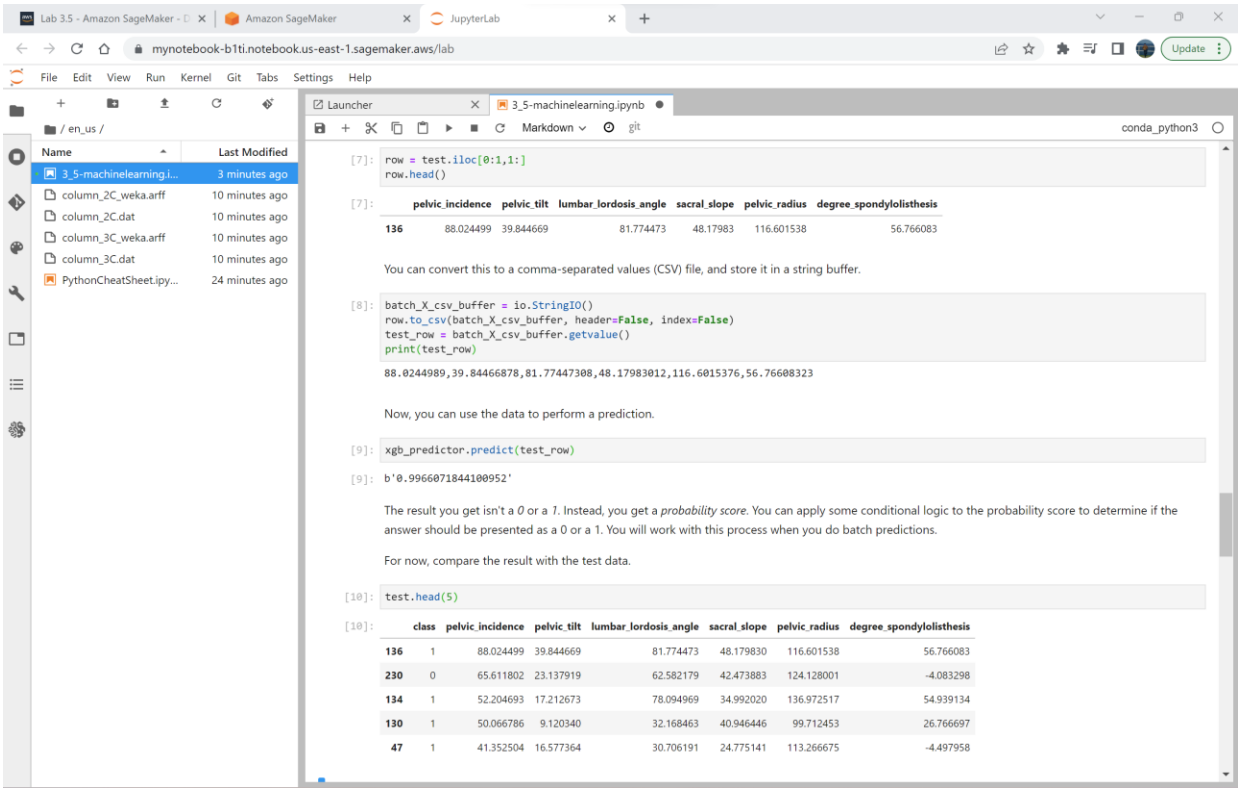
You have 31 instances, with seven attributes. The first five instances are:

```
[6]: test.head(5)
```

[6]:

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

Here, we do step 1 for hosting the model and then step 2 for performing the predictions.



```
[7]: row = test.iloc[0:1,1:]
     row.head()
```

[7]:

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	88.024499	39.844669	81.774473	48.17983	116.601538	56.766083

You can convert this to a comma-separated values (CSV) file, and store it in a string buffer.

```
[8]: batch_X_csv_buffer = io.StringIO()
     row.to_csv(batch_X_csv_buffer, header=False, index=False)
     test_row = batch_X_csv_buffer.getvalue()
     print(test_row)
88.0244989,39.84466878,81.77447308,48.17983012,116.6015376,56.76608323
```

Now, you can use the data to perform a prediction.

```
[9]: xgb_predictor.predict(test_row)
[9]: b'0.9966071844100952'
```

The result you get isn't a *0* or a *1*. Instead, you get a *probability score*. You can apply some conditional logic to the probability score to determine if the answer should be presented as a 0 or a 1. You will work with this process when you do batch predictions.

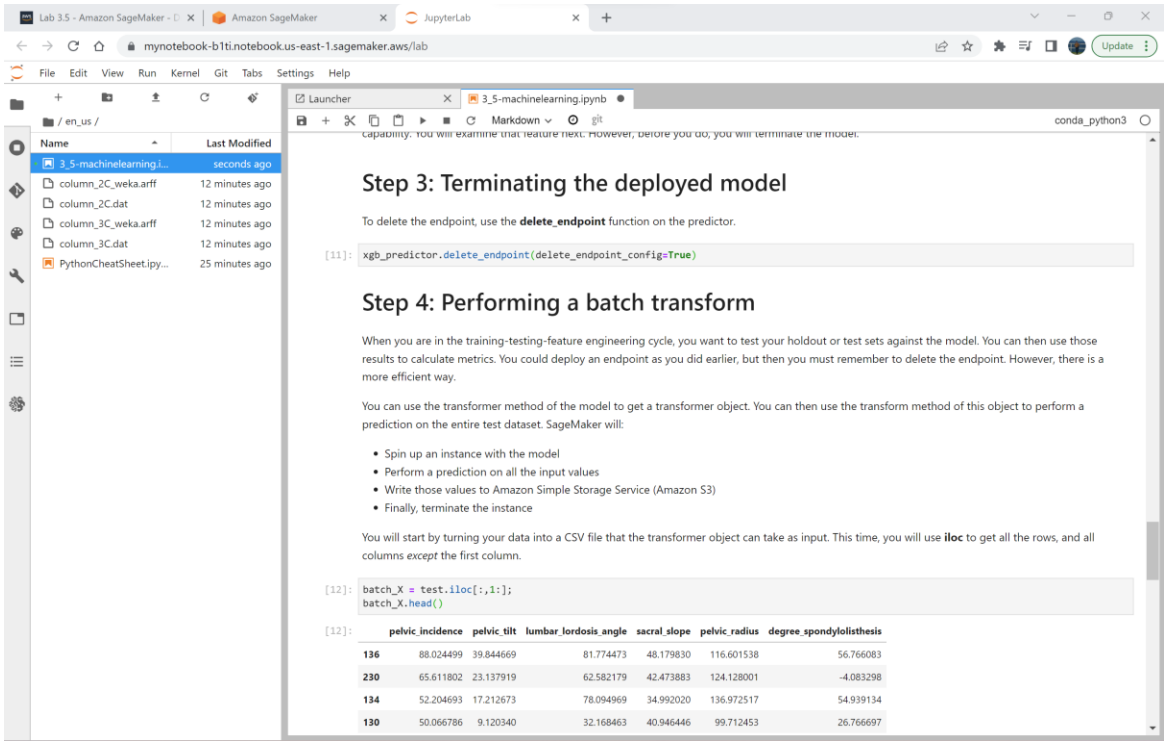
For now, compare the result with the test data.

```
[10]: test.head(5)
```

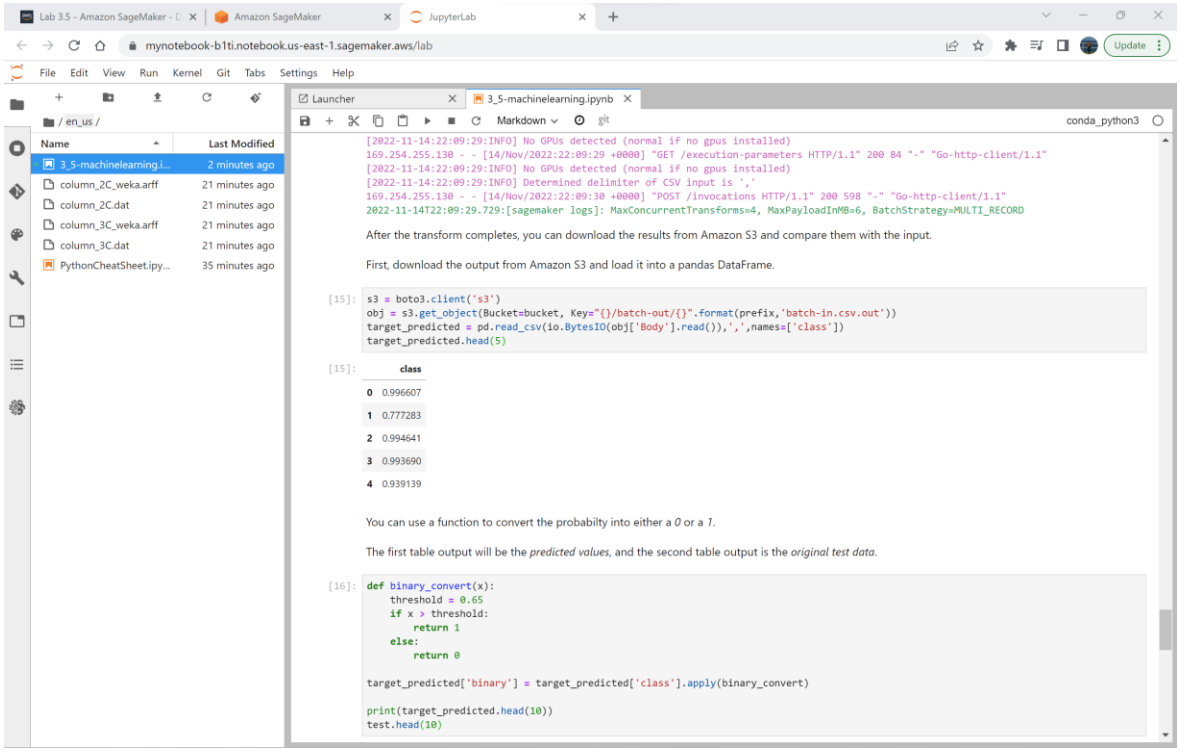
[10]:

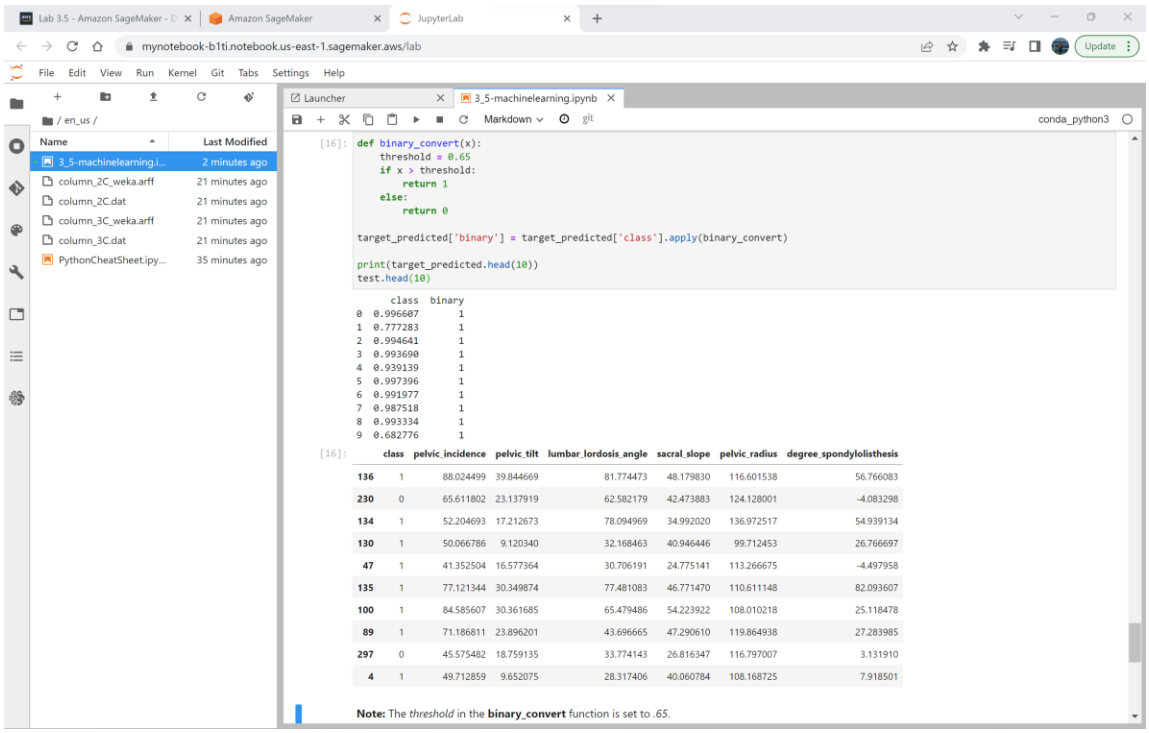
	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

Here we can analyse the result, the its has **99% accuracy** and it's the best fit model.



Now, we move to step 3 which is terminating the deployed model. Then we form batch transform for step 4. In this step we start by turning out data into csv file that the transformer object can take as input. This time, we use iloc to geta all the rows, and all column except the first columns





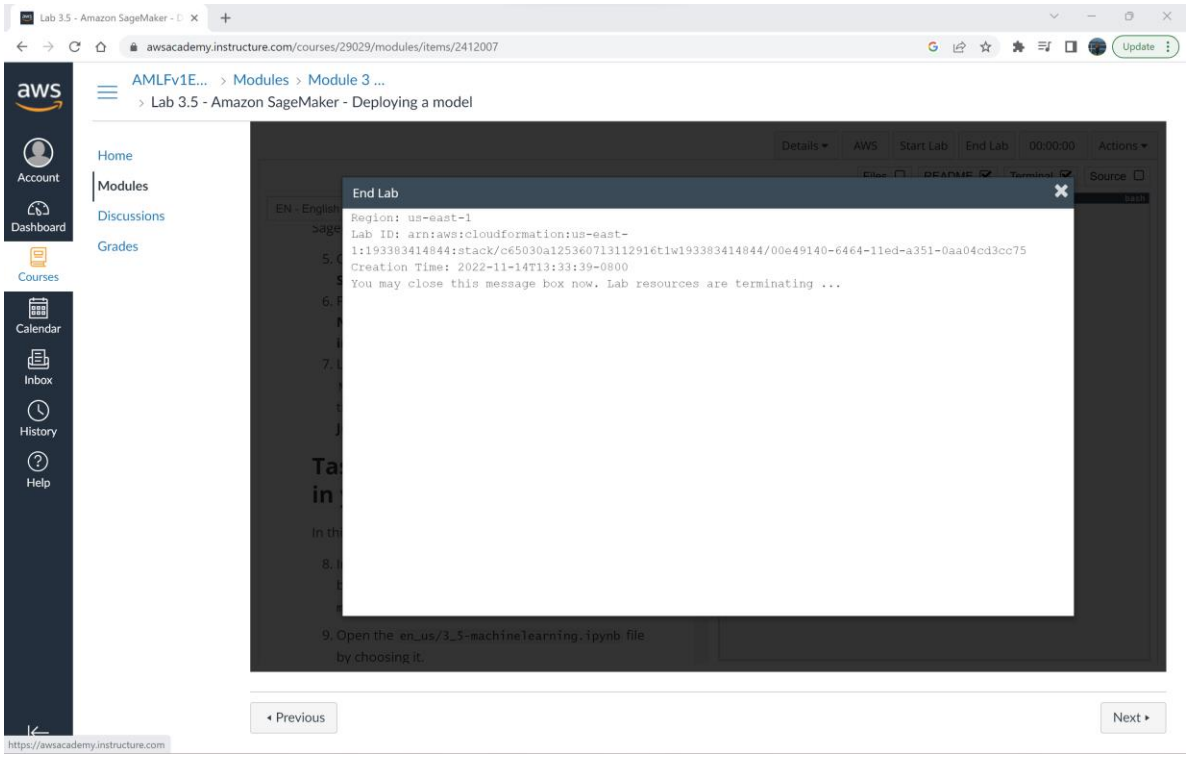
```
def binary_convert(x):
    threshold = 0.65
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['binary'] = target_predicted['class'].apply(binary_convert)

print(target_predicted.head(10))
test.head(10)
```

```
      class  binary
0  0.996607       1
1  0.777283       1
2  0.994641       1
3  0.993690       1
4  0.939139       1
5  0.997396       1
6  0.991977       1
7  0.987518       1
8  0.993334       1
9  0.682776       1
```

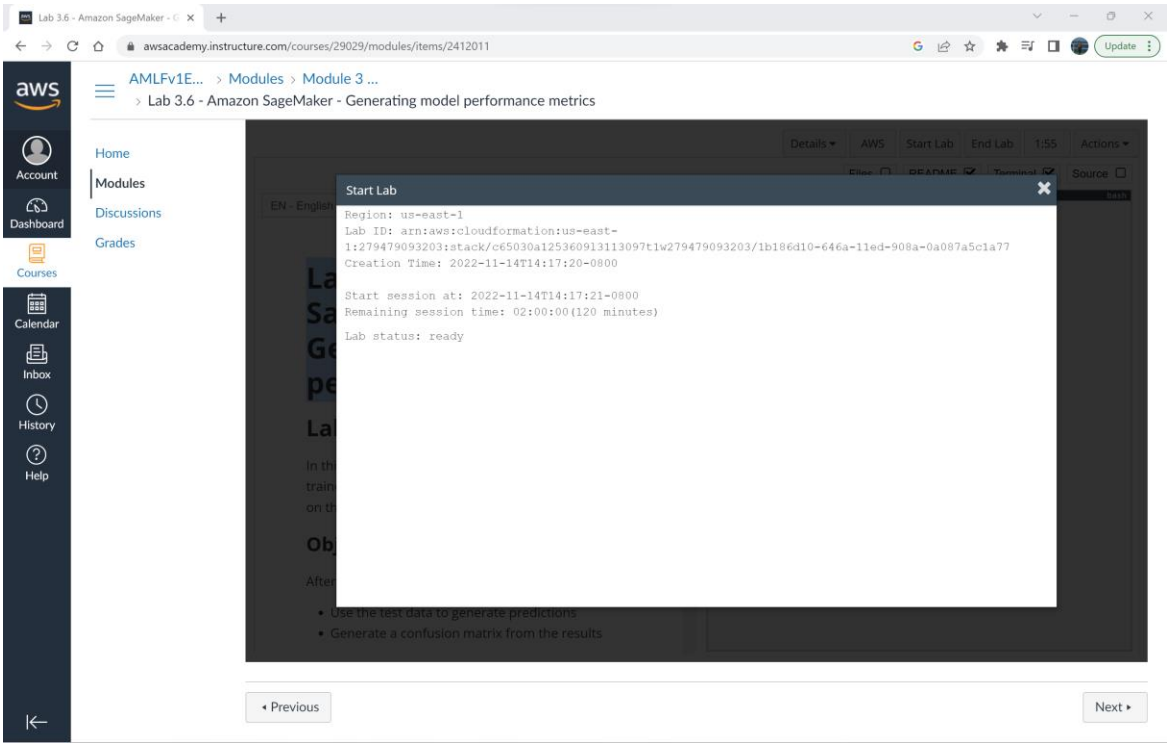
	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958
135	1	77.121344	30.349874	77.481083	46.771470	110.611148	82.093607
100	1	84.585607	30.361685	65.479486	54.223922	108.010218	25.118478
89	1	71.186811	23.896201	43.696665	47.290610	119.864938	27.283985
297	0	45.575482	18.759135	33.774143	26.816347	116.797007	3.131910
4	1	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501

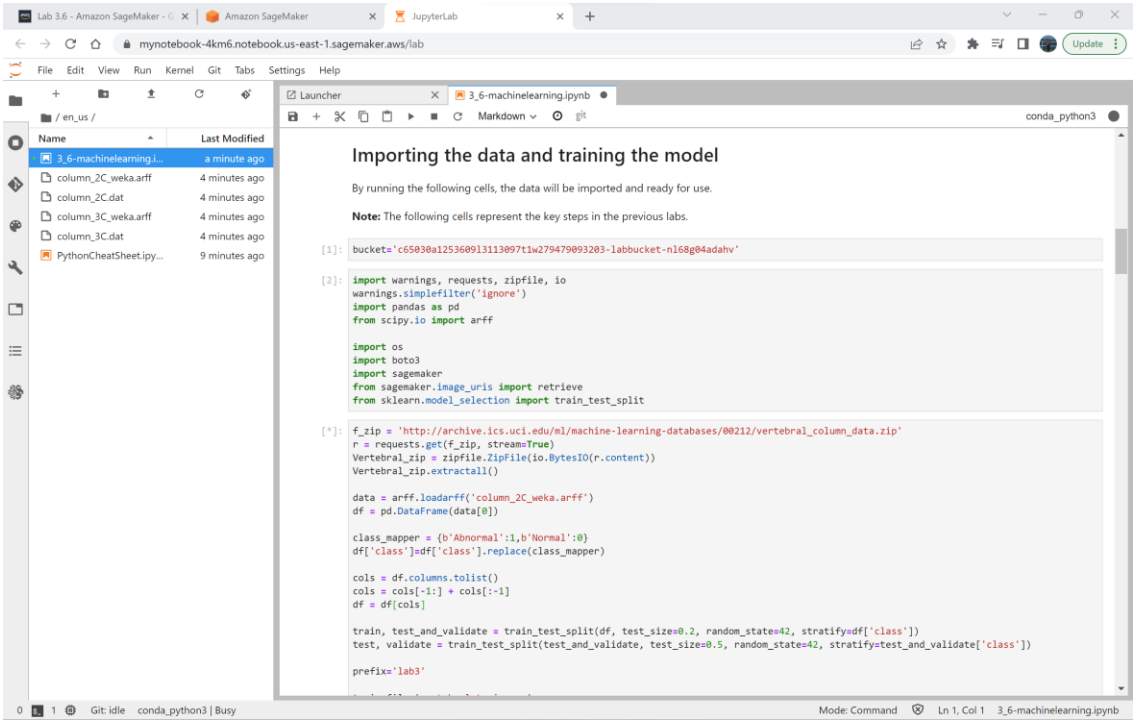
Note: The *threshold* in the **binary_convert** function is set to .65.

At the end we managed to perform all the lab work for his lab. And completed successfully.

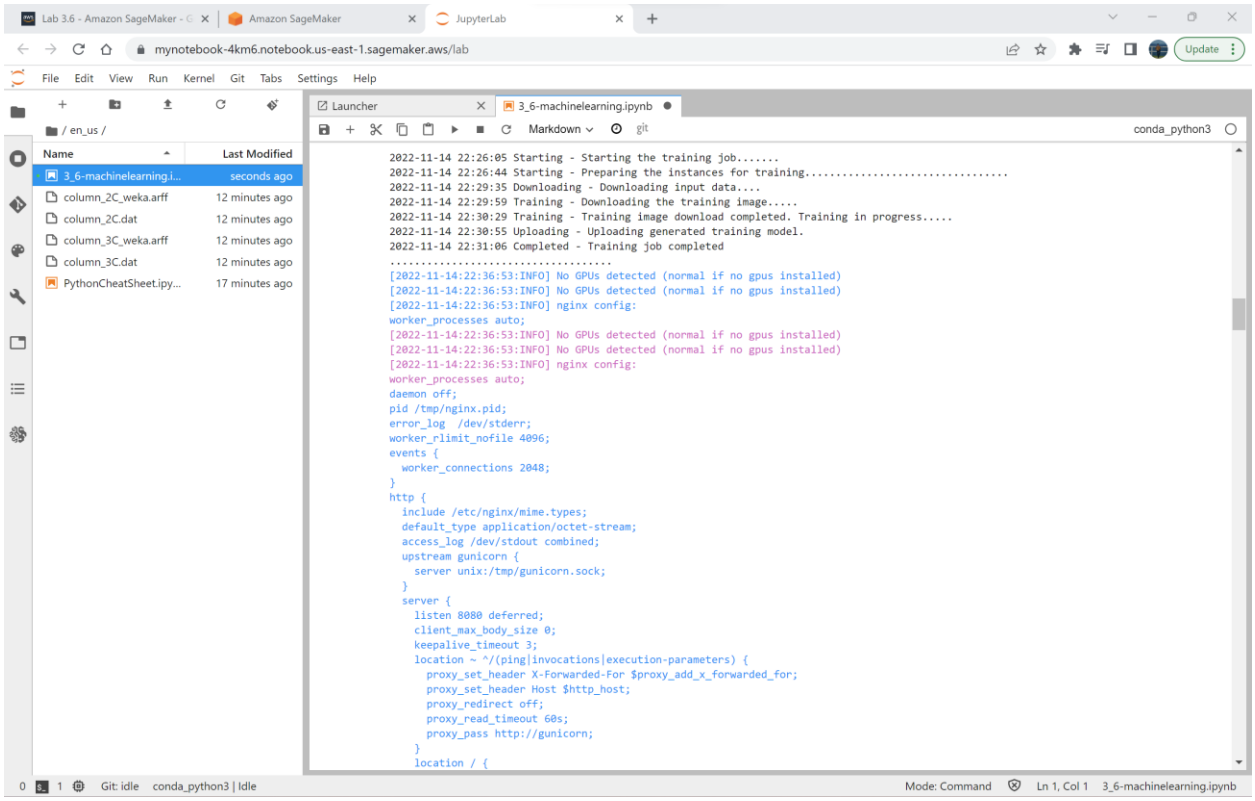


Lab 3.6 - Amazon Sage Maker - Generating model performance metrics

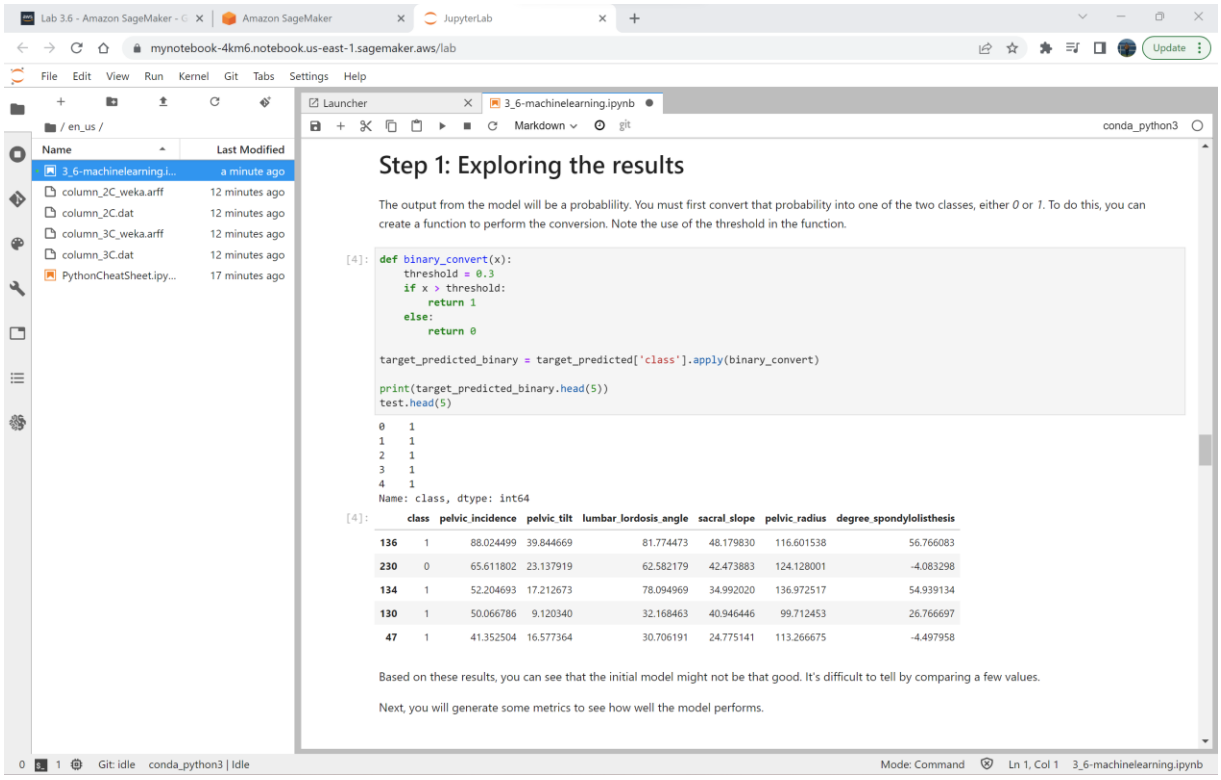




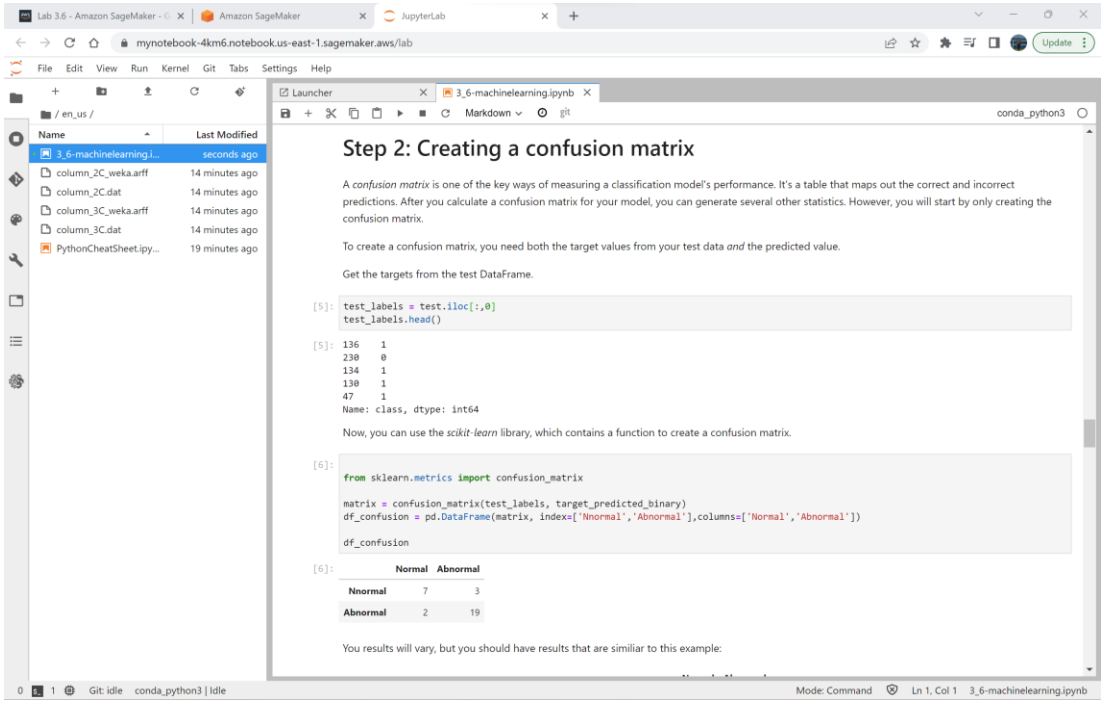
Here, we load the data from sage maker. Run the code file inside the en_us/ 3_6 machine learning. After importing we do train split for the data with 80- 20 split.



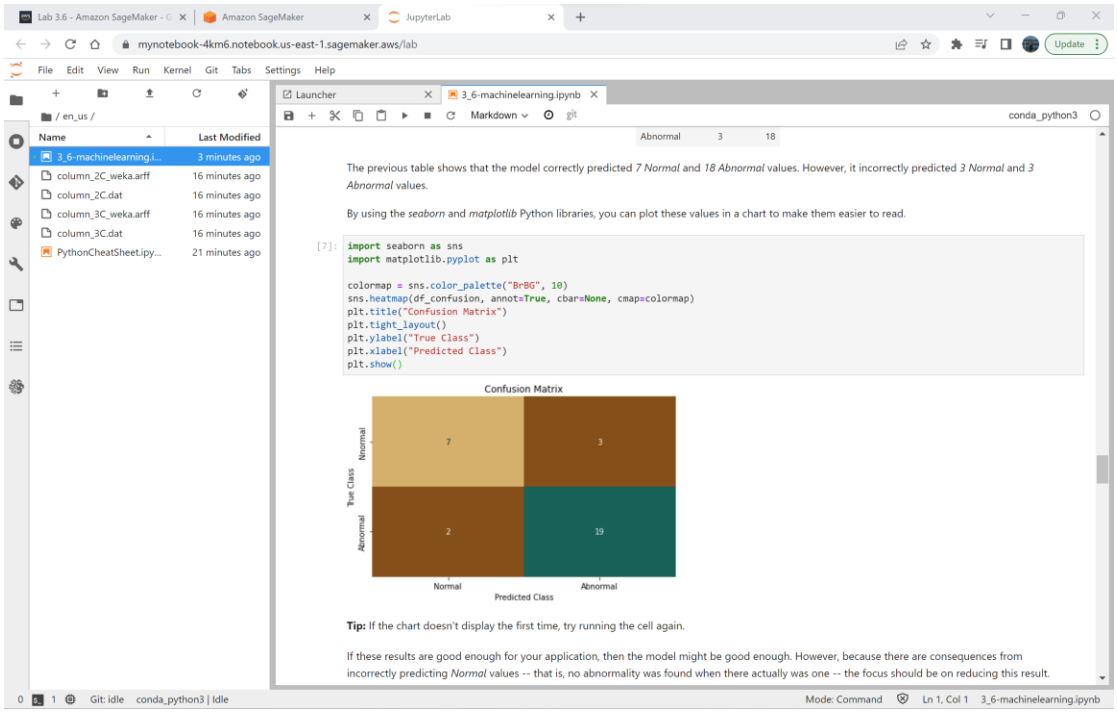
The result of training the model. Here we can see the how does model work first it start the training job, the create the instances for training, then download input data, train the image, uploaded generated training model after that training job completed.



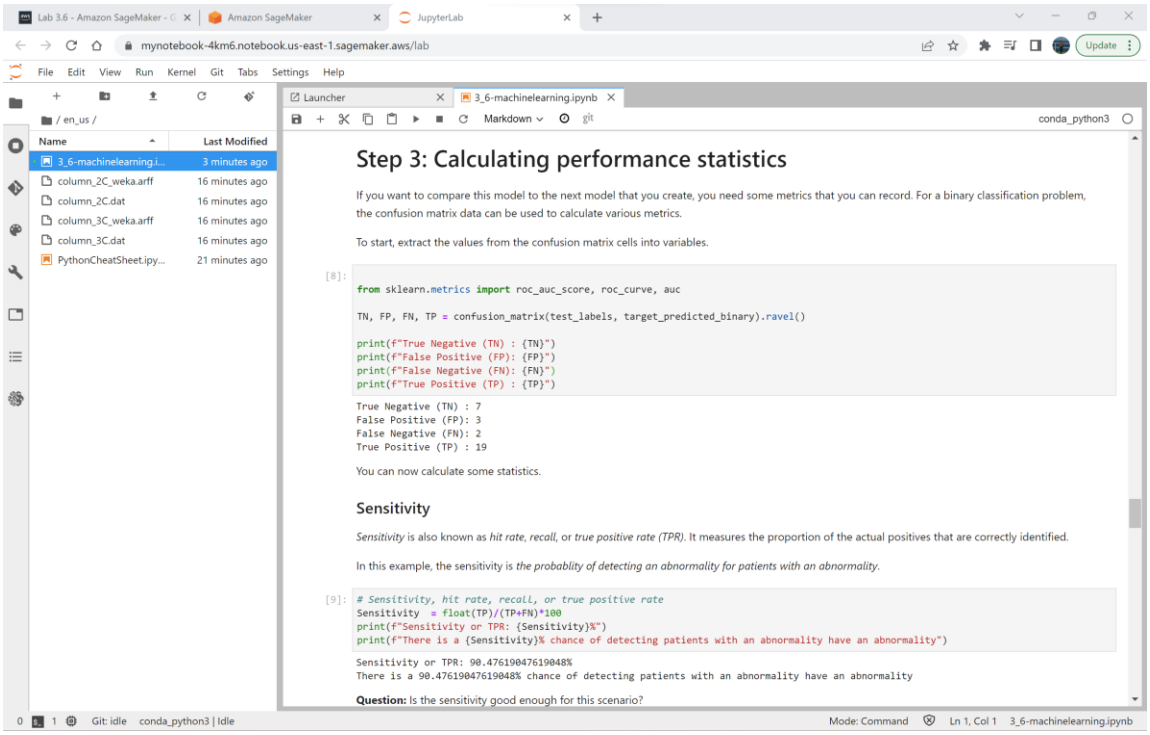
Now, Step 1 is to explore the result. Based on result we can visualize that the initial model might not be that good. Sometimes its difficult to tell by just comparing a few values.



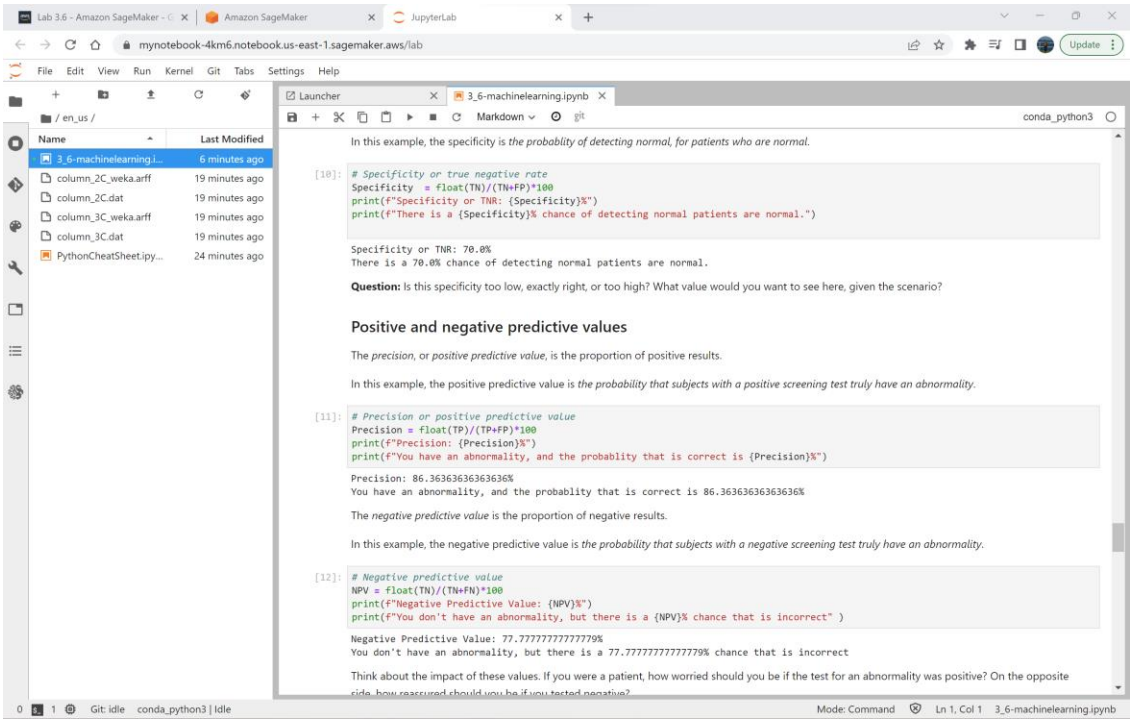
In step 2 -> Creating a confusion matrix. For that we import the model form sklearn.metrics called confusion matrix. We load that passing two parameter called test_labels, and target_predicted_binay.



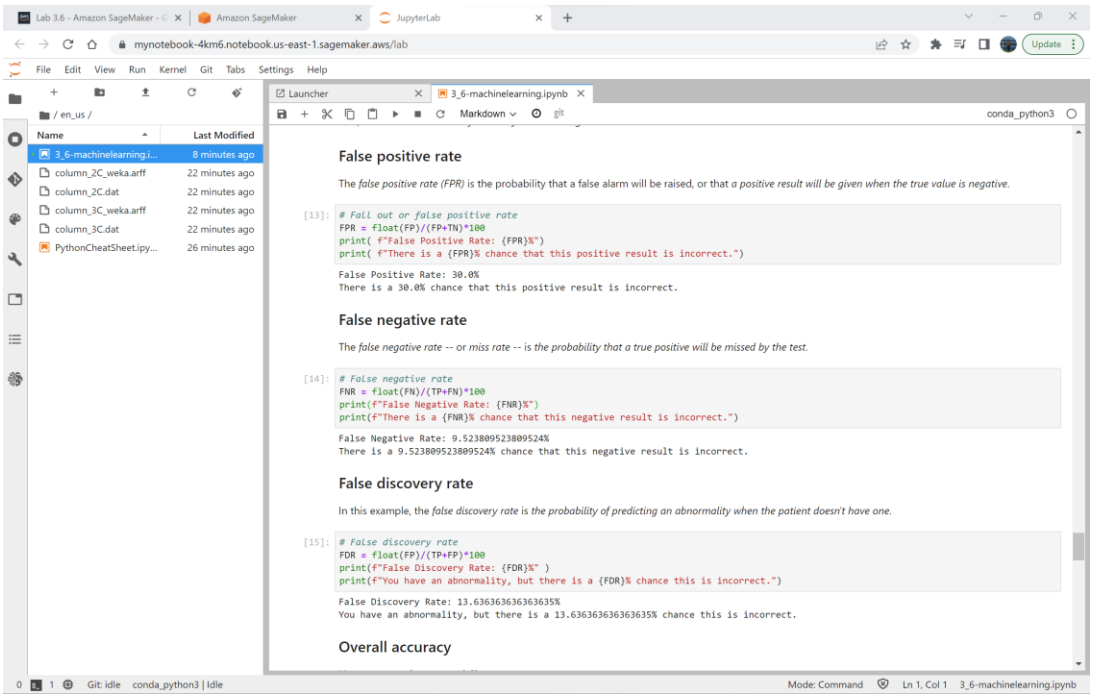
Here, we can visualize same result in heatmap using confusion matrix. These result are good enough for my application so the model might be good enough.



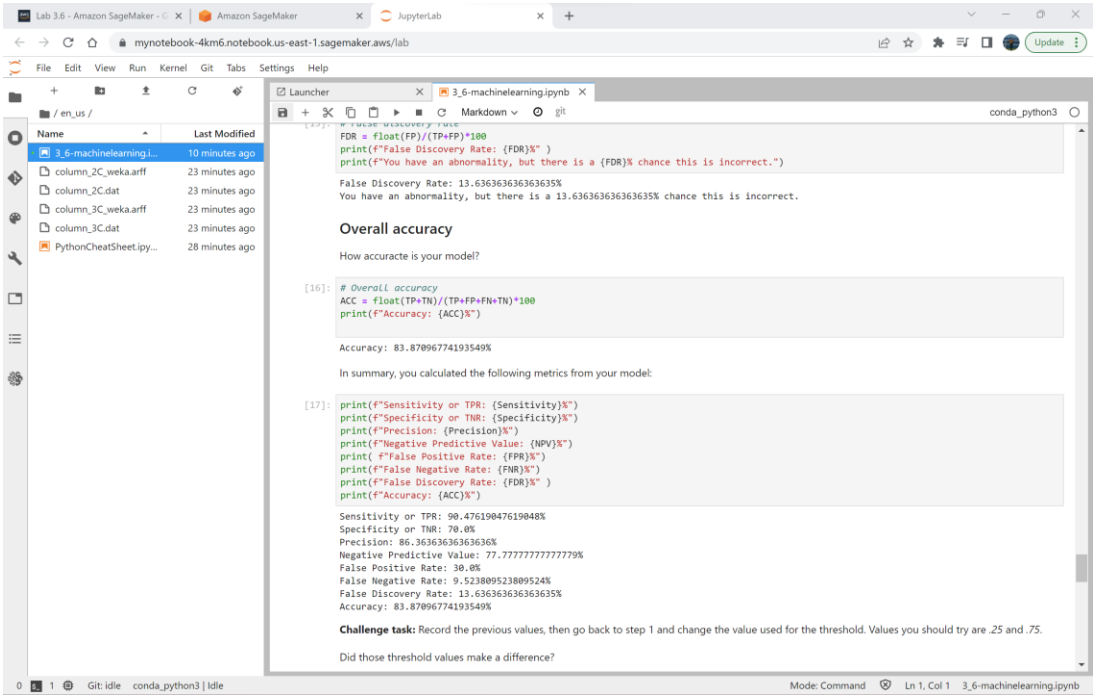
Step 3 is to find out the performance statistics like True Negative, False Positive, False Negative, True positive. After the we find out the sensitivity, hit rate, recall or true positive rate of that data.



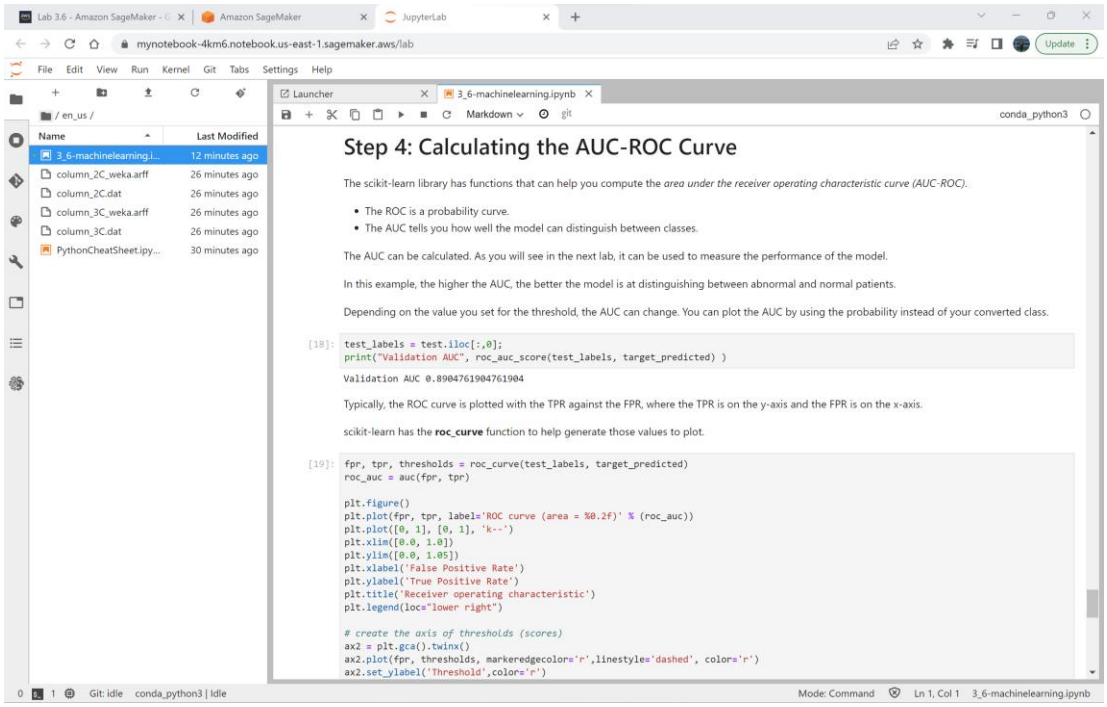
Here, we have the positive and negative predictive vales which is also the proportion of positive results. We got precision: **86,36** where as the negative predictive value is the proportion of negative results. We got negative predictive value : **77.77.**



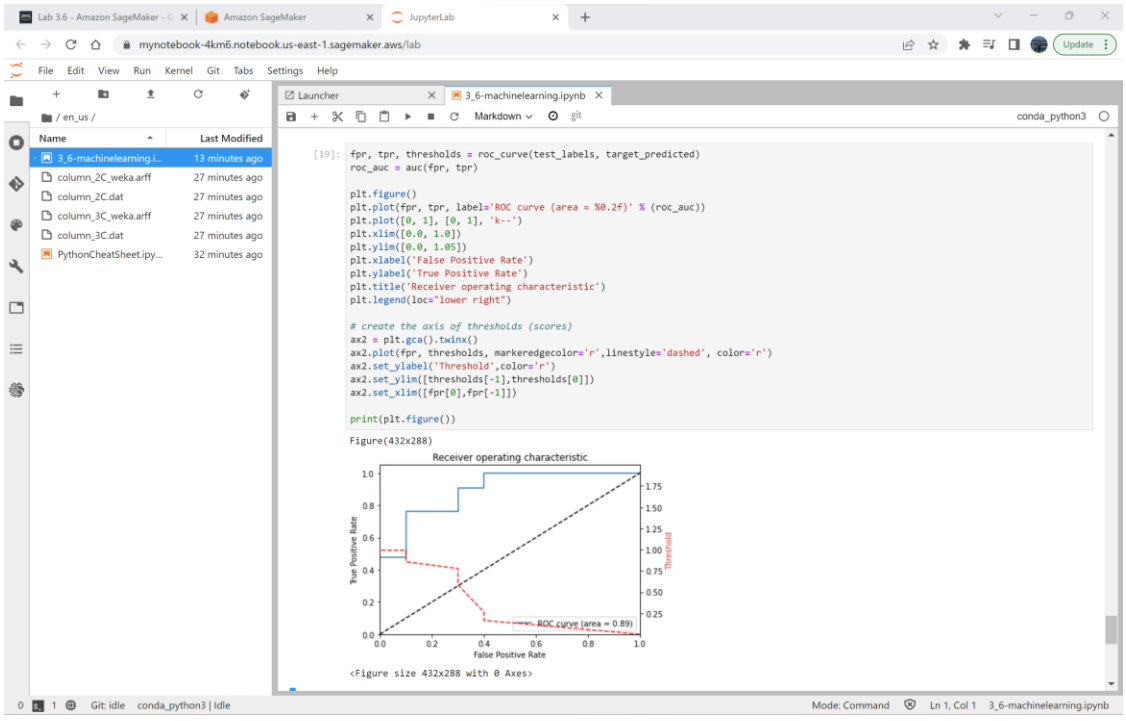
Here we find out the **false positive rate** which is around **30 %**, **false negative** rate which is around 9.52% and false discovery rate which is **13.63%.**



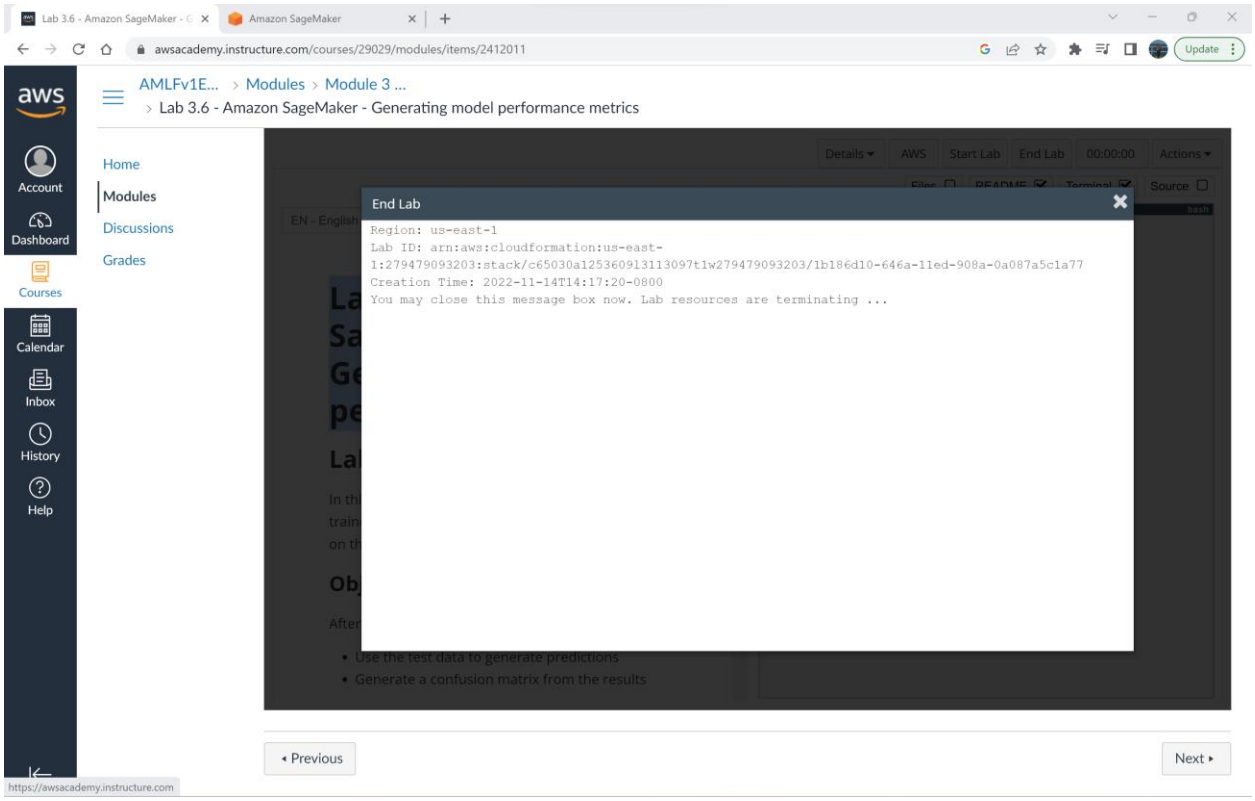
Here is overall accuracy of performance including: sensitivity, specificity, precision**, NPV, FPR, FNR, FDR, ACC .**we out accuracy of **83.87.**



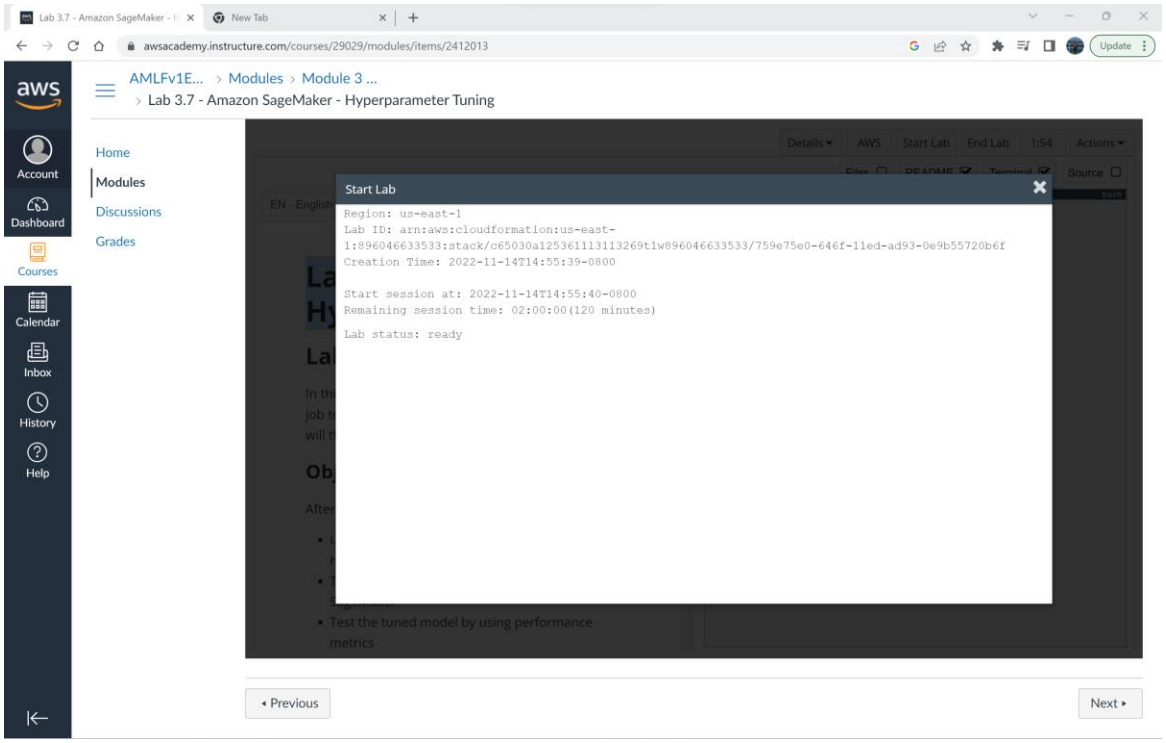
In step 4, we calculate the AUC-ROC curve, and below is the visualization of rov_curve.

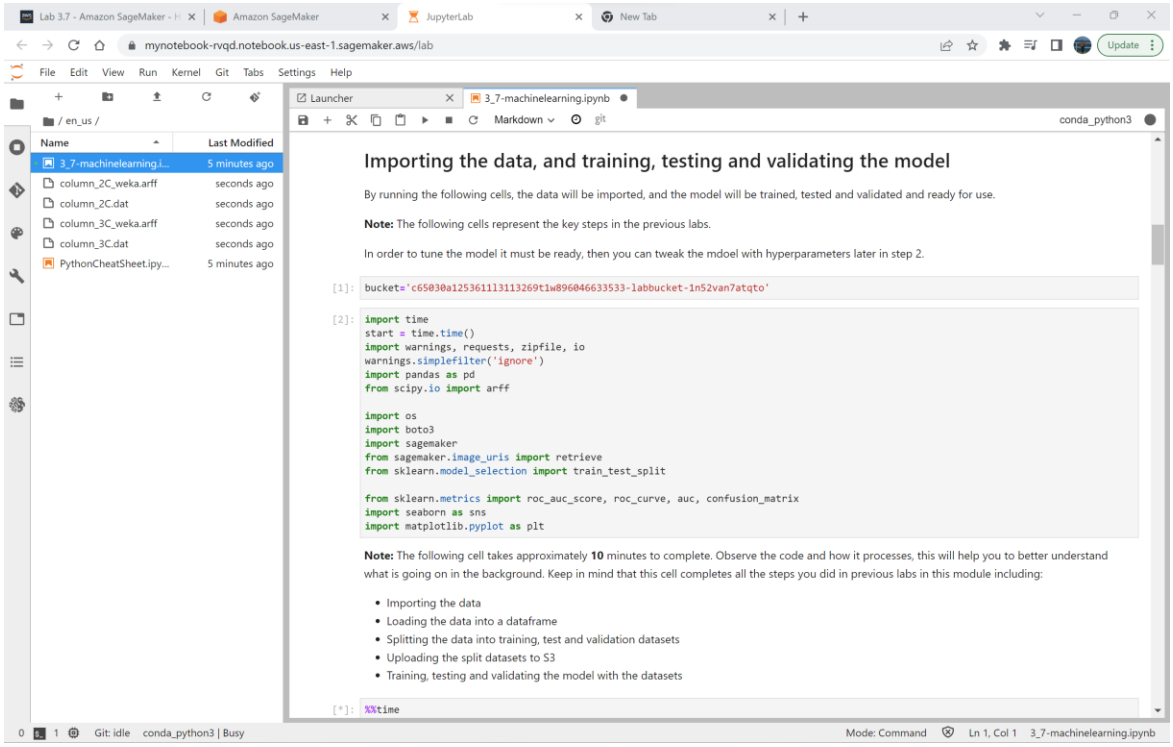


That’s all for this lab, we completed all the step that are required for this lab sucessfuly.

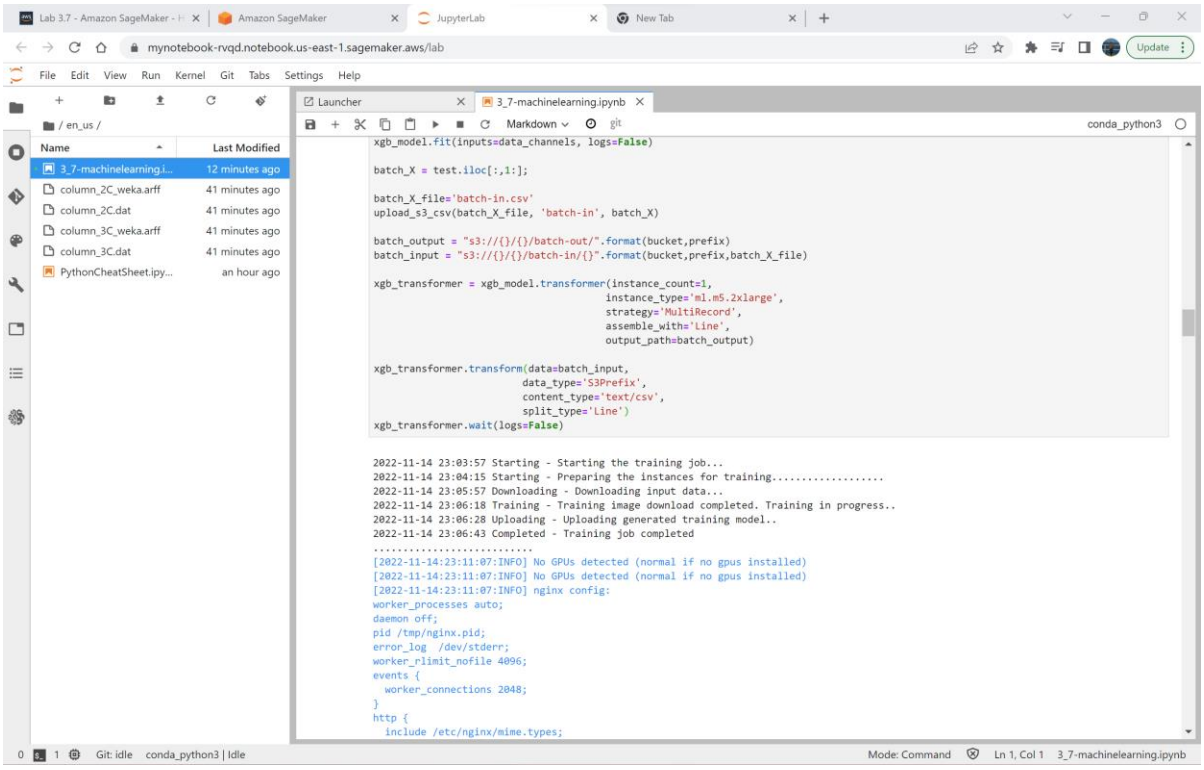

Lab 3.6 - Amazon SageMaker
Amazon SageMaker
awsacademy.instructure.com/courses/29029/modules/items/2412011
Update
AMLFv1E... > Modules > Module 3 ...
Lab 3.6 - Amazon SageMaker - Generating model performance metrics
Account
Dashboard
Courses
Calendar
Inbox
History
Help
Home
Modules
Discussions
Grades
Details
AWS
Start Lab
End Lab
00:00:00
Actions
Source
End Lab
Region: us-east-1
Lab ID: arn:aws:cloudformation:us-east-1:279479093203:stack/c65030a1253609131130971w279479093203/1b186d10-646a-11ed-908a-0a087a5c1a77
Creation Time: 2022-11-14T14:17:20-0800
You may close this message box now. Lab resources are terminating ...
Use the test data to generate predictions
Generate a confusion matrix from the results
Previous
Next
https://awsacademy.instructure.com

Lab 3.7 - Tuning Hyperparameters

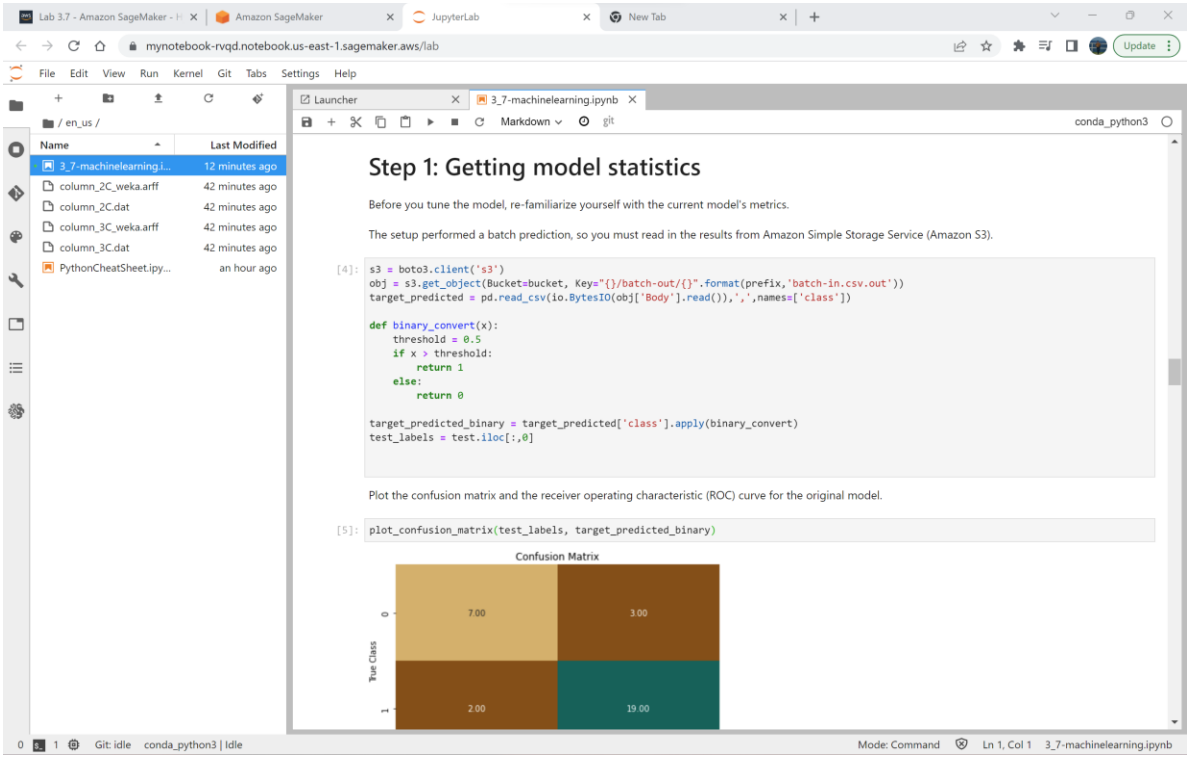


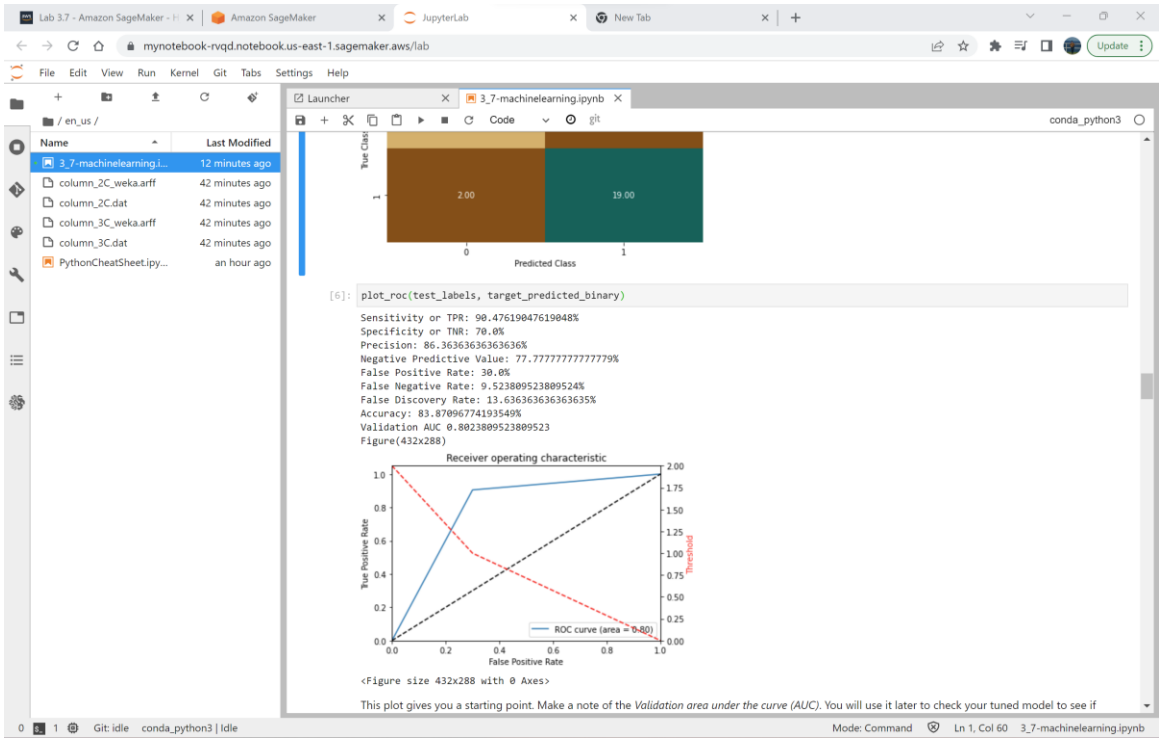


First, we import the data and do the training and testing along with validation. After that we upload the split datasets to S3.



Here the job training is been process with complete job training going through all the required steps.





Here are the statistics of the data the we have for test labels and target pred.

We can see that Sensitivity or TPR: 90.47

Specificity or TNR: 70.0%

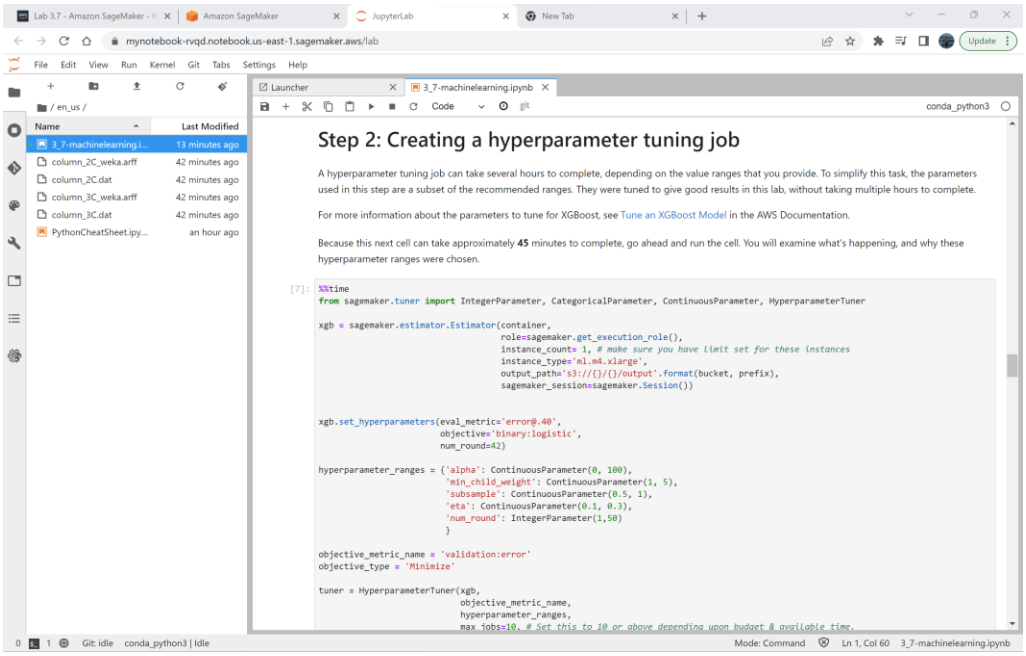
Precision: 86.36

Negative Predictive Value: 77.7

False Positive Rate: 30%

False Negative Rate: 9.52%

Accuracy:83.87%



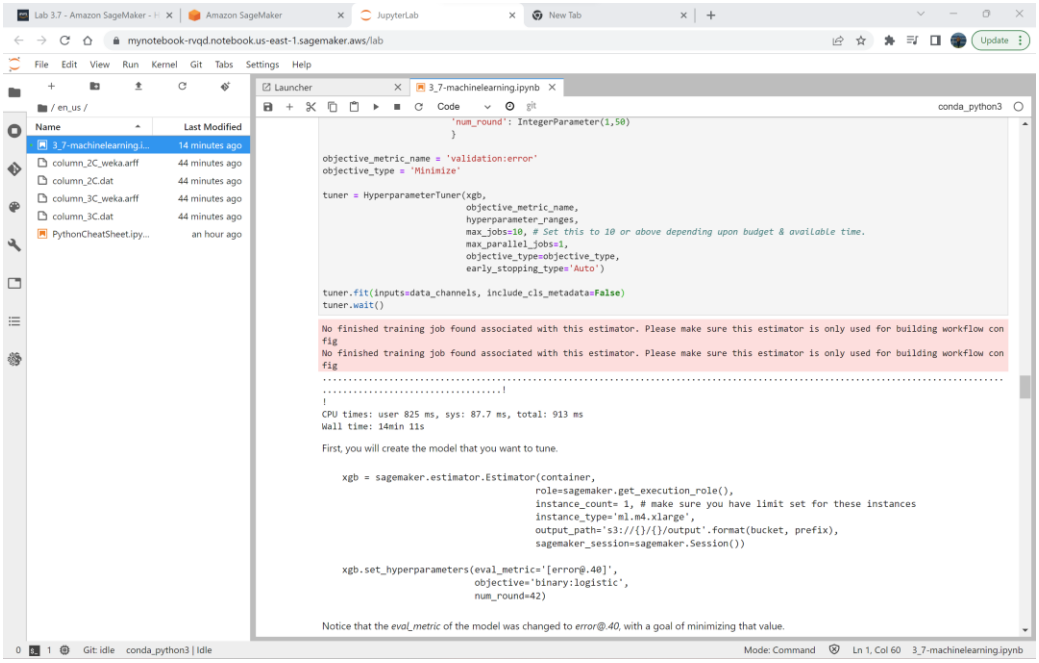
Step 2: Creating a hyperparameter tuning job

A hyperparameter tuning job can take several hours to complete, depending on the value ranges that you provide. To simplify this task, the parameters used in this step are a subset of the recommended ranges. They were tuned to give good results in this lab, without taking multiple hours to complete.

For more information about the parameters to tune for XGBoost, see Tune an XGBoost Model in the AWS Documentation.

Because this next cell can take approximately **45** minutes to complete, go ahead and run the cell. You will examine what's happening, and why these hyperparameter ranges were chosen.

```
%%time
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousParameter, HyperparameterTuner

xgb = sagemaker.estimator.Estimator(container,
                                    role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make sure you have limit set for these instances
                                    instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket, prefix),
                                    sagemaker_session=sagemaker.Session())


xgb.set_hyperparameters(eval_metric='error@.40',
                        objective='binary:logistic',
                        num_round=42)

hyperparameter_ranges = {'alpha': ContinuousParameter(0, 100),
                         'min_child_weight': ContinuousParameter(1, 5),
                         'subsample': ContinuousParameter(0.5, 1),
                         'eta': ContinuousParameter(0.1, 0.3),
                         'num_round': IntegerParameter(1,50)
                         }

objective_metric_name = 'validation:error'
objective_type = 'Minimize'

tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or above depending upon budget & available time.
```



```
                            'num_round': IntegerParameter(1,50)
                            }

objective_metric_name = 'validation:error'
objective_type = 'Minimize'

tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or above depending upon budget & available time.
                            max_parallel_jobs=1,
                            objective_type=objective_type,
                            early_stopping_type='Auto')

tuner.fit(inputs=data_channels, include_cls_metadata=False)
tuner.wait()
```

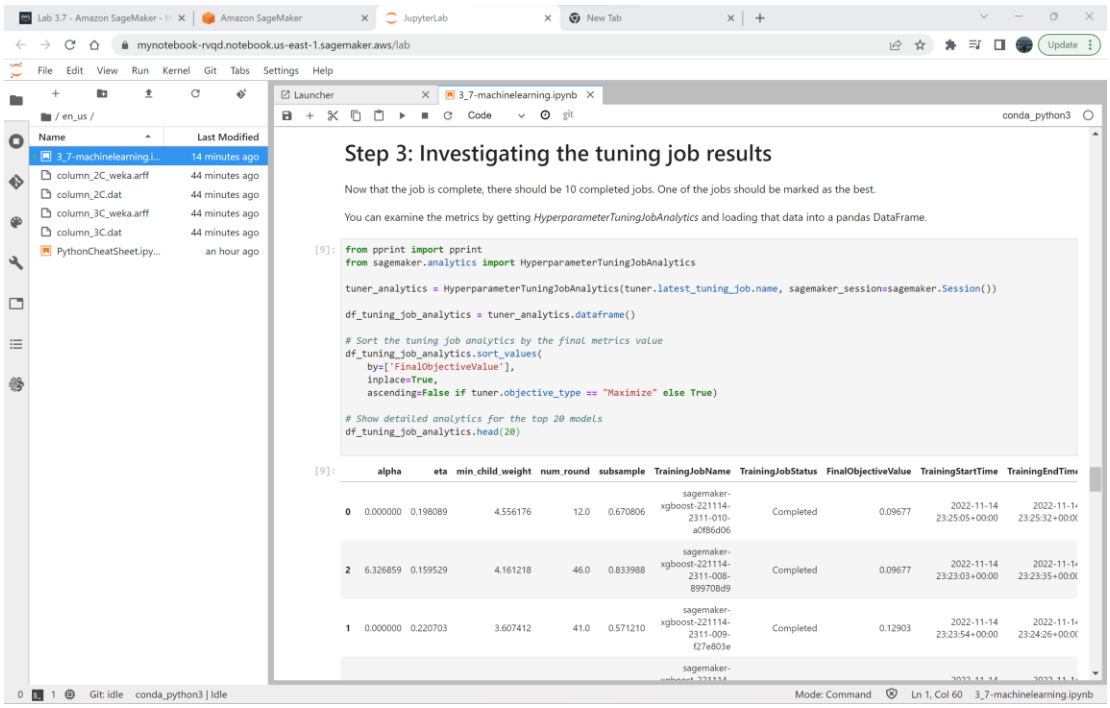
```
No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow con
fig
No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow con
fig
.....
.....!
!
CPU times: user 825 ms, sys: 87.7 ms, total: 913 ms
Wall time: 14min 11s
```

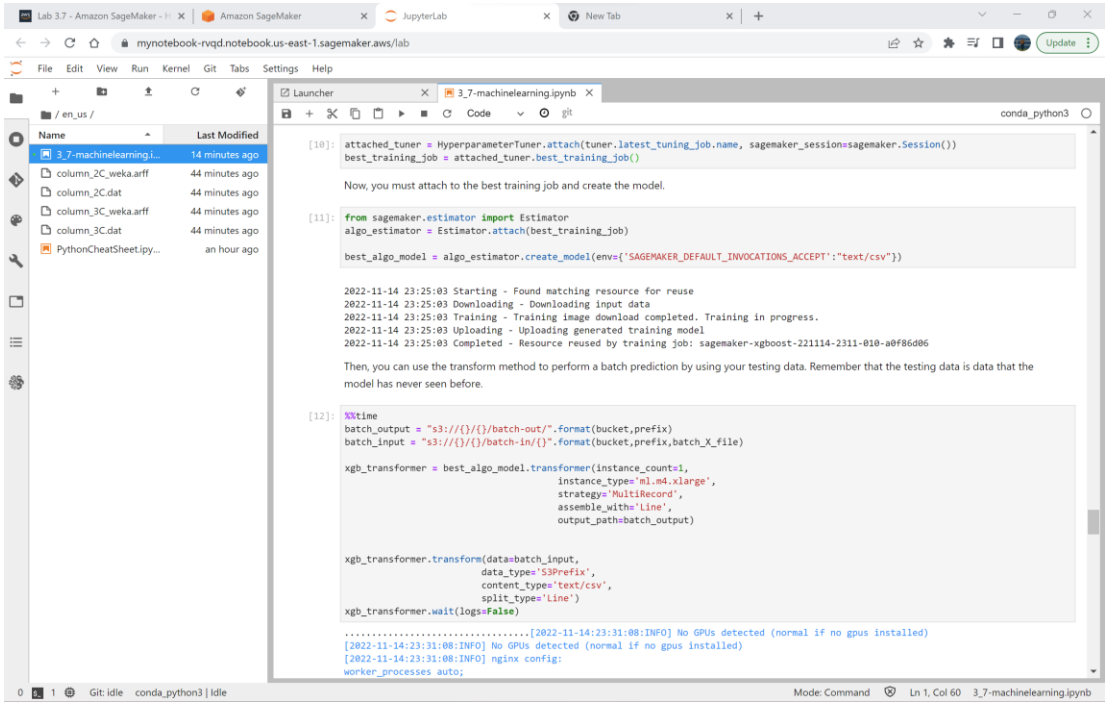
First, you will create the model that you want to tune.

```
    xgb = sagemaker.estimator.Estimator(container,
                                        role=sagemaker.get_execution_role(),
                                        instance_count= 1, # make sure you have limit set for these instances
                                        instance_type='ml.m4.xlarge',
                                        output_path='s3://{}/{}/output'.format(bucket, prefix),
                                        sagemaker_session=sagemaker.Session())

    xgb.set_hyperparameters(eval_metric='[error@.40]',
                            objective='binary:logistic',
                            num_round=42)
```

Notice that the *eval_metric* of the model was changed to *error@.40*, with a goal of minimizing that value.

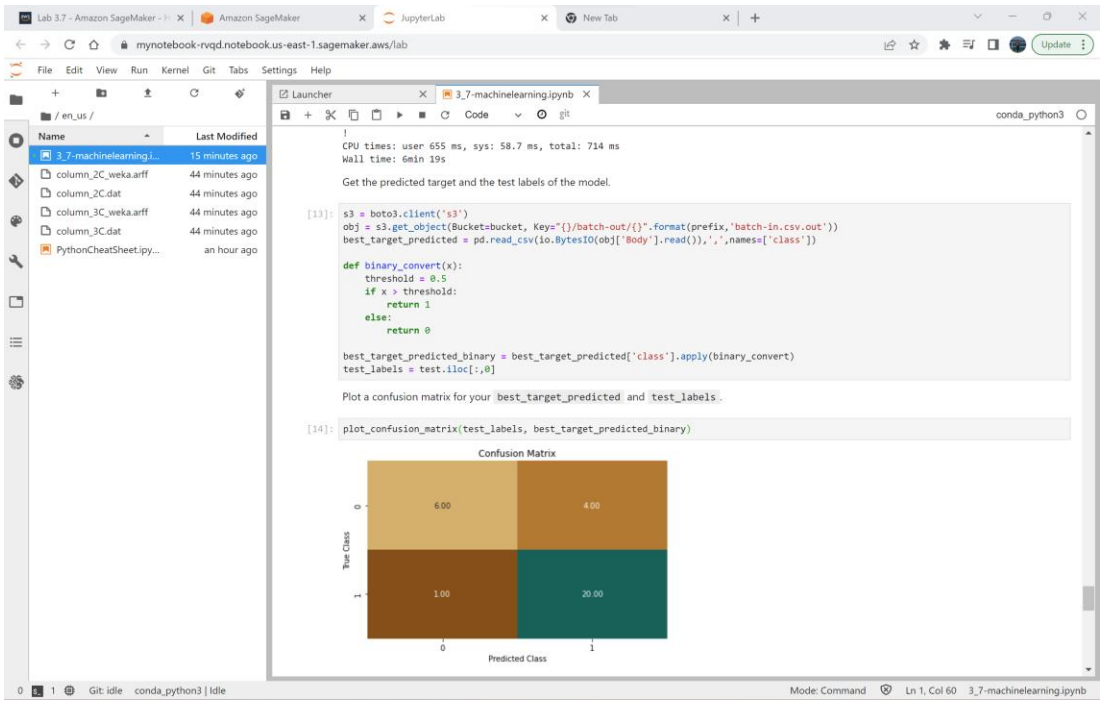
Now we are tuning our job using hyper parameter. A hyperparameter tuning job can take several hours to complete. The total time it took me to complete this job is **14min 11sec.**

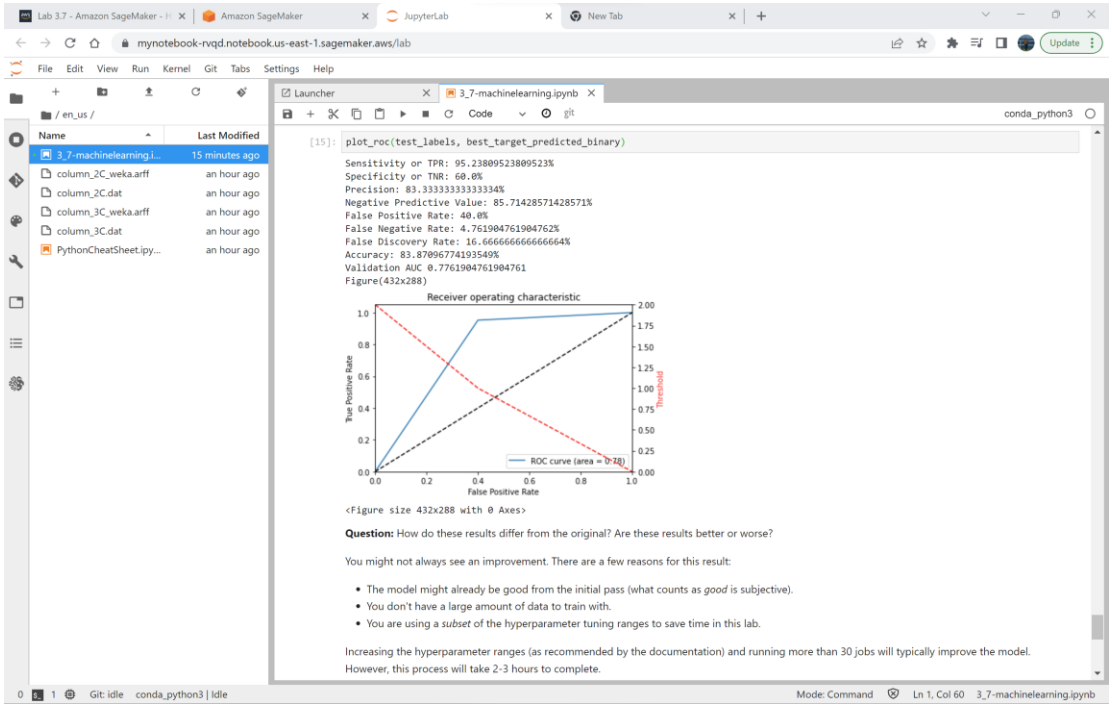




Step 3 is to Investigate the tuning job results. Here we loas the metrics of Hyperparameter Tuning Job into pandas' data frames.

After that we attach the best training model and create the model. We use the transform method to perform a batch prediction by using out testing data. Since testing data is data that the model has never seen before.





Here are the statistics of the data we have for test labels and target pred.

We can see that Sensitivity or TPR: 95.23

Specificity or TNR: 60.0%

Precision: 83.33

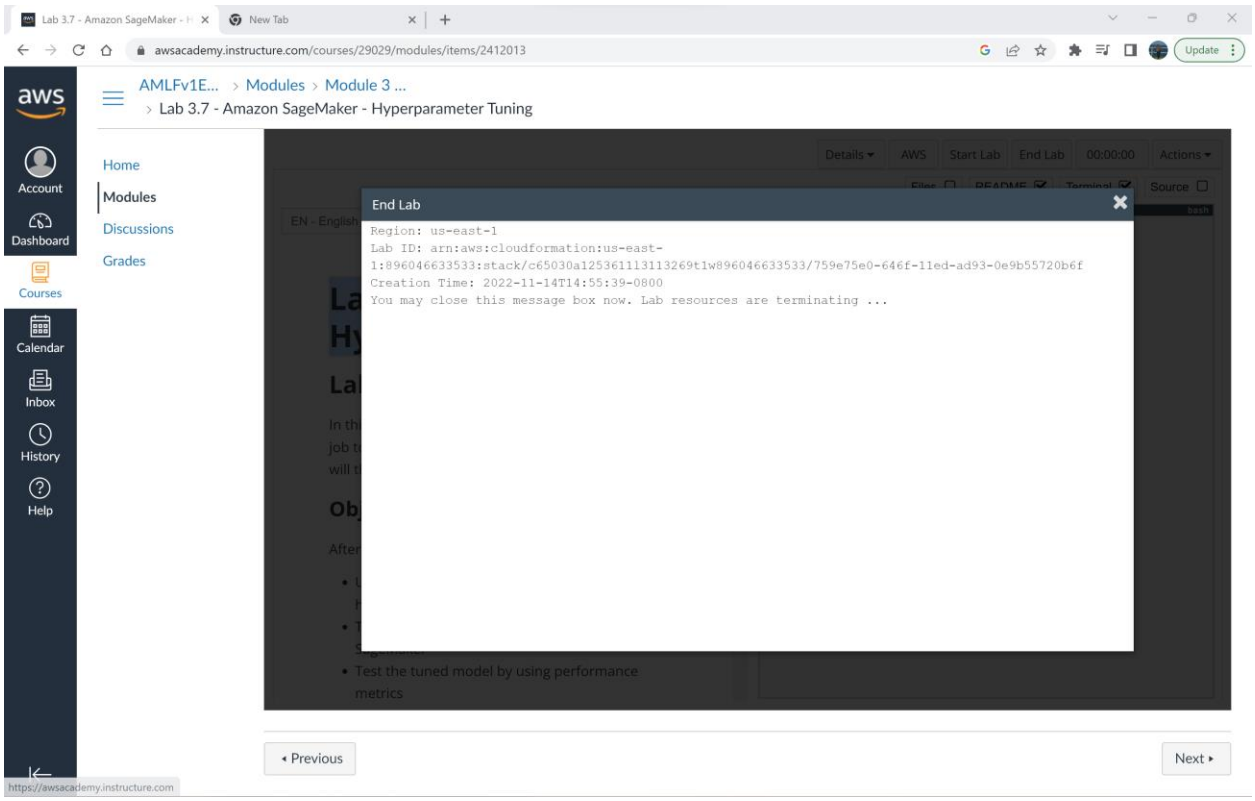
Negative Predictive Value: 85.71

False Positive Rate: 40.0%

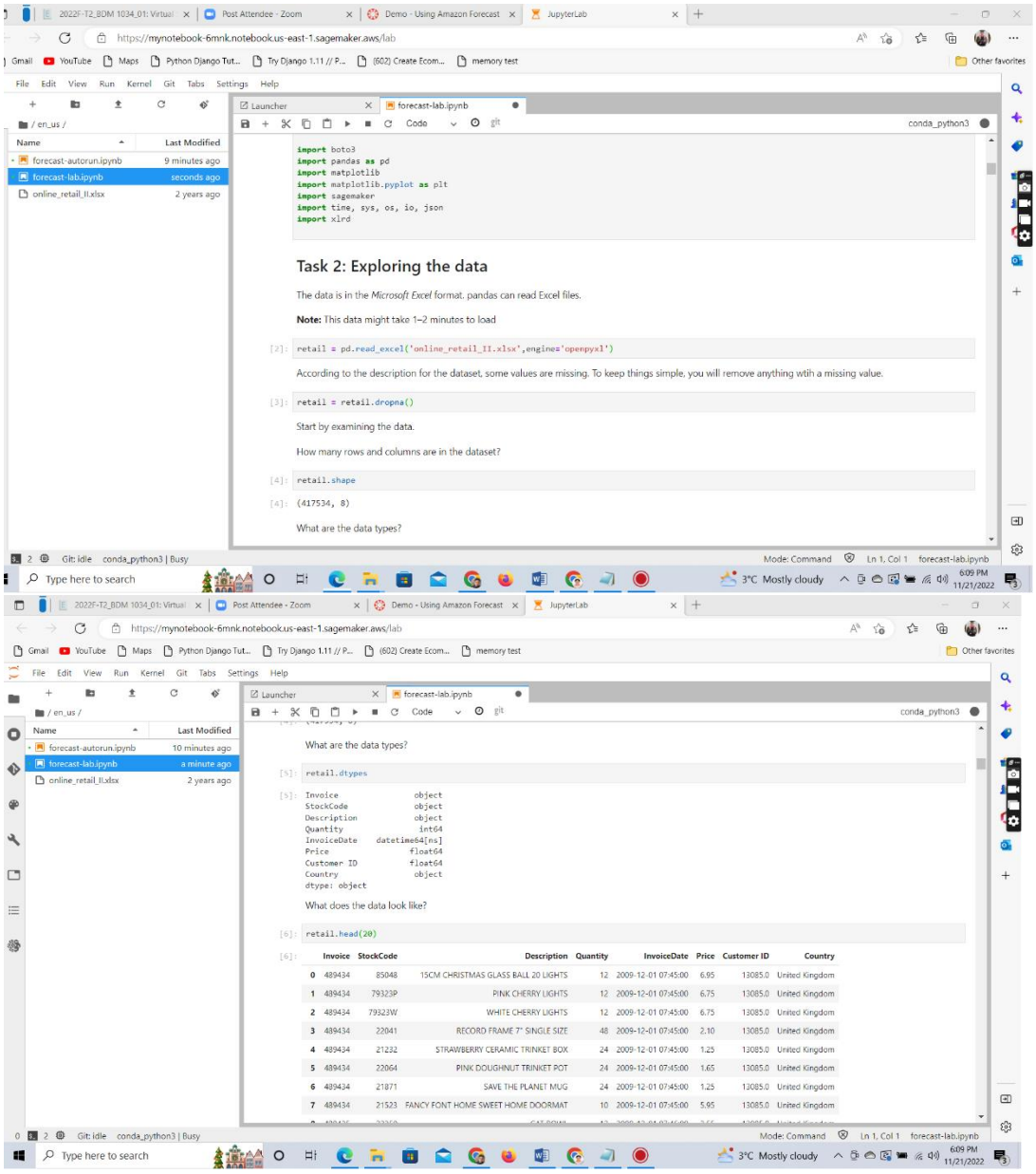
False Negative Rate: 4.76%

Accuracy:83.87%

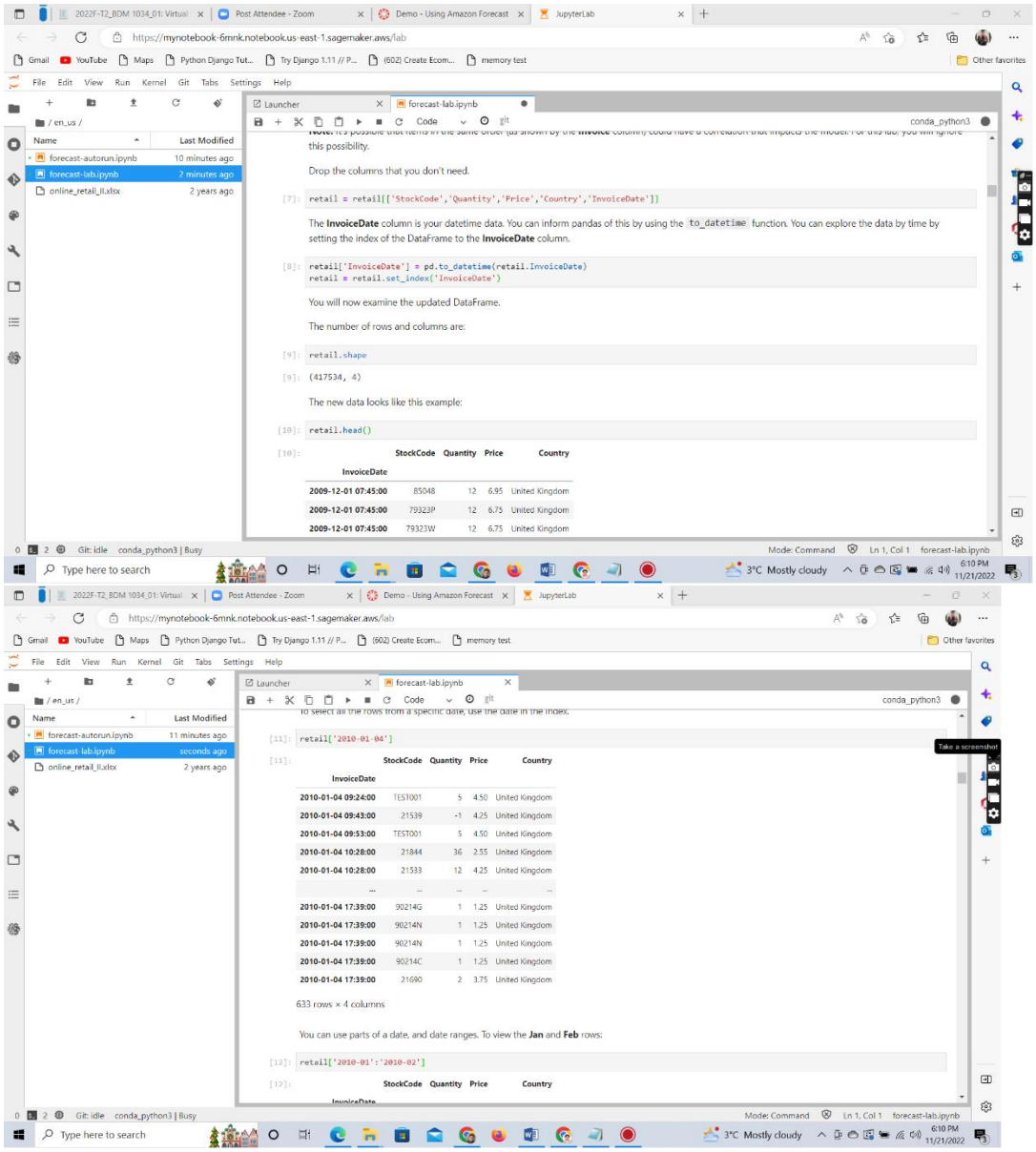
That's all the lab work to complete this lab. We completed and performed all the required task for the lab and analyse , compare the results between with or without hyperparameter tuning jobs.



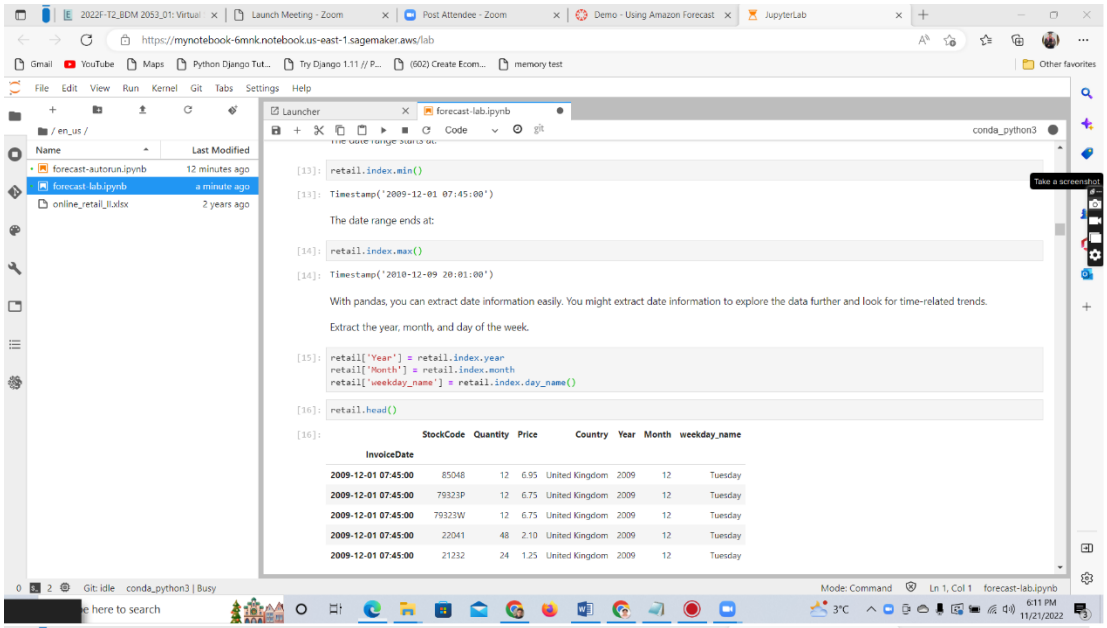
Lab 4 - Creating a forecast with Amazon Forecast

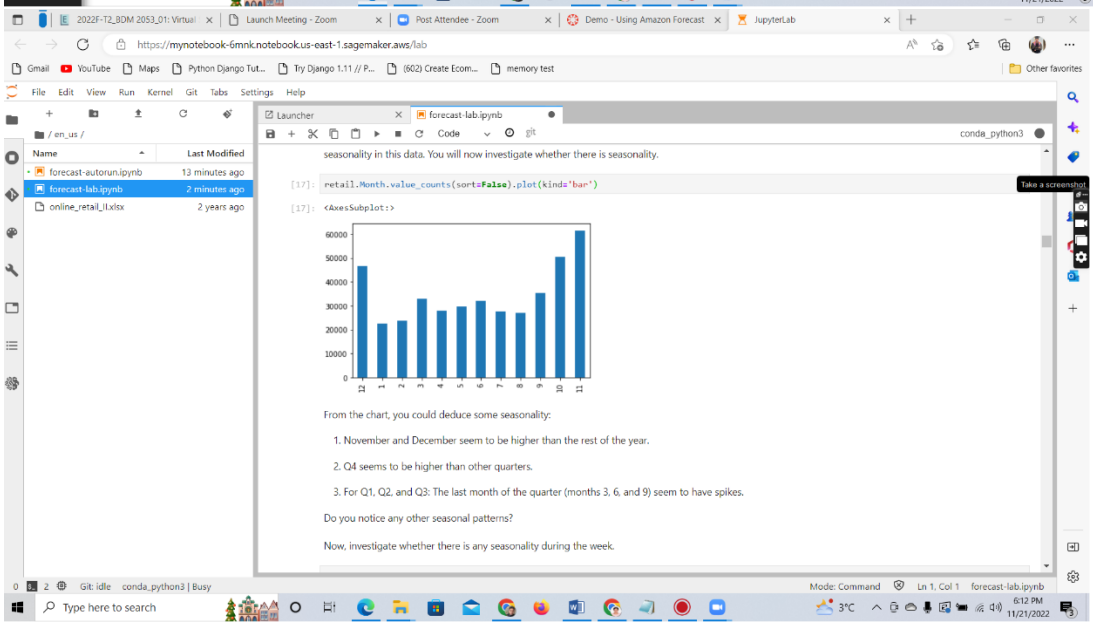


First, we import the data and explore the dataset which we have save into pandas dataframe. We look for the data types for this dataset . we view the data set. We can see that it has 8 columns which are Invoice, StockCode, Descirption, Quantity, InvoiceDate, Price, CustomerID and Country,

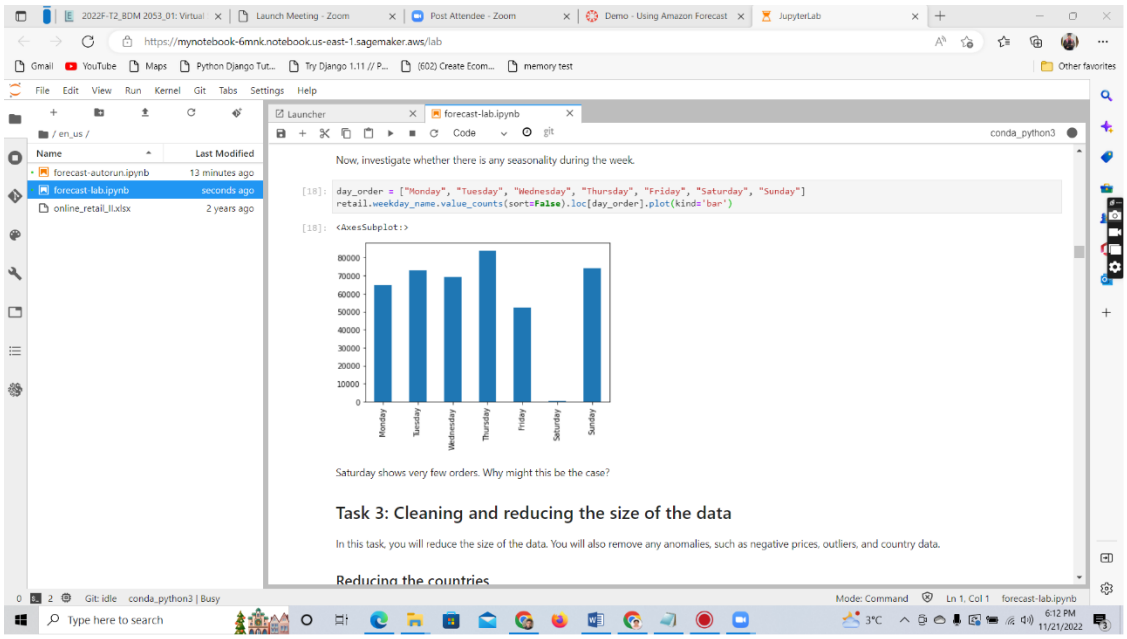


Here we going to drop the columns that we don't need. We just keeping stock code, quantity, price, country and invoice date. After that we convert the invoice date into to_datetime so that we can explore date by time by setting the index of the Dataframe to the invoice column.

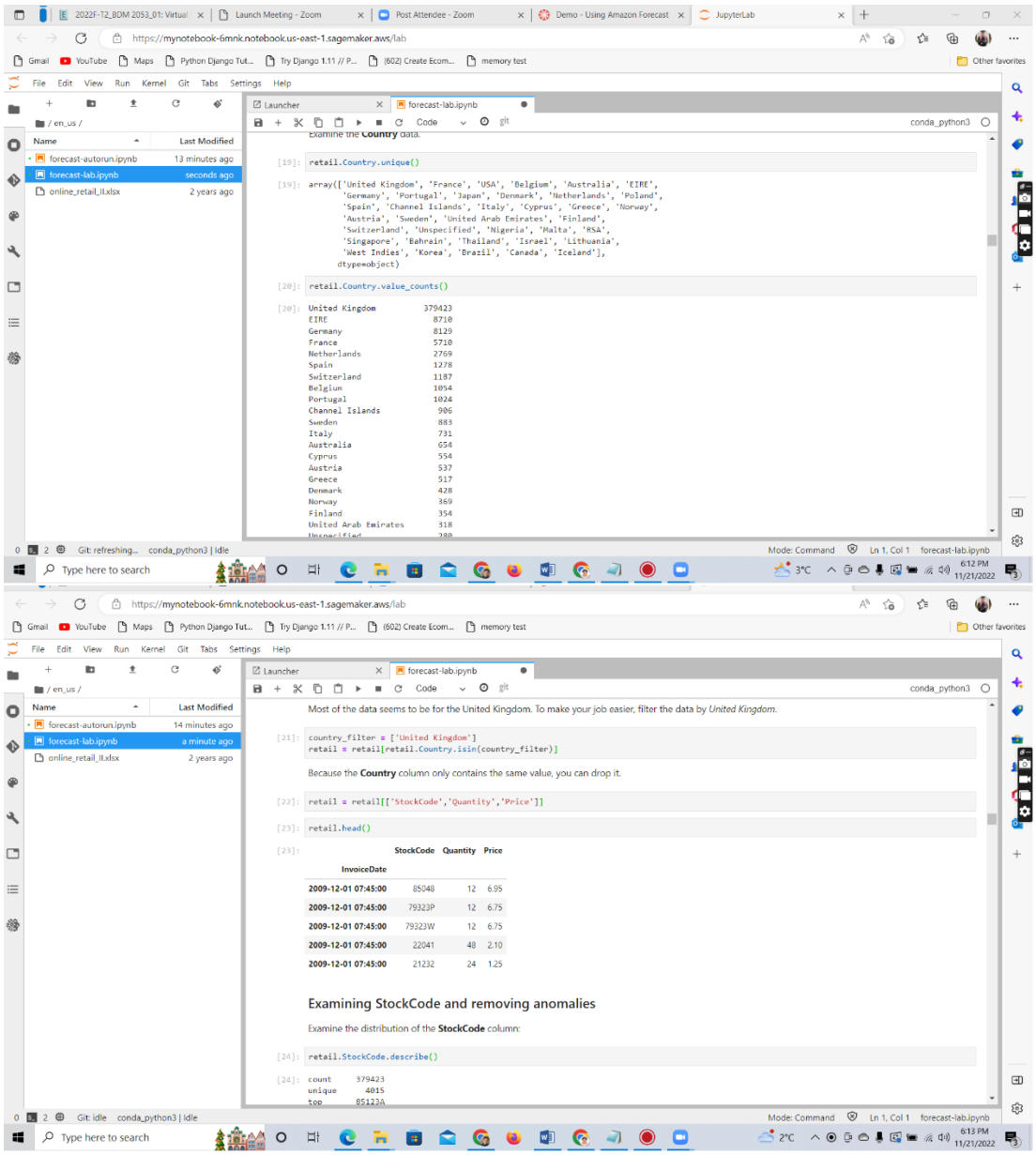




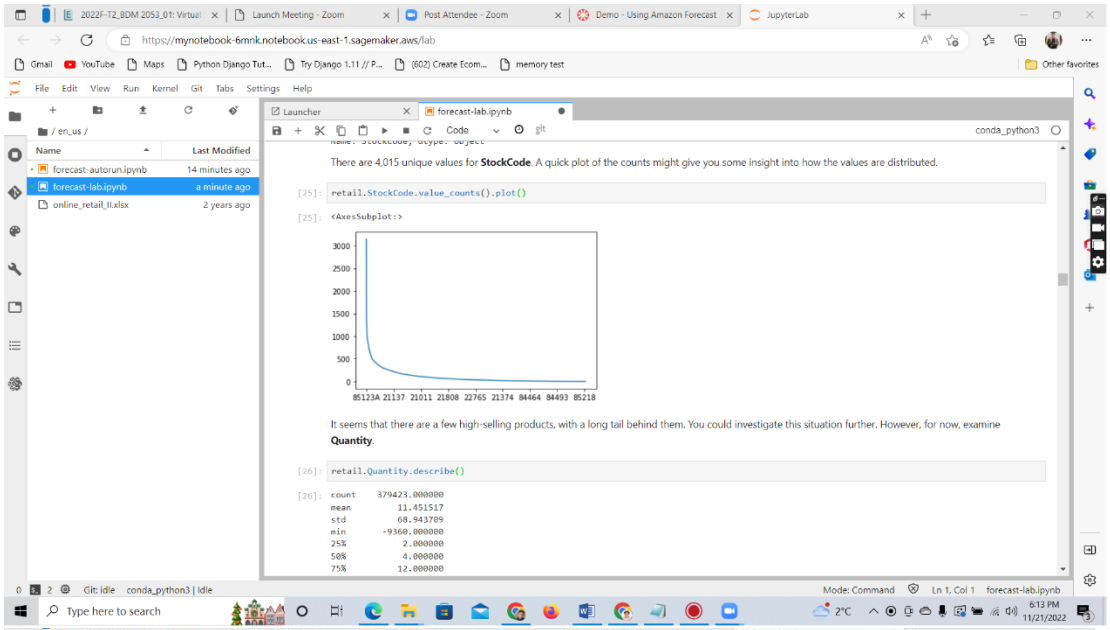
We create new three column year, month and weekday by extracting the information to look for time-related trends. From the chart you can see that Nov and dec seem to be higher than the rest of the year. Q4 seems to be higher than other quarters.

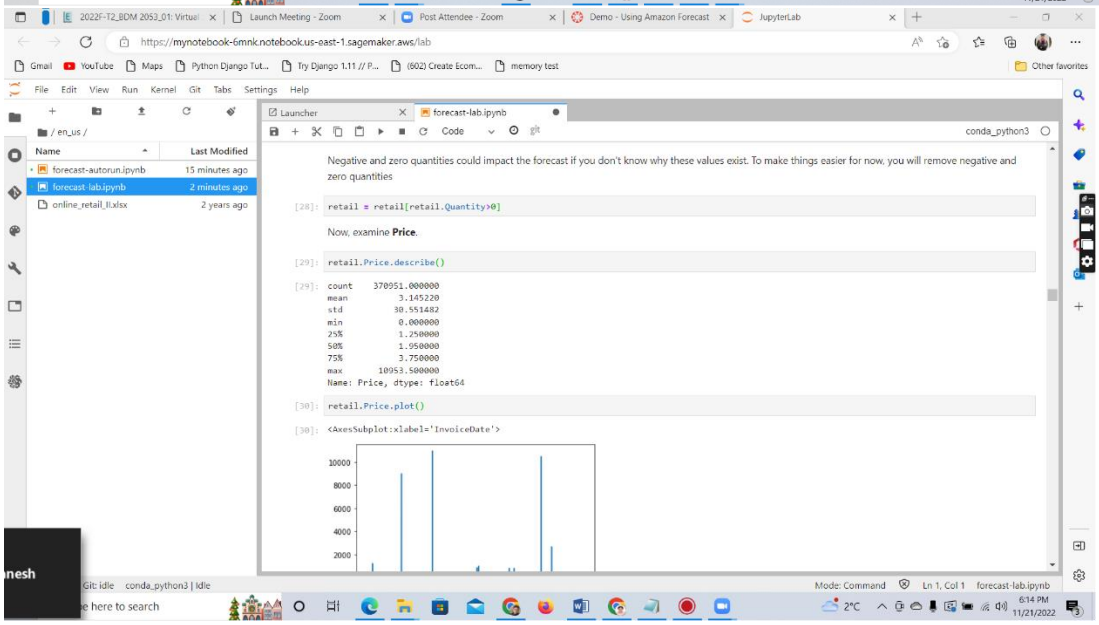


Conclusion, Saturday shows very few order cause most of shops are close on Saturday.

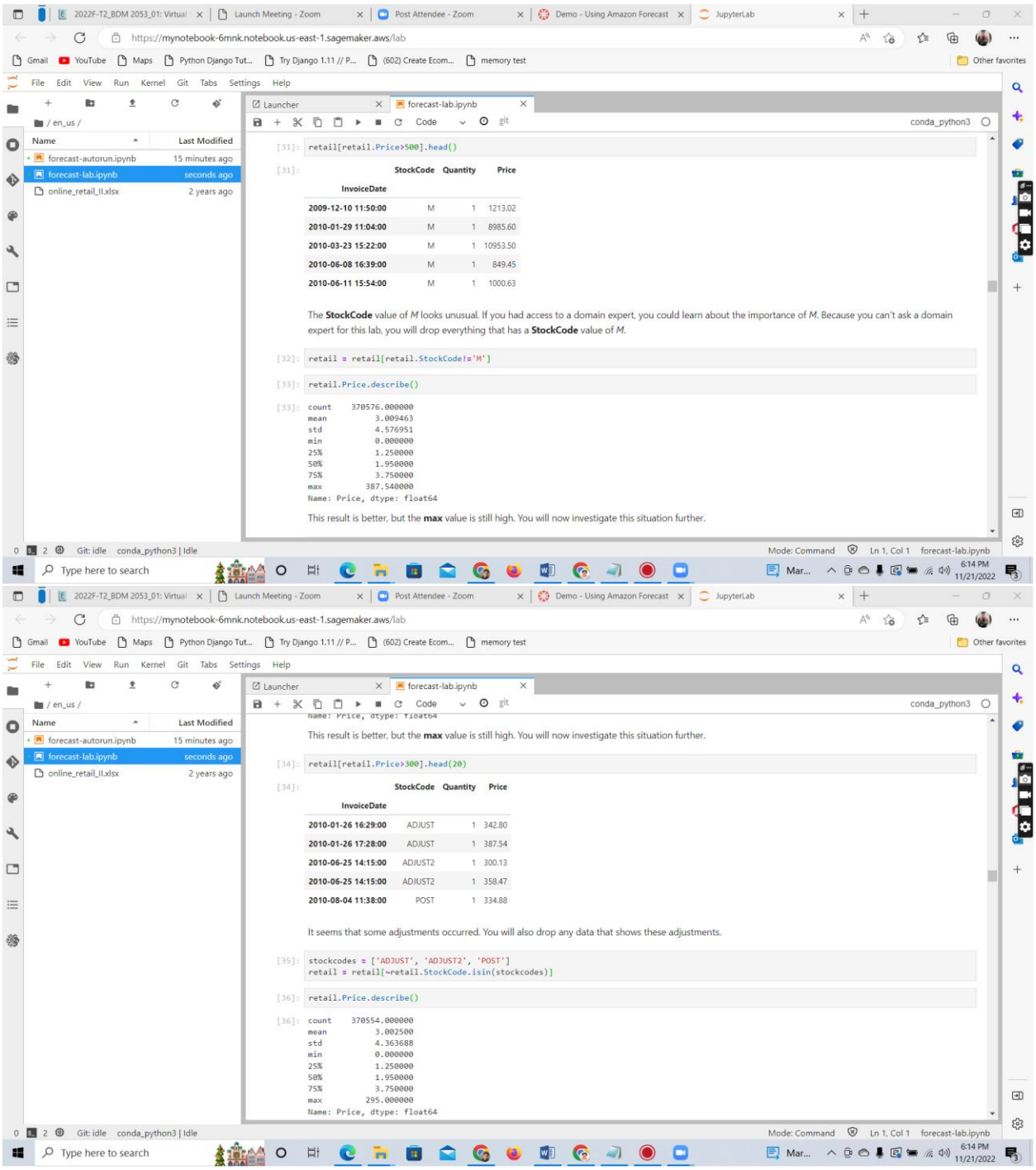


AS you can see most of the data are related to UK. We will just store data of UK for further analysis.

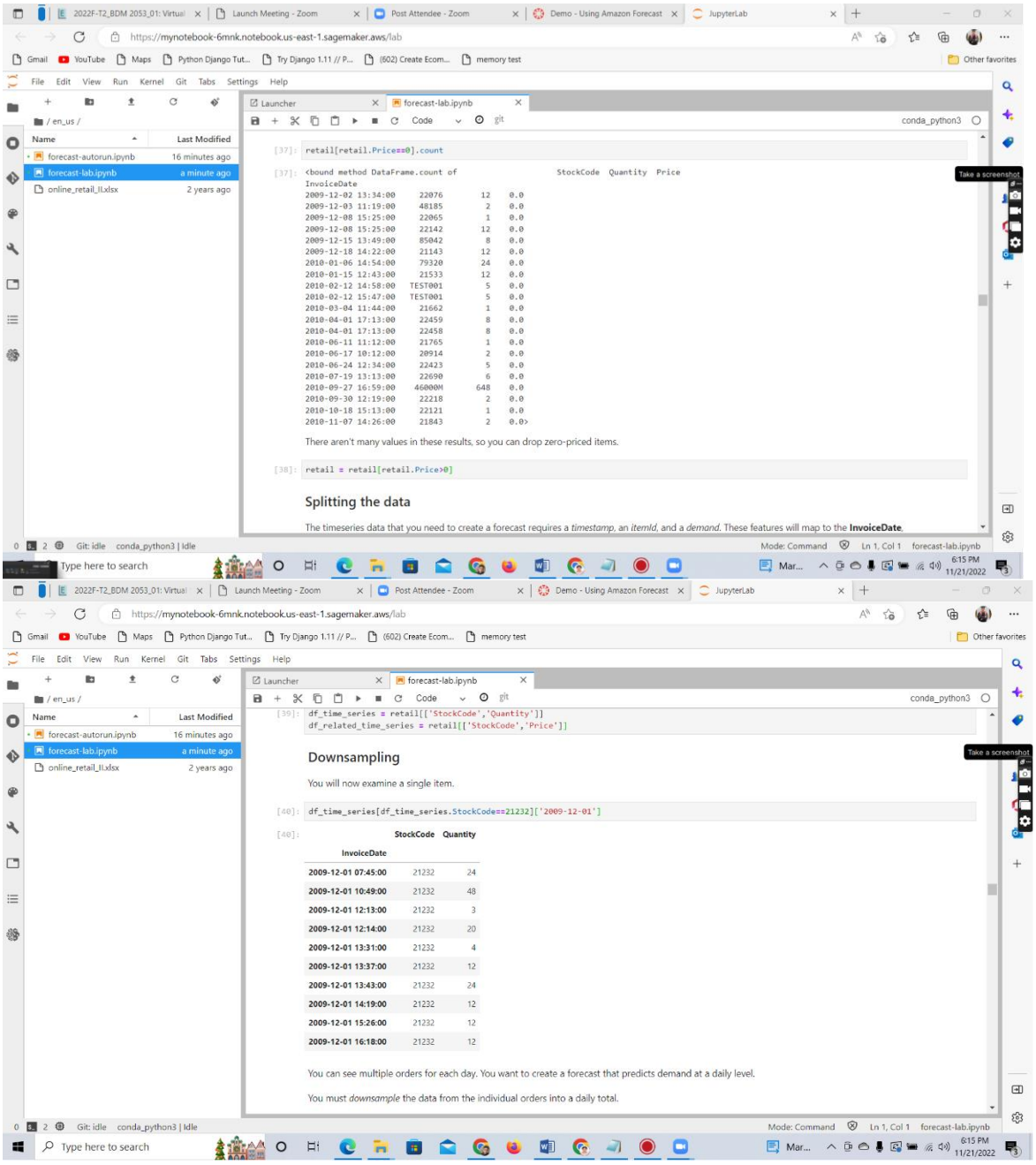




There are 4015 unique values for stock code. It seems that there are a few high selling products with a long tail behind them.we could investigate this situation further.

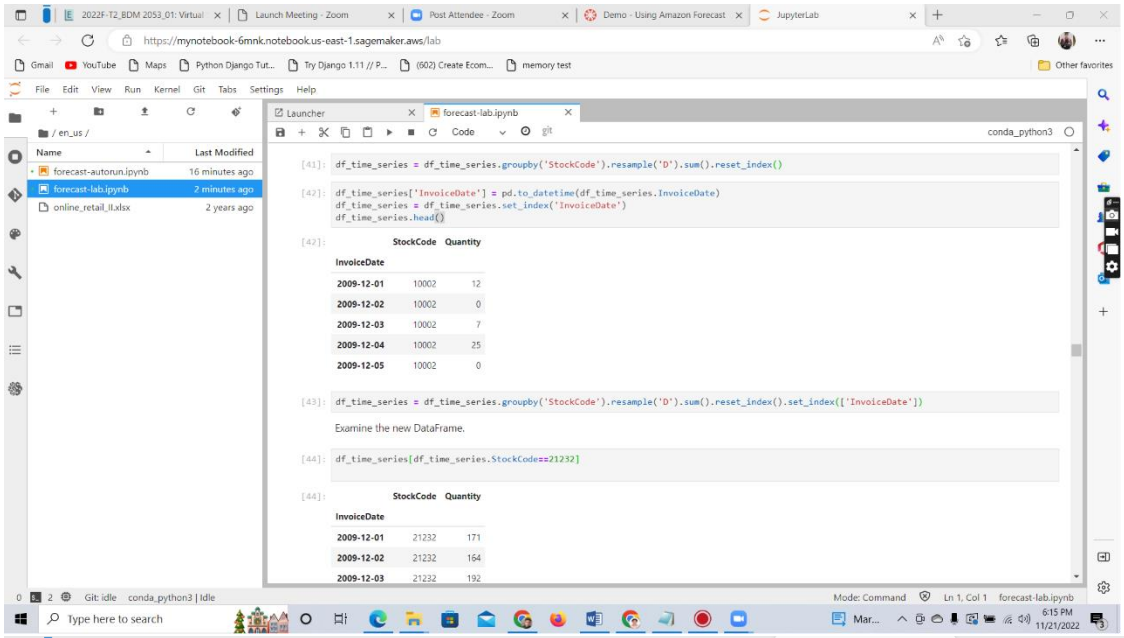


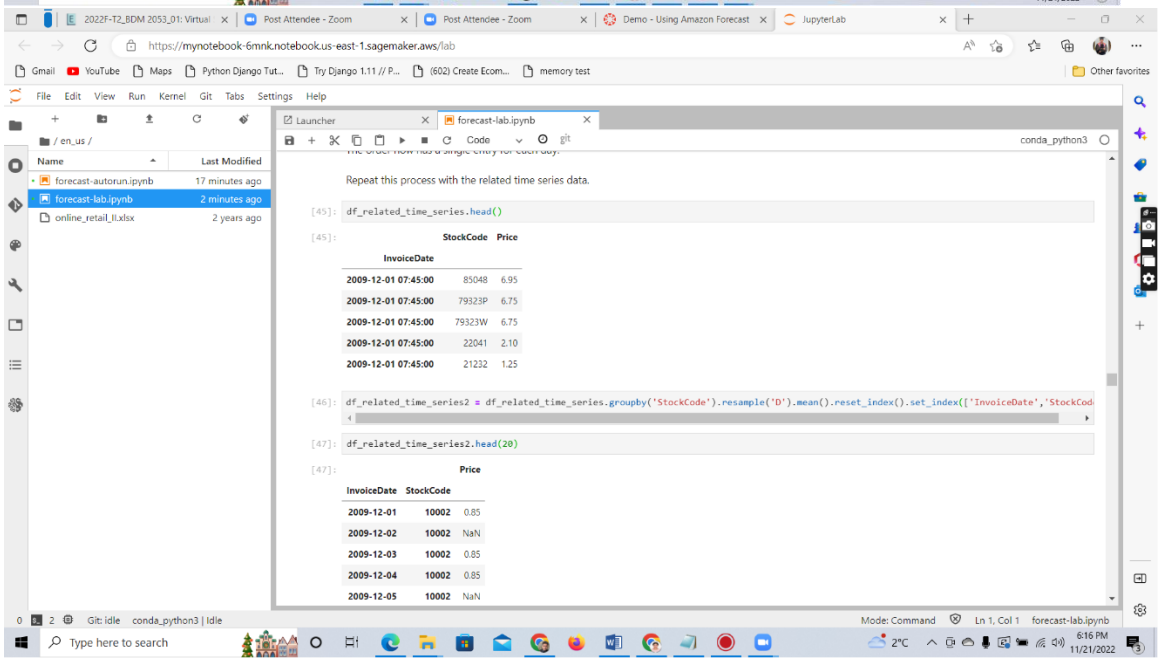
We have drop all the value related to stock code 'M'. This result is better, but the Max value is still high. It seems that some adjustment occurred. We will drop any data that shows these adjustment.

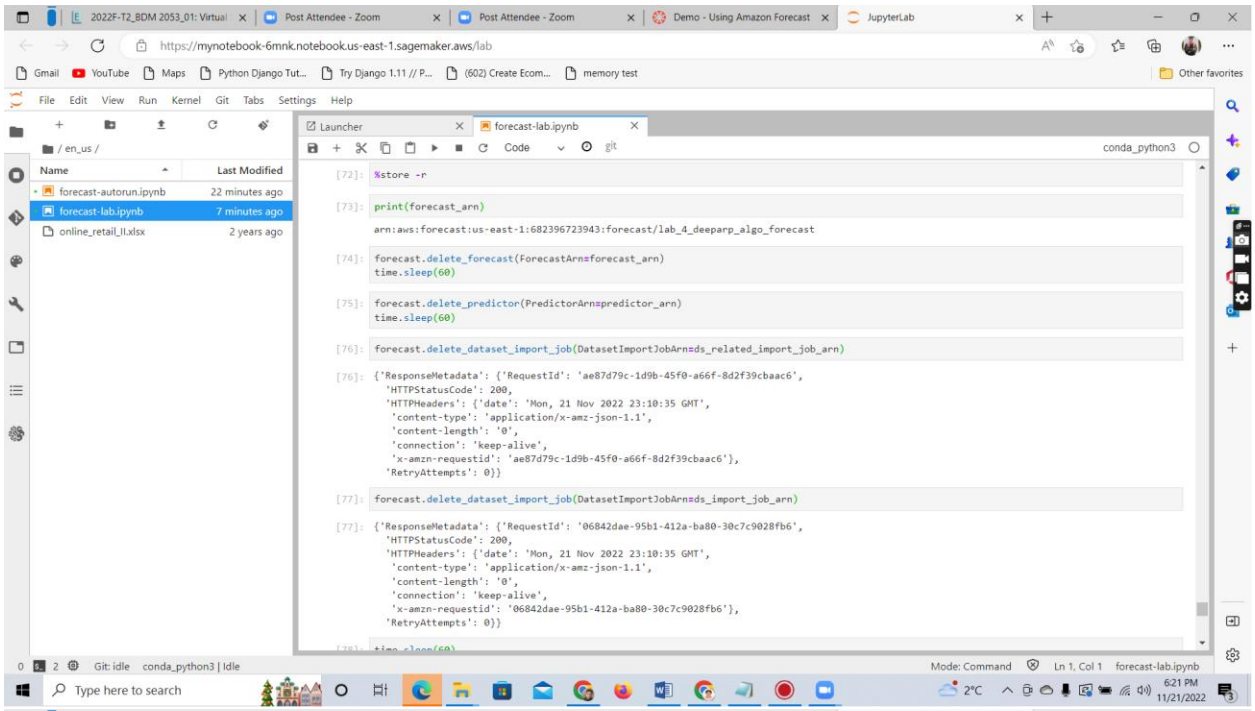


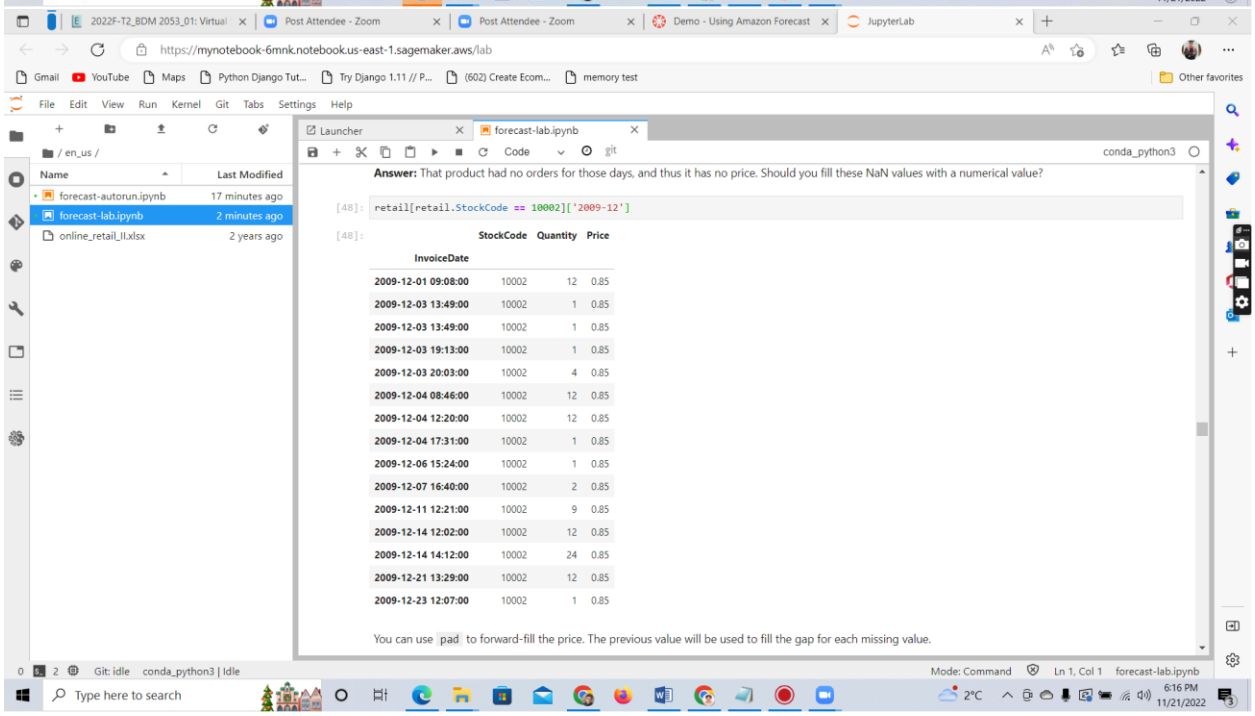
Here we are splitting the data as timeseries data we need to create forecast requires timestamps, an item id and a demand. These features will map to Invoice Date.

We can see multiple orders for each day, You wont to create a forecast that predicts demand at a daily level. We must down sample the data from the individuals orders into a daily total.

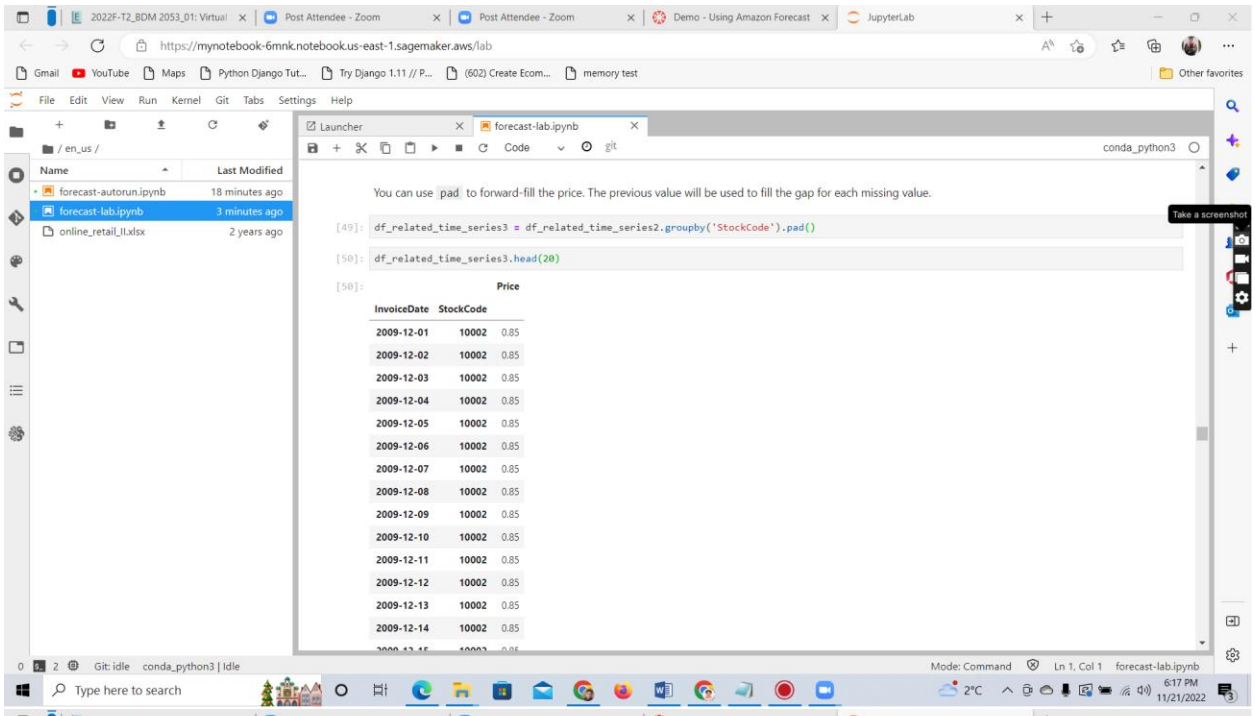

https://mynotebook-6mnk.notebook.us-east-1.sagemaker.aws/lab
forecast-lab.ipynb
conda_python3
df_time_series = df_time_series.groupby('StockCode').resample('D').sum().reset_index()
df_time_series['InvoiceDate'] = pd.to_datetime(df_time_series.InvoiceDate)
df_time_series = df_time_series.set_index('InvoiceDate')
df_time_series.head()
StockCode Quantity
InvoiceDate
2009-12-01 10002 12
2009-12-02 10002 0
2009-12-03 10002 7
2009-12-04 10002 25
2009-12-05 10002 0
df_time_series = df_time_series.groupby('StockCode').resample('D').sum().reset_index().set_index(['InvoiceDate'])
Examine the new DataFrame.
df_time_series[df_time_series.StockCode==21232]
StockCode Quantity
InvoiceDate
2009-12-01 21232 171
2009-12-02 21232 164
2009-12-03 21232 192


https://mynotebook-6mnk.notebook.us-east-1.sagemaker.aws/lab
forecast-lab.ipynb
conda_python3
Repeat this process with the related time series data.
df_related_time_series.head()
StockCode Price
InvoiceDate
2009-12-01 07:45:00 85048 6.95
2009-12-01 07:45:00 79323P 6.75
2009-12-01 07:45:00 79323W 6.75
2009-12-01 07:45:00 22041 2.10
2009-12-01 07:45:00 21232 1.25
df_related_time_series2 = df_related_time_series.groupby('StockCode').resample('D').mean().reset_index().set_index(['InvoiceDate','StockCod
df_related_time_series2.head(20)
Price
InvoiceDate StockCode
2009-12-01 10002 0.85
2009-12-02 10002 NaN
2009-12-03 10002 0.85
2009-12-04 10002 0.85
2009-12-05 10002 NaN





Here we fill all the product that has no orders for those days, and thus it has no price. We use pad to forward fill the price. The previous value will be used to fill the gap for each missing value.

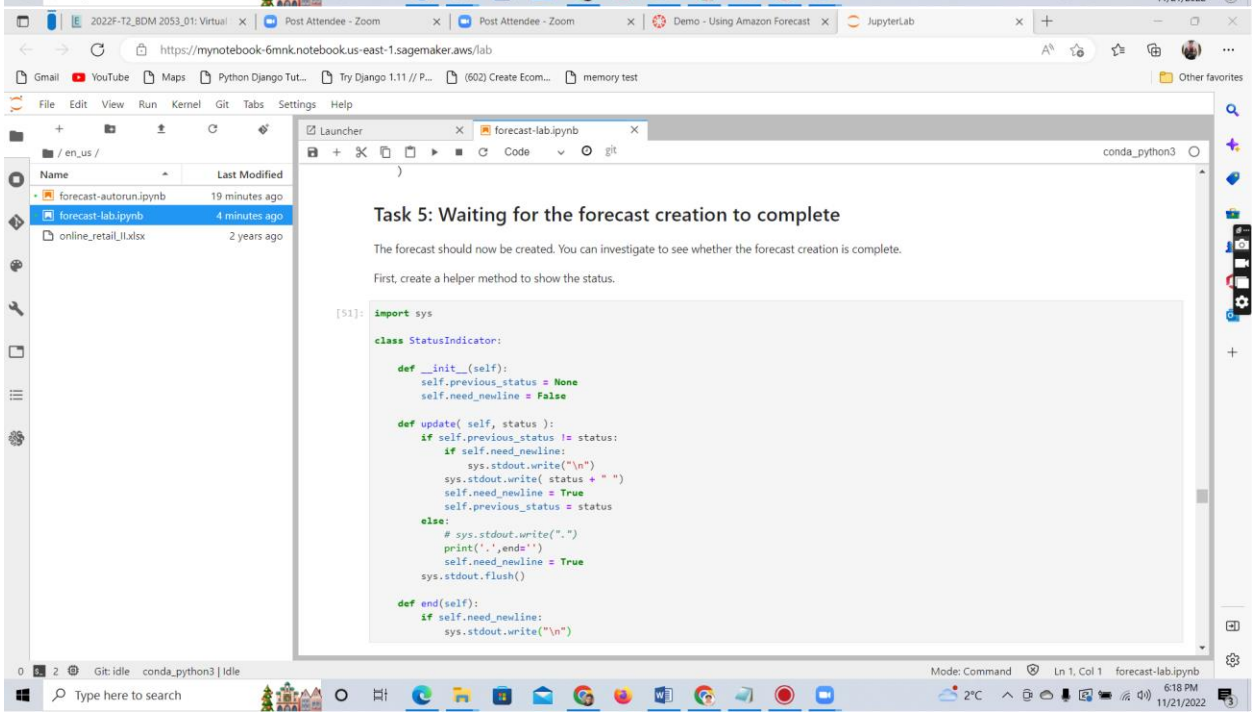


You can use `pad` to forward-fill the price. The previous value will be used to fill the gap for each missing value.

```
[49]: df_related_time_series3 = df_related_time_series2.groupby('StockCode').pad()
[50]: df_related_time_series3.head(20)
```

[50]:

InvoiceDate	StockCode	Price
2009-12-01	10002	0.85
2009-12-02	10002	0.85
2009-12-03	10002	0.85
2009-12-04	10002	0.85
2009-12-05	10002	0.85
2009-12-06	10002	0.85
2009-12-07	10002	0.85
2009-12-08	10002	0.85
2009-12-09	10002	0.85
2009-12-10	10002	0.85
2009-12-11	10002	0.85
2009-12-12	10002	0.85
2009-12-13	10002	0.85
2009-12-14	10002	0.85



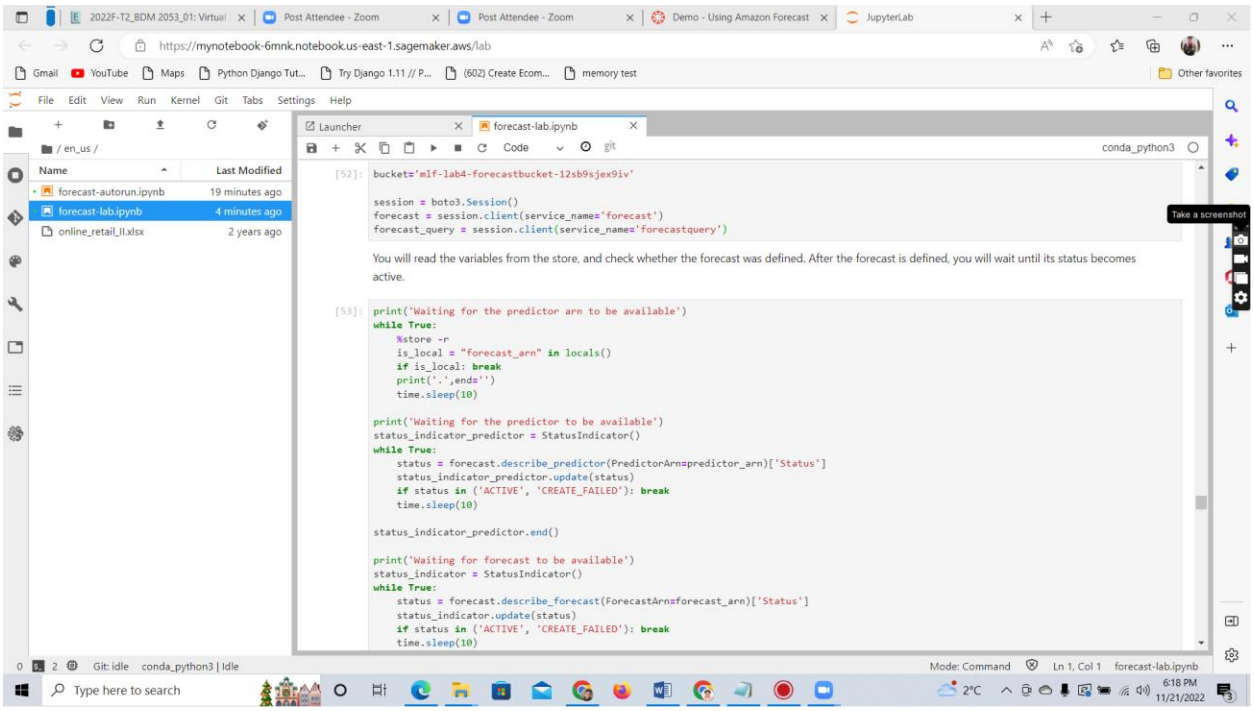
Task 5: Waiting for the forecast creation to complete

The forecast should now be created. You can investigate to see whether the forecast creation is complete.

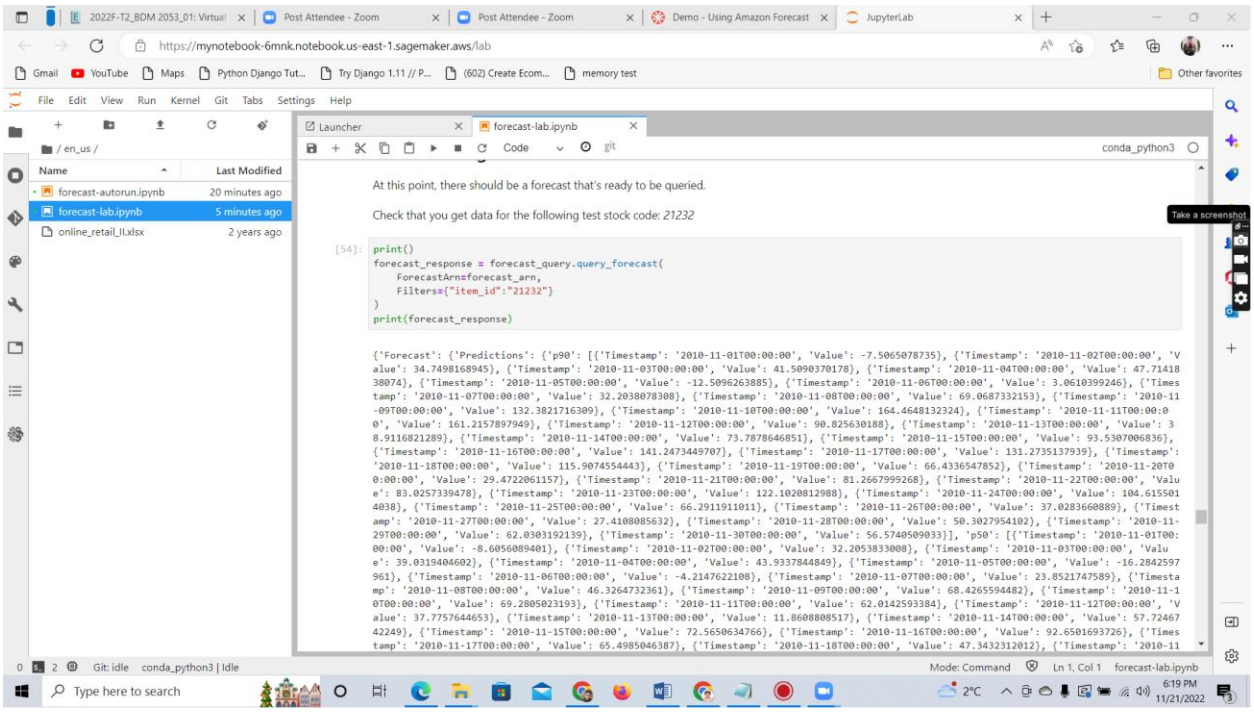
First, create a helper method to show the status.

```
[51]: import sys

class StatusIndicator:

    def __init__(self):
        self.previous_status = None
        self.need_newline = False

    def update( self, status ):
        if self.previous_status != status:
            if self.need_newline:
                sys.stdout.write("\n")
            sys.stdout.write( status + " ")
            self.need_newline = True
            self.previous_status = status
        else:
            # sys.stdout.write(".")
            print('.',end='')
            self.need_newline = True
        sys.stdout.flush()

    def end(self):
        if self.need_newline:
            sys.stdout.write("\n")
```

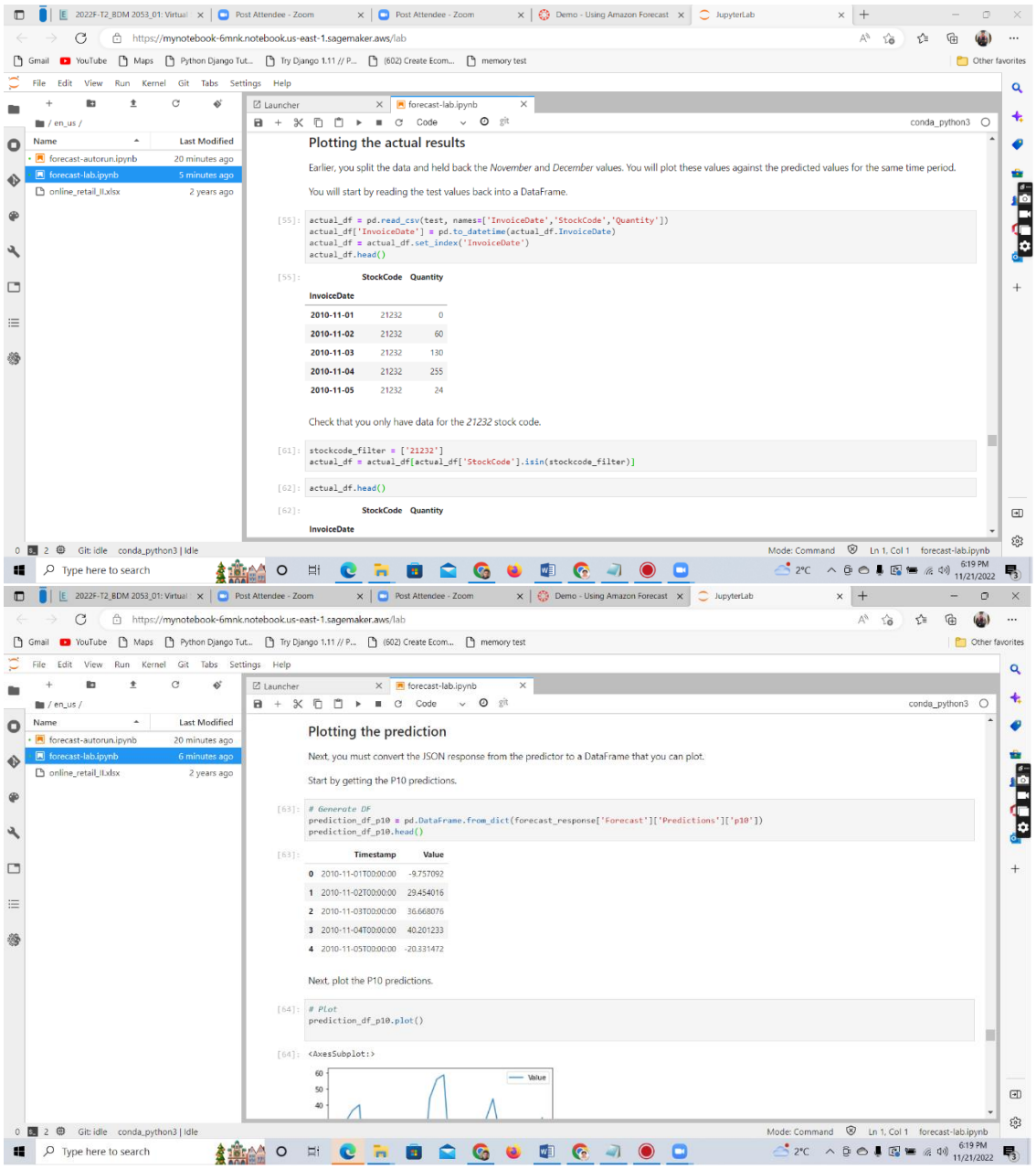
This is the code that for the forecast creation to complete. First we create helper method to show the status.



We will read the variables form the store and check whether the forecast was defined. After the forecast is defined, we will wait util its status becomes active.

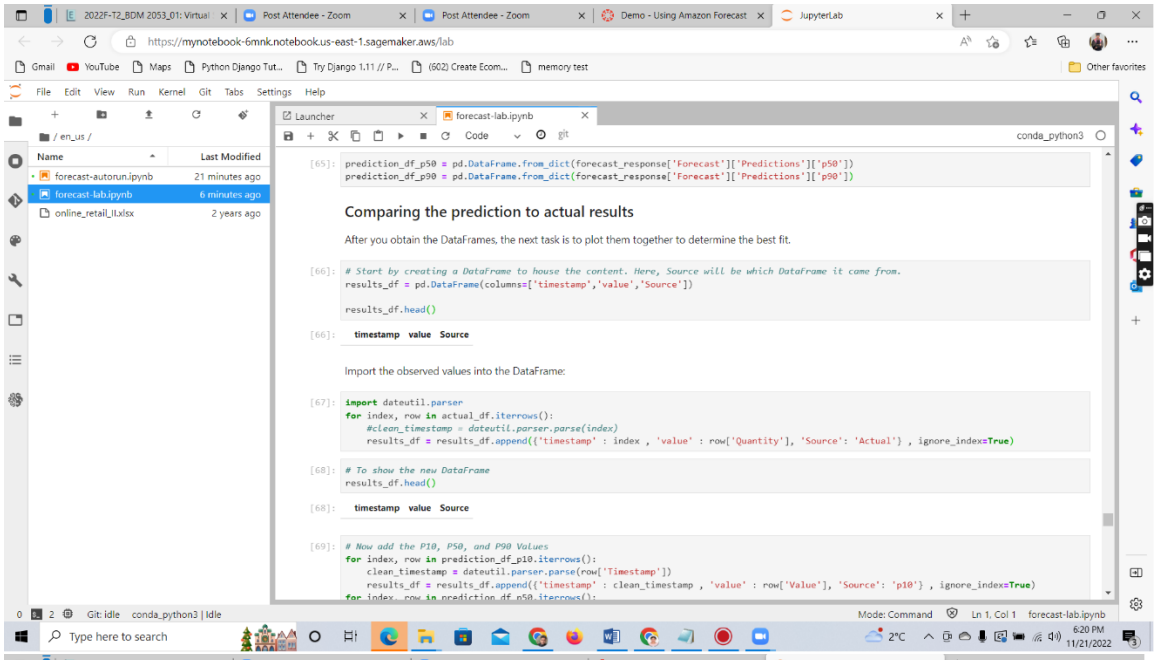


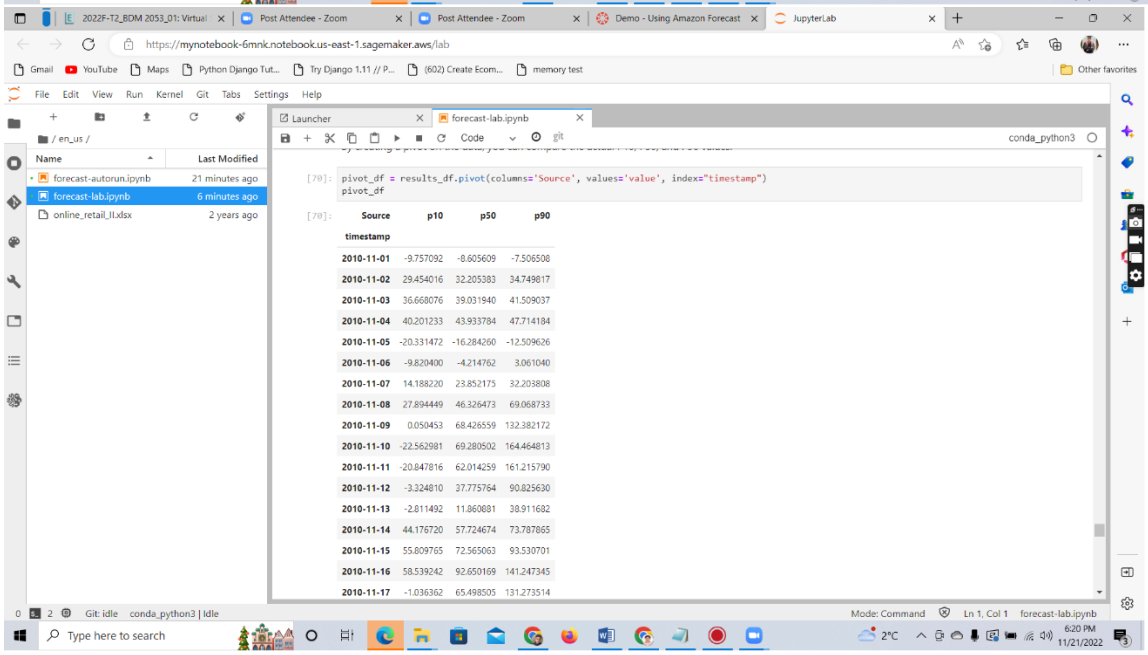
Here, we have a forecast that's ready to be queried. As you can see the query forecast.



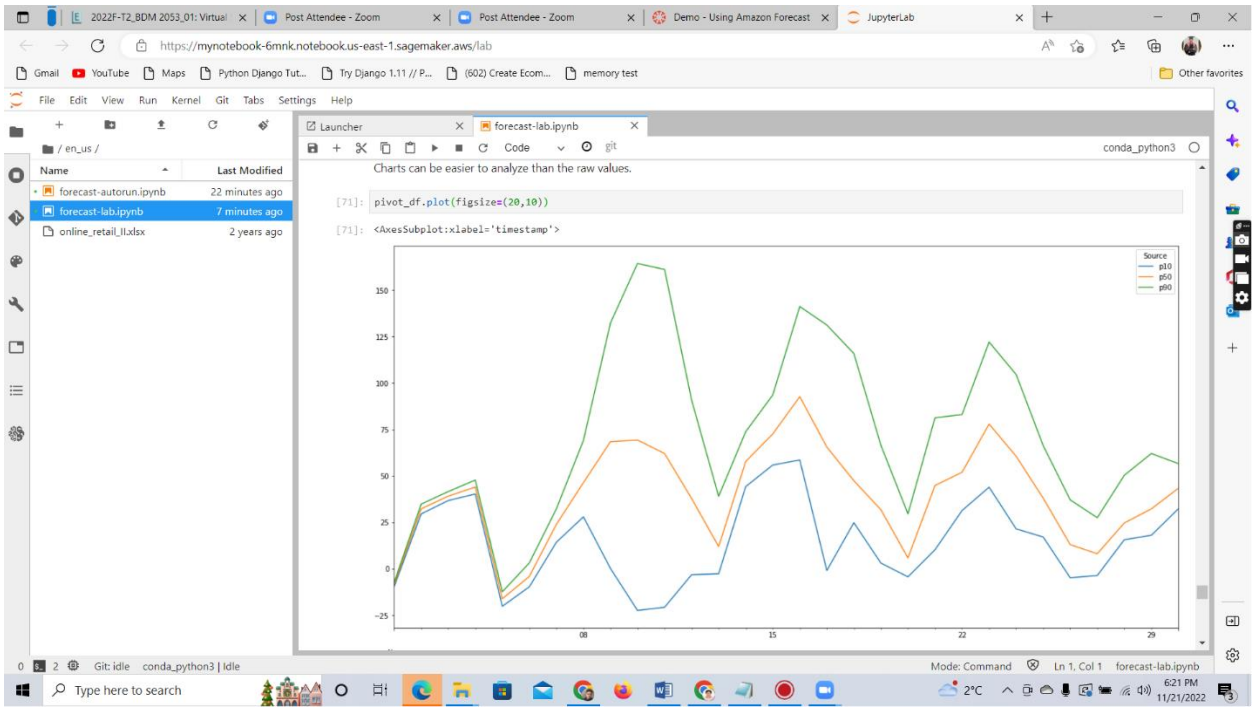
When we plotting the result, we split the data and held back the Nov and Dec values. We will plot these values against the predicted values for the same time.

Here, we are plotting the prediction, we covert the JSON response of forecast from the predictor to a Data Frame that we can plot.





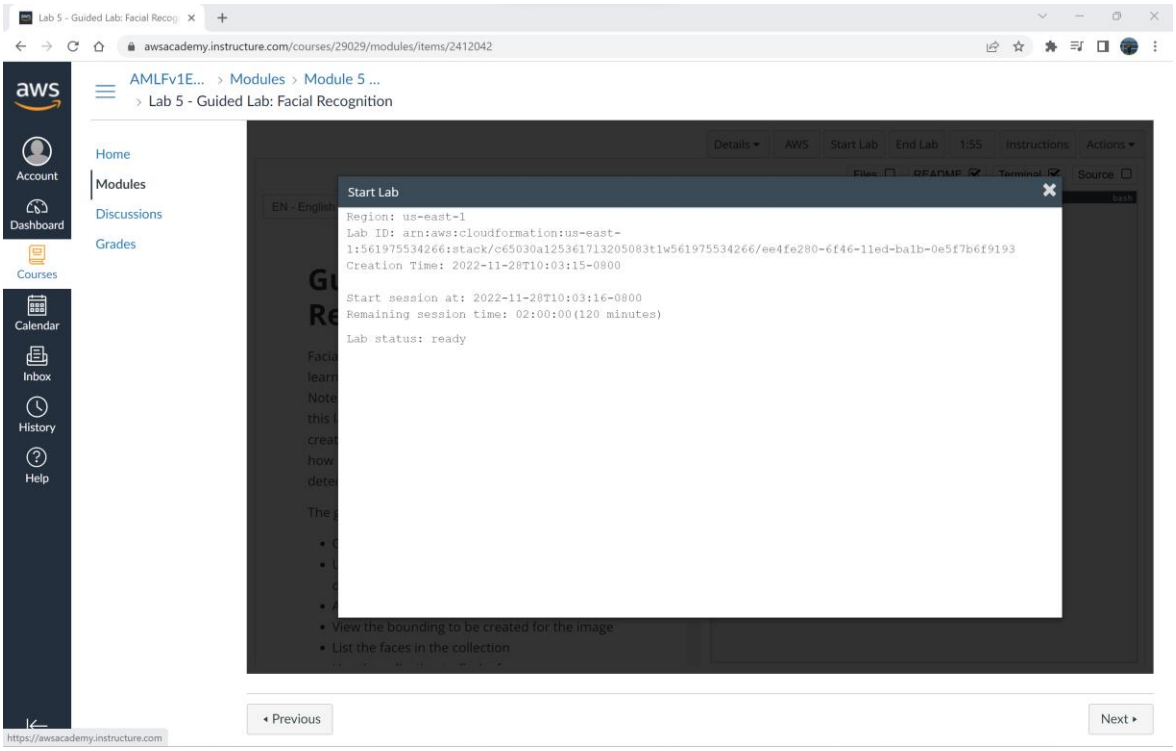
First we are comparing the prediction to actual results. We split into p10, p50, p90. As you can see the pivot df for the result with timestamp and values.

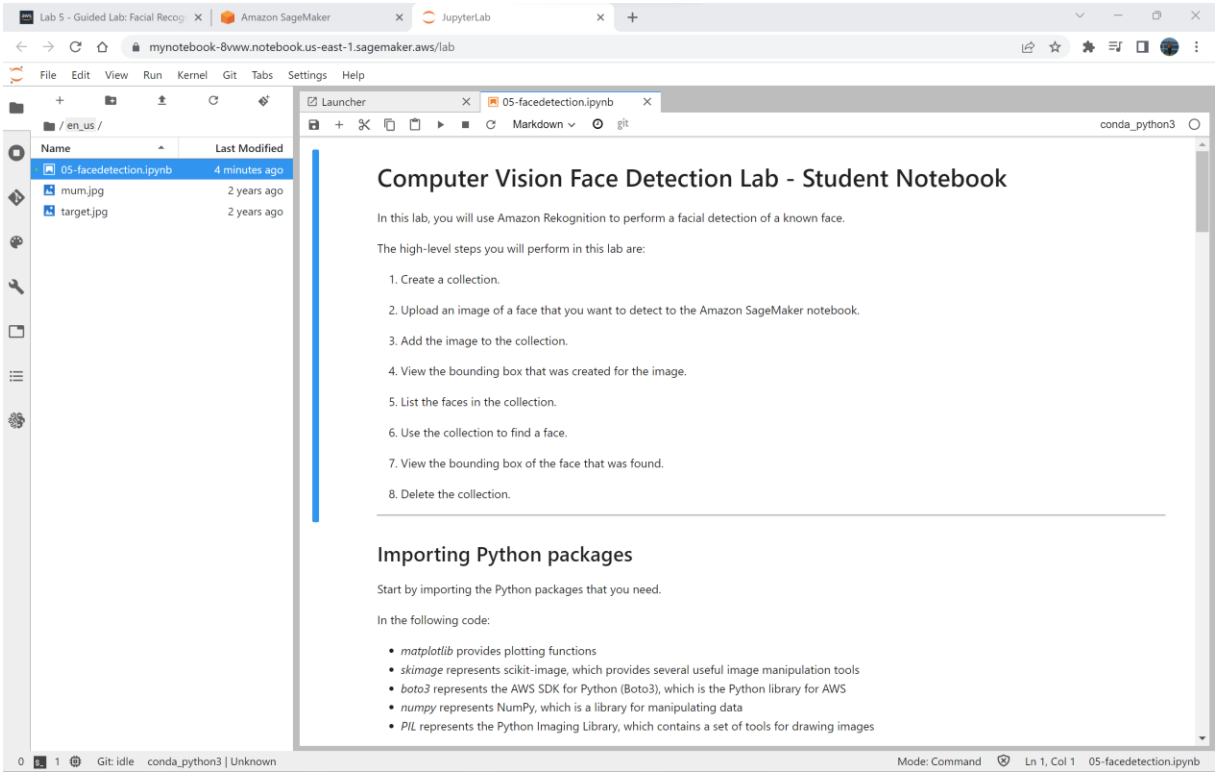


This graph represents, the three source p10, p50, p90.

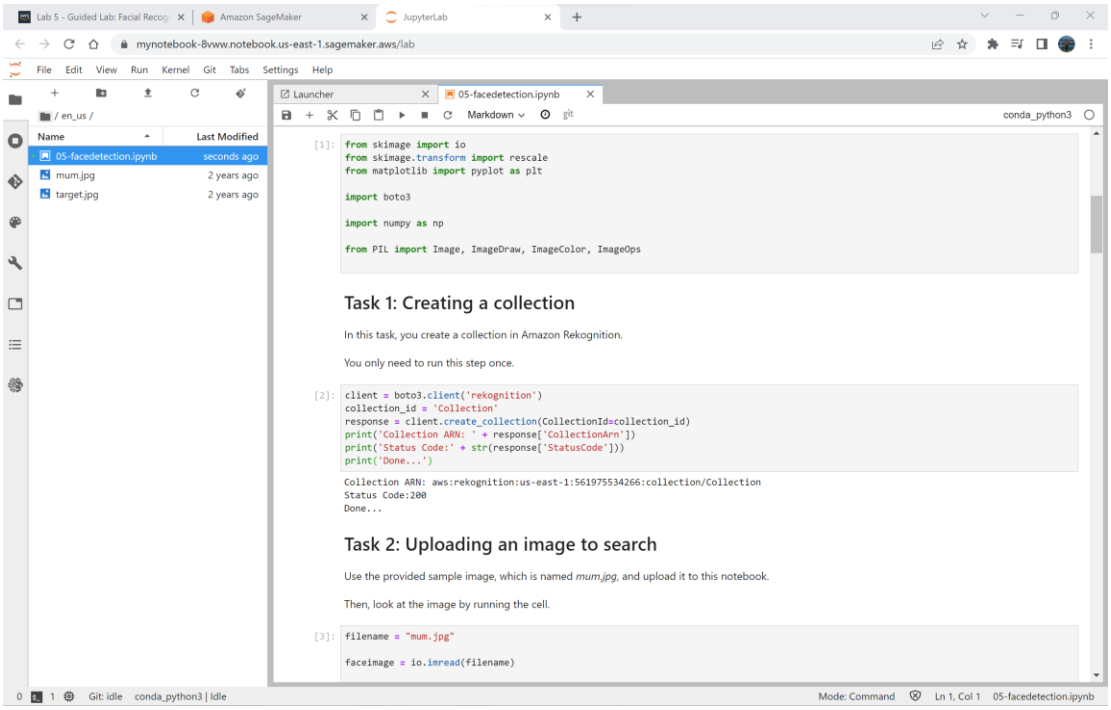
After all we mange complete our task for this lab. We completed lab 4 successfully for time series data using amazon sage maker with forecasting model.

Lab 5 Guided Lab: Facial Recognition



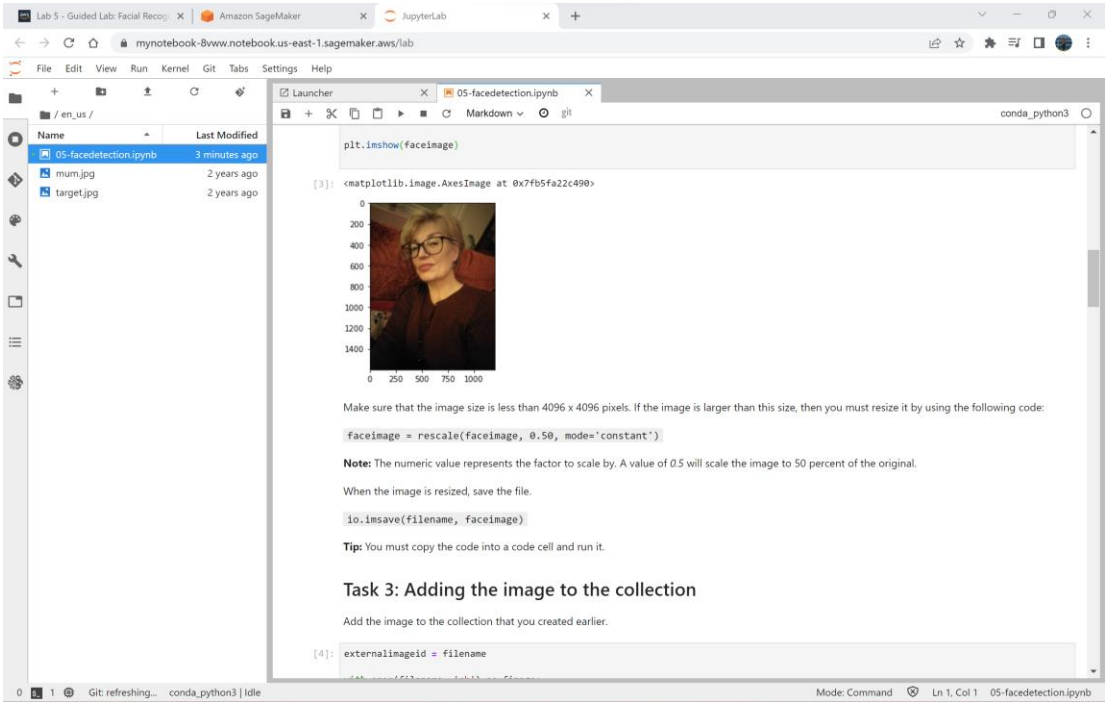


First for this lab, we load the file from Jupiter Notebook called face detection. Then we start to analyse the process.

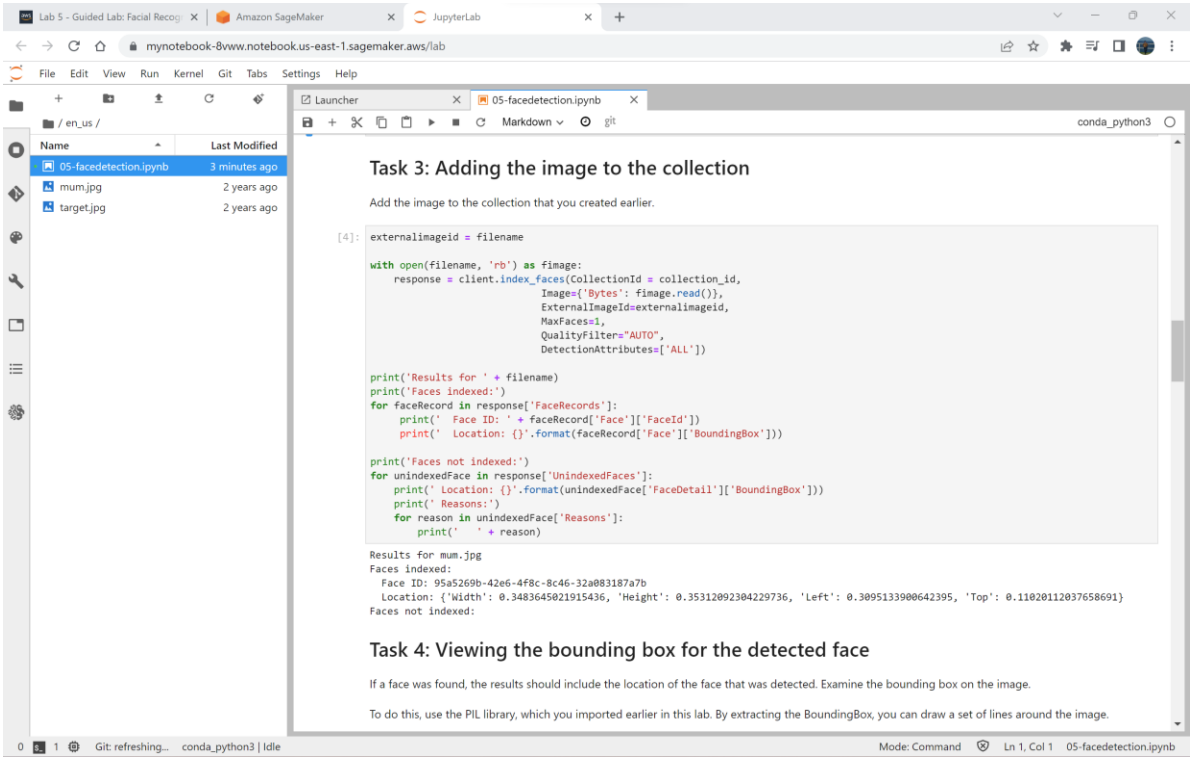


First of all we import all the packages that are necessary for this analysis. Such as, skimage, rescale, matplotlib PTL and so on. For the TASK1: we create the collection to store the raw data named as client.

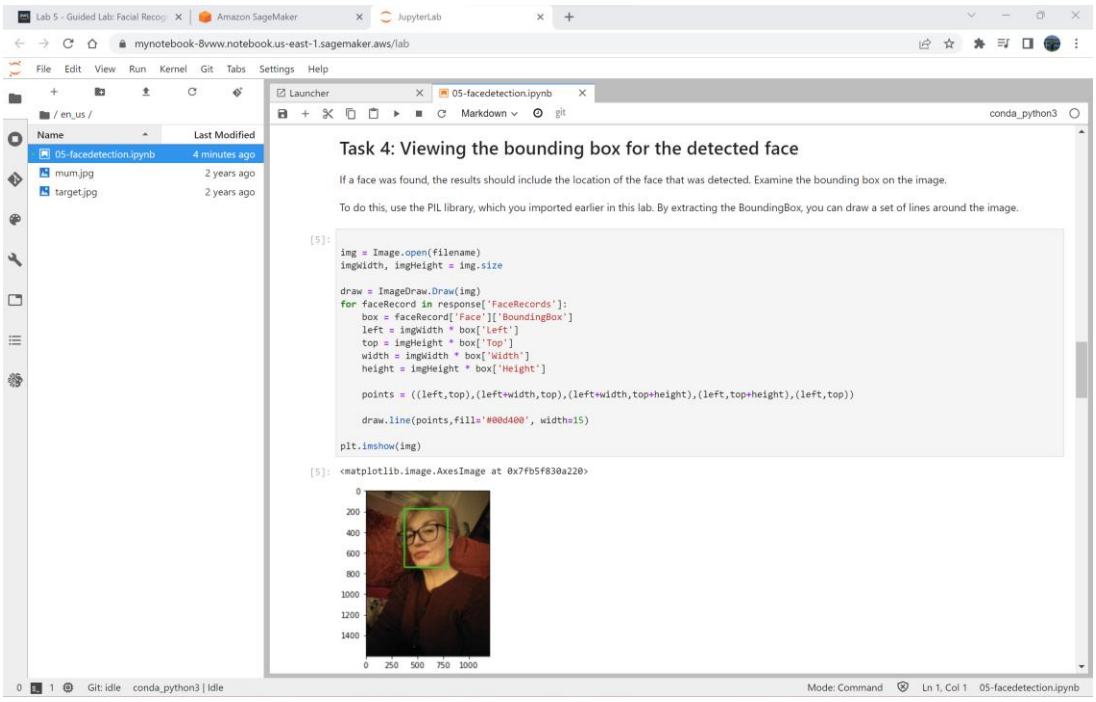
Then For task 2, we use the provided sample image, which is named mum.jpg and upload an image to search.



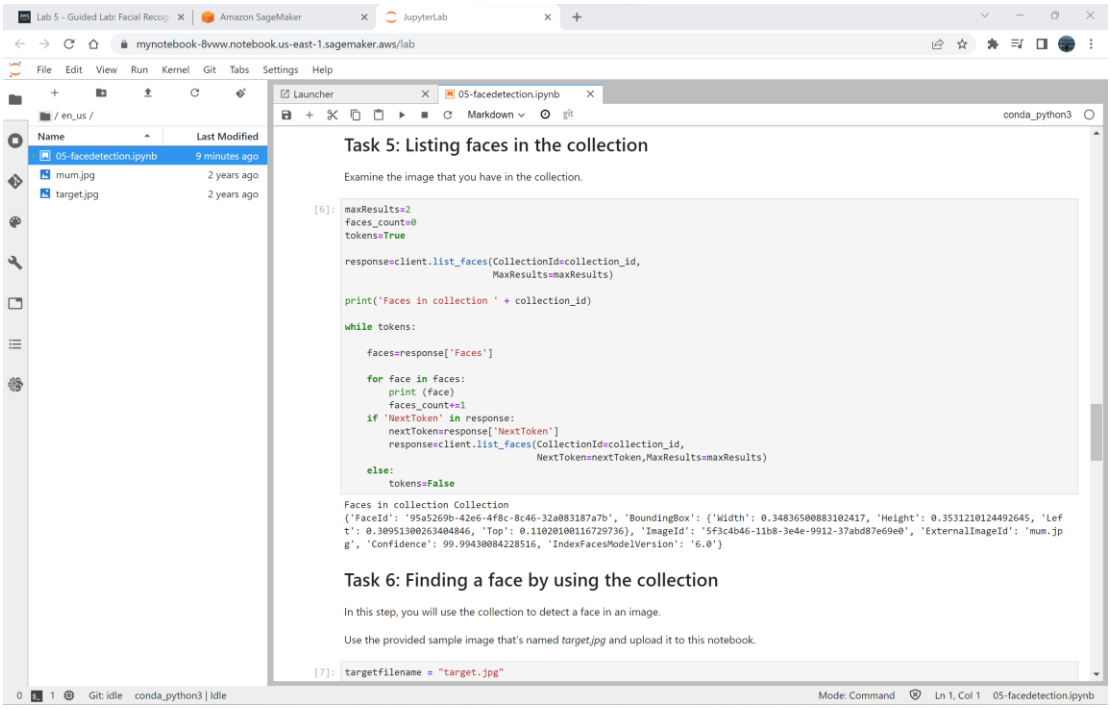
Here you can see the image that we have uploaded. There is some numeric representation which is known as scale factor. A value of 0.5 will scale the image to 50 percent of the original.



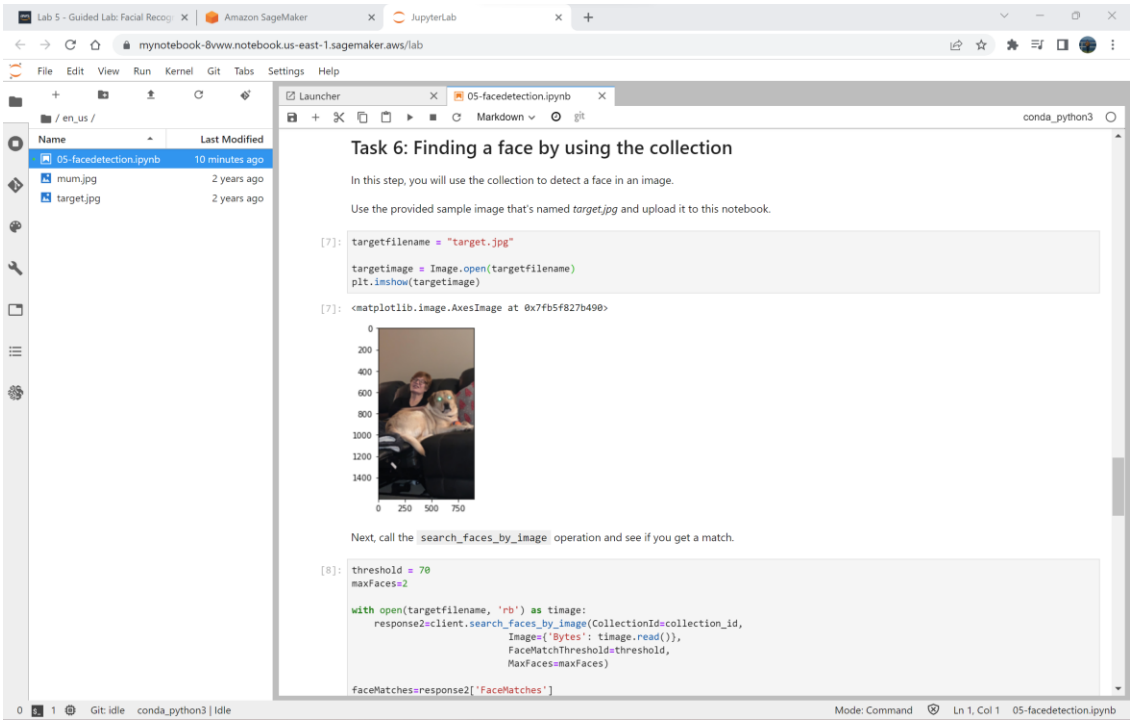
For task 3: we added the images to the collection that we have created earlier. The result give use the face id, and location as label of image.



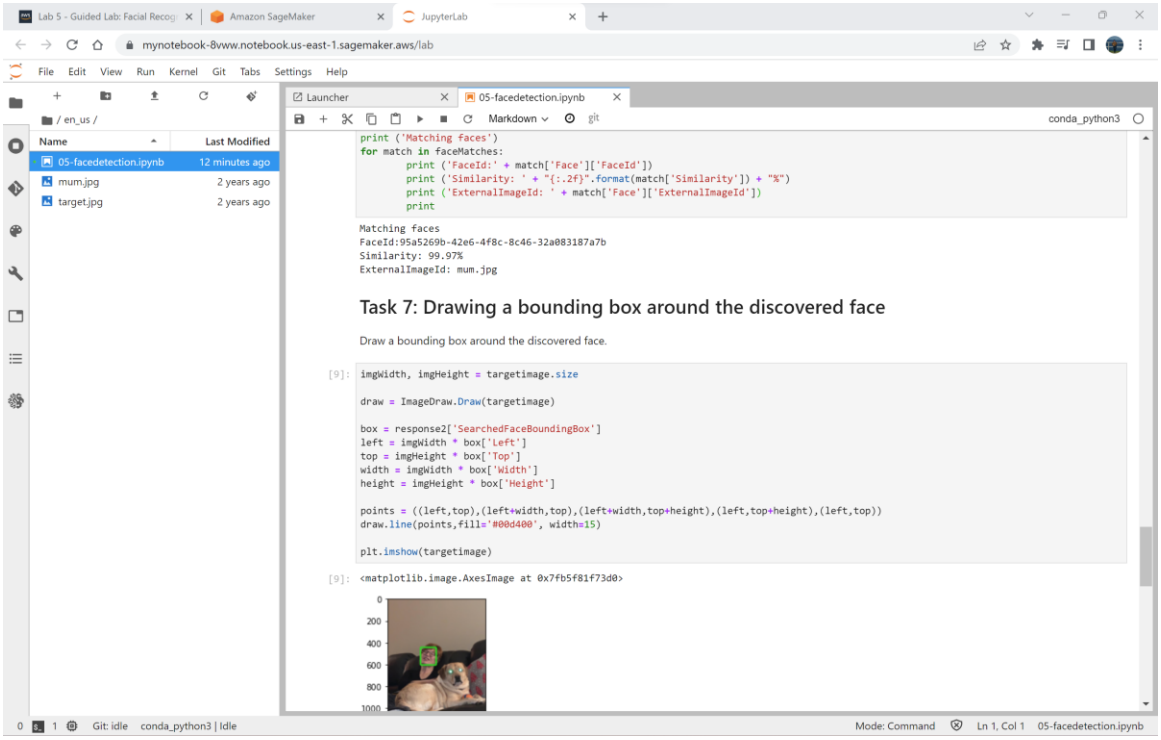
For Task 4, here we are detecting face inside the bounding box. If the face was found, the result should include the location of the face that was detected. We use PIL library, and extract the bounding box. In following image, we can see the face has been bounded by box with certain width and height.



Task 5 is about the listing the faces which are inside the collection. This collection contains FaceId, Bounding Box, ImageId and confidence score.

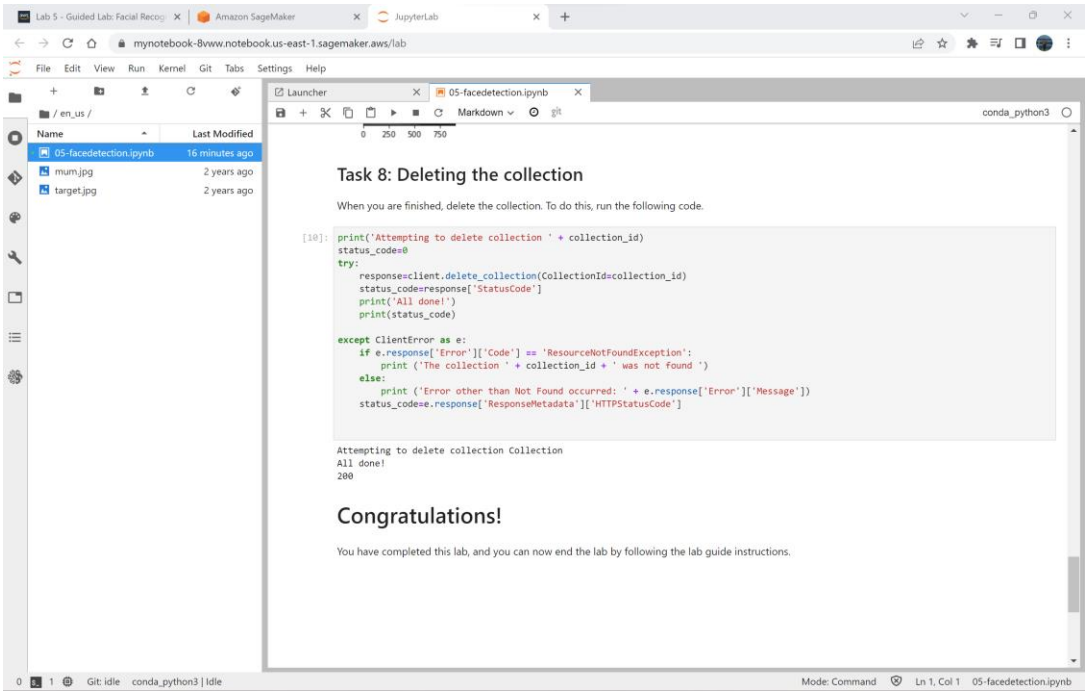


Task 6 : we try to find the face by using our collection in that particular image. We use the image called targe.jpg . Then we use search_faces_by_image to operate and see if we get a match.



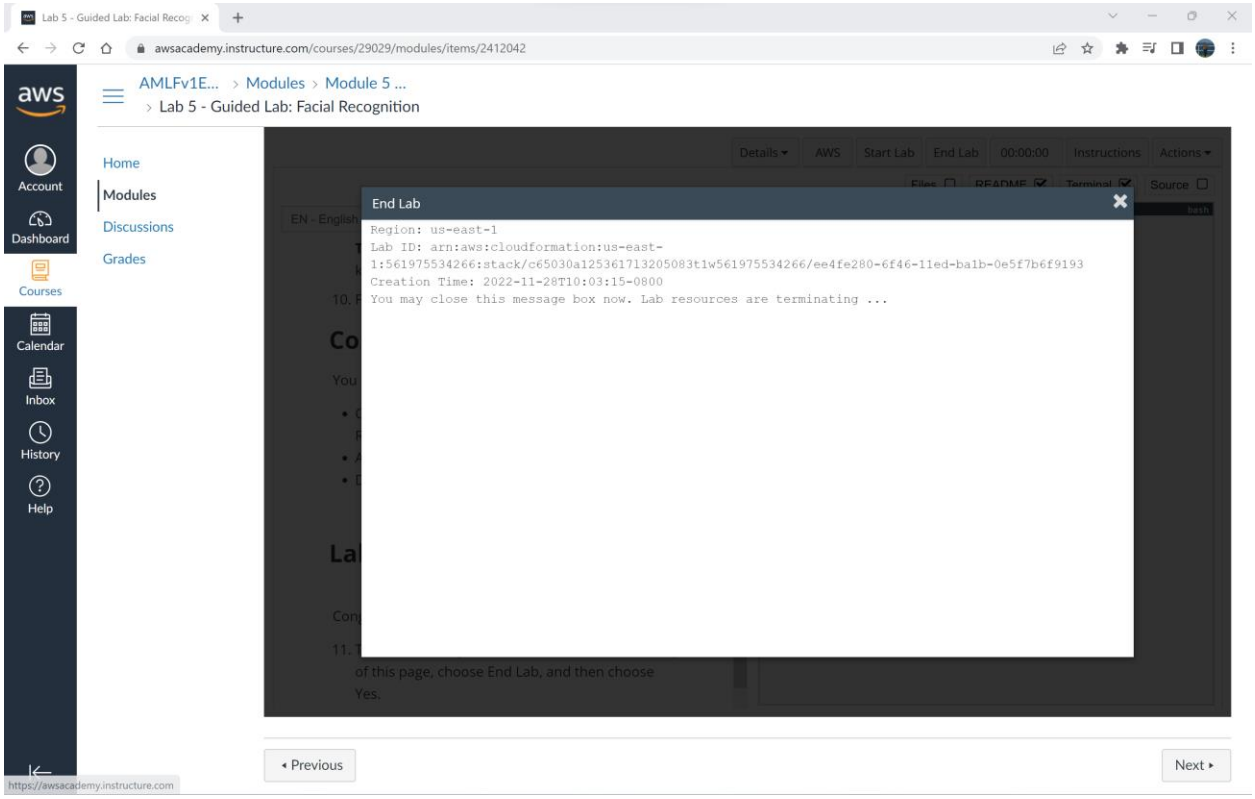
Here we can see the result. The face of new image has matching similarity of **99.97%** with mum.jpg

Task7: We will be drawing bounding box to discover face. The result can see on the image.

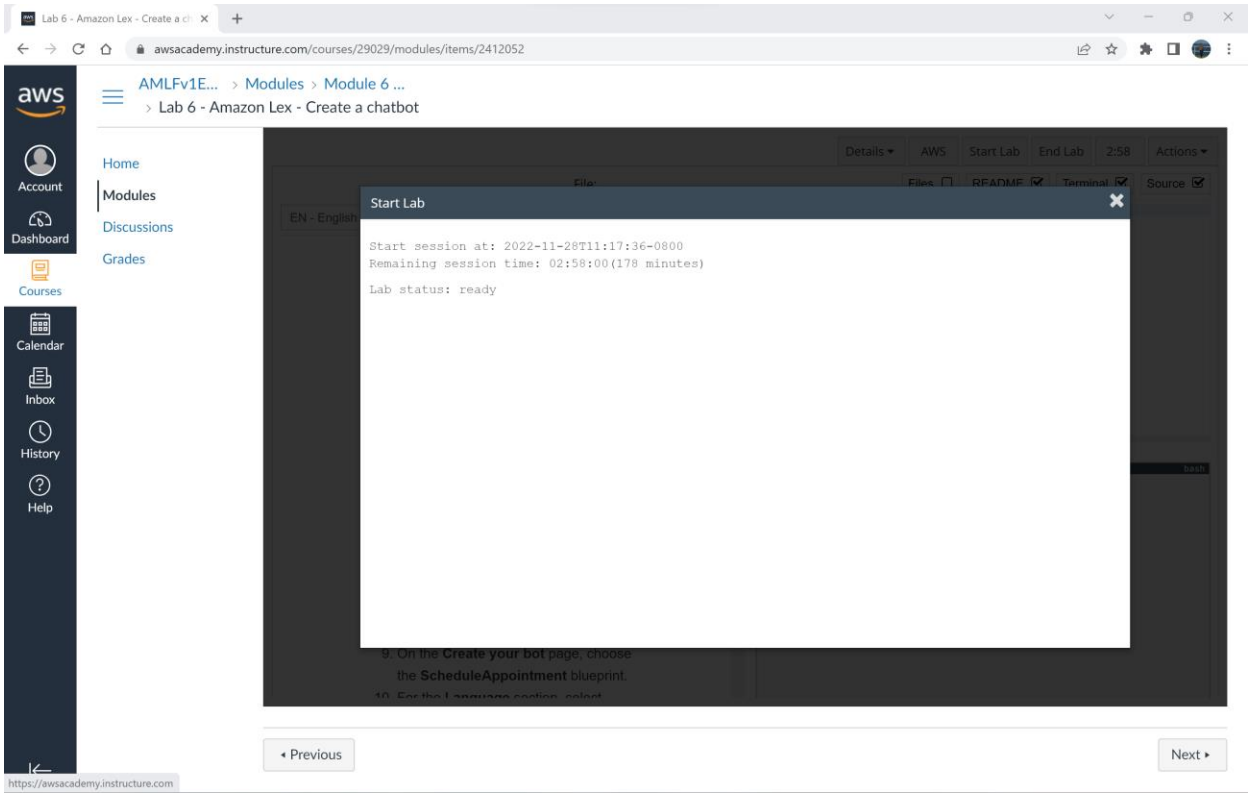


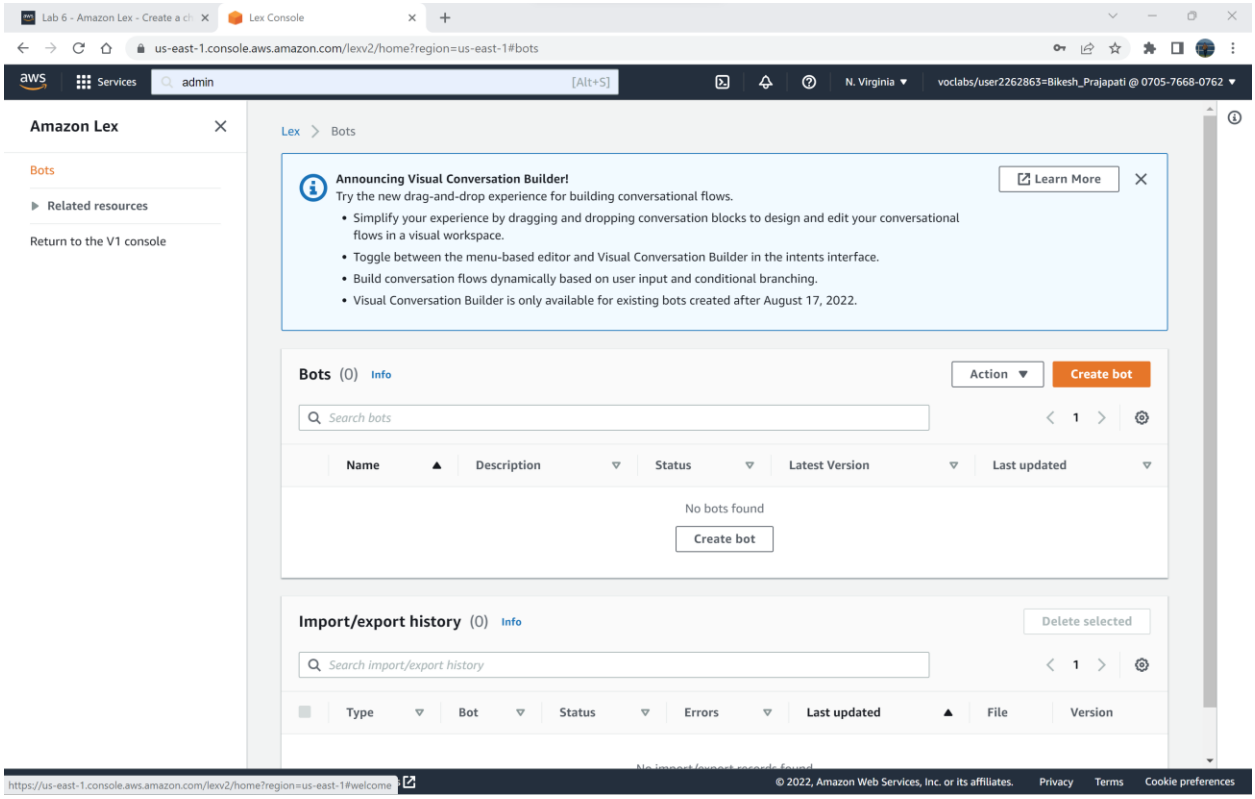
For Task 8: After we finish our detection , we delete our collection. And print out the success code which is 200.

That’s all for this lab. We complete all the required step for this lab to detect the facial recognition model and check the accuracy.



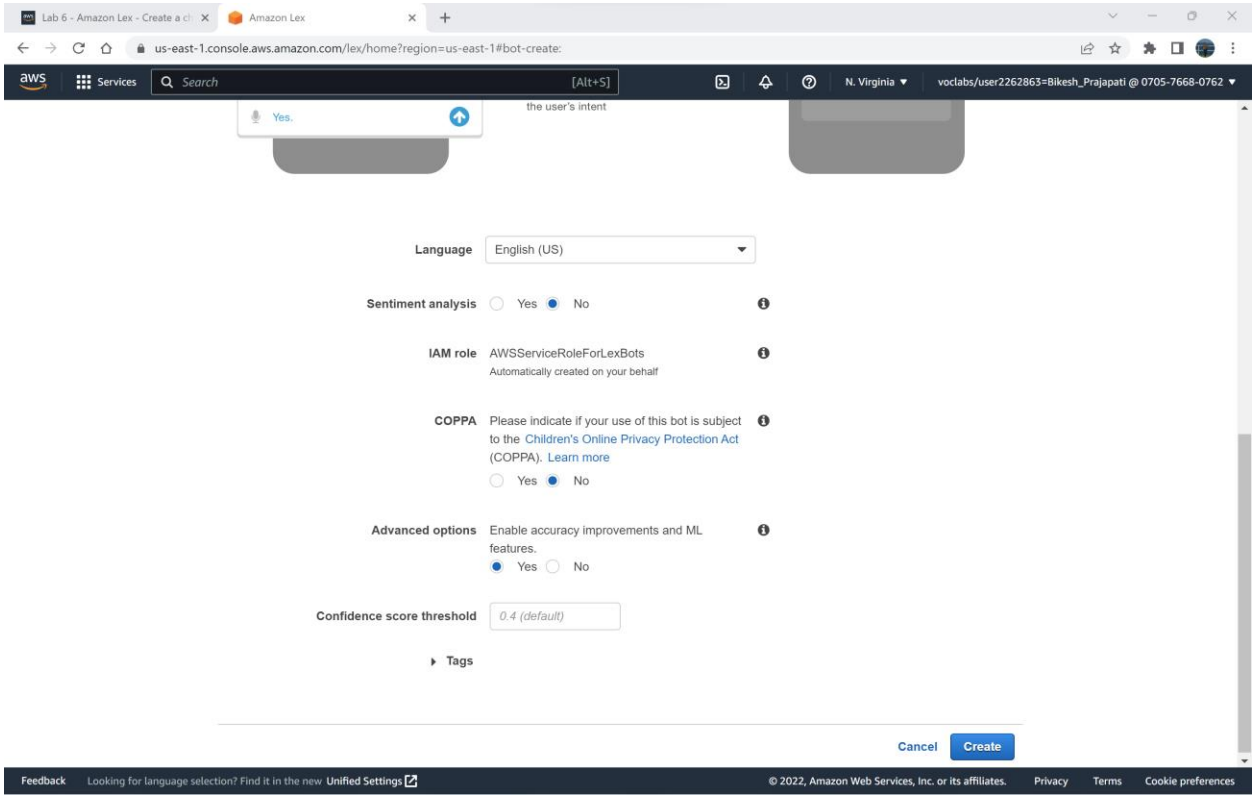
LAB 6 Guided Lab: Natural Language Processing – Create chatbot



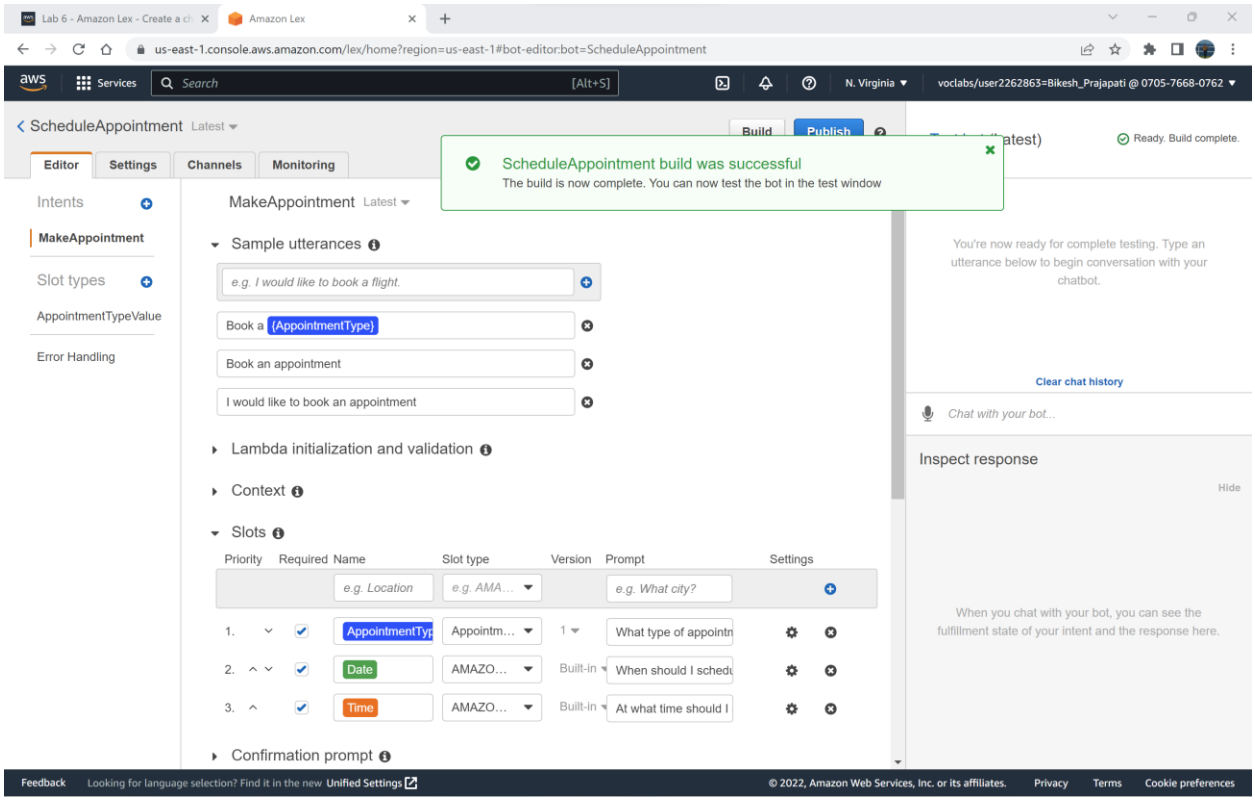


First we go to amazon services and then select the amazon lex for this lab. After that we click on return to the v1 console, where we create the bot for the system. Then click on create button.

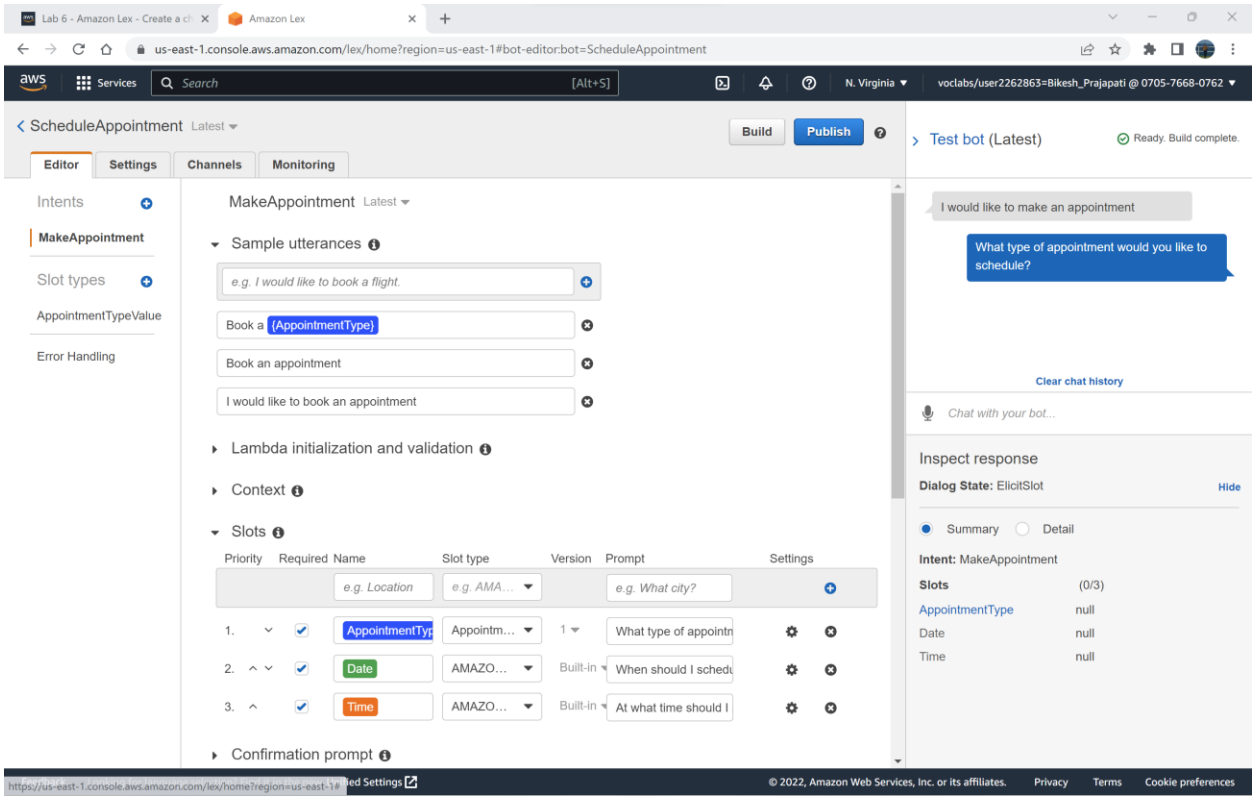
By following the instruction given in lab then we provides all the essential field require for this creation an then create a bot .

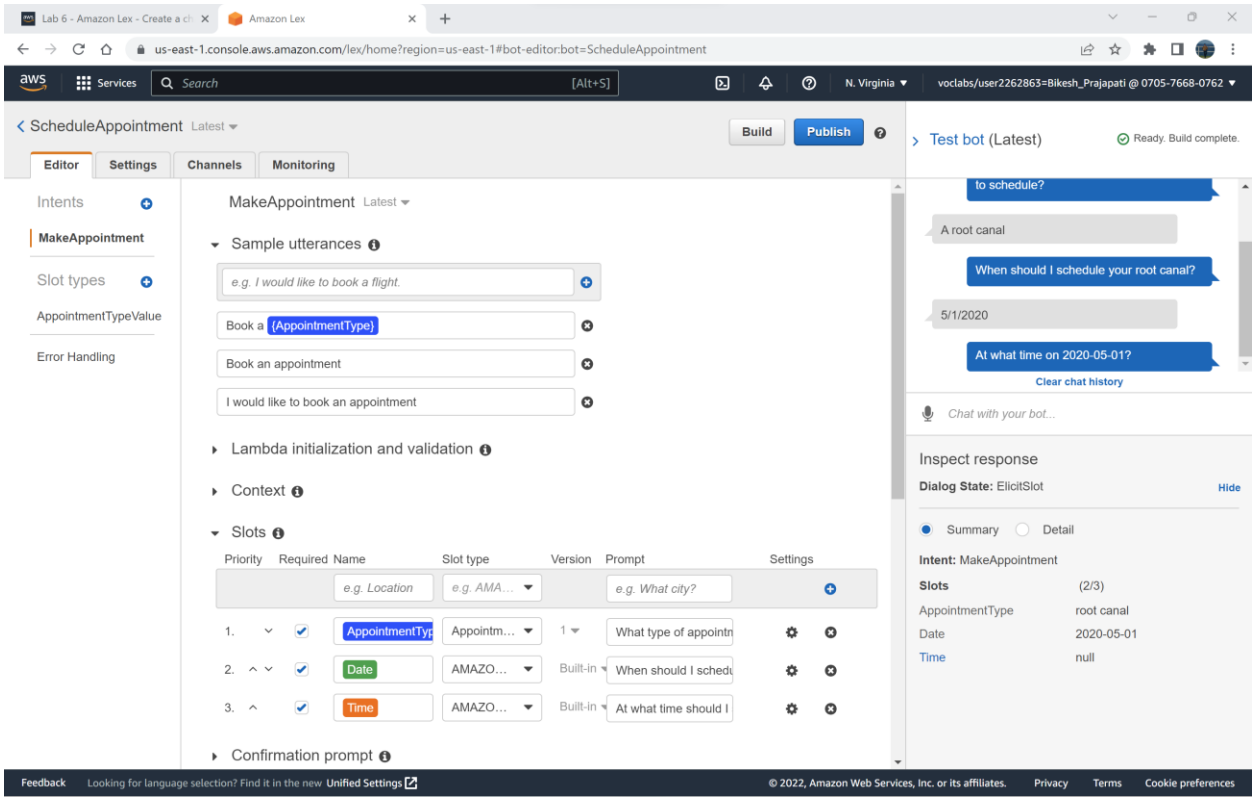


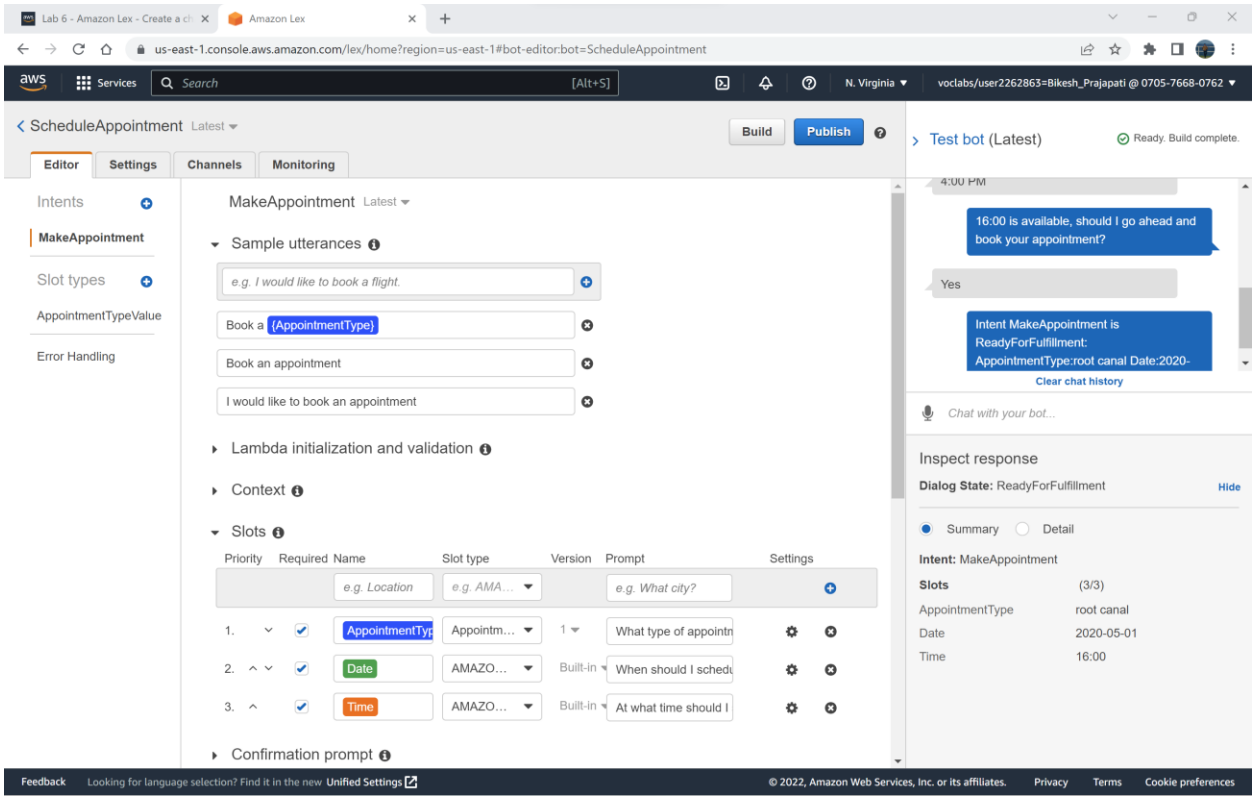
These are all the require selection for lab to create the bot.



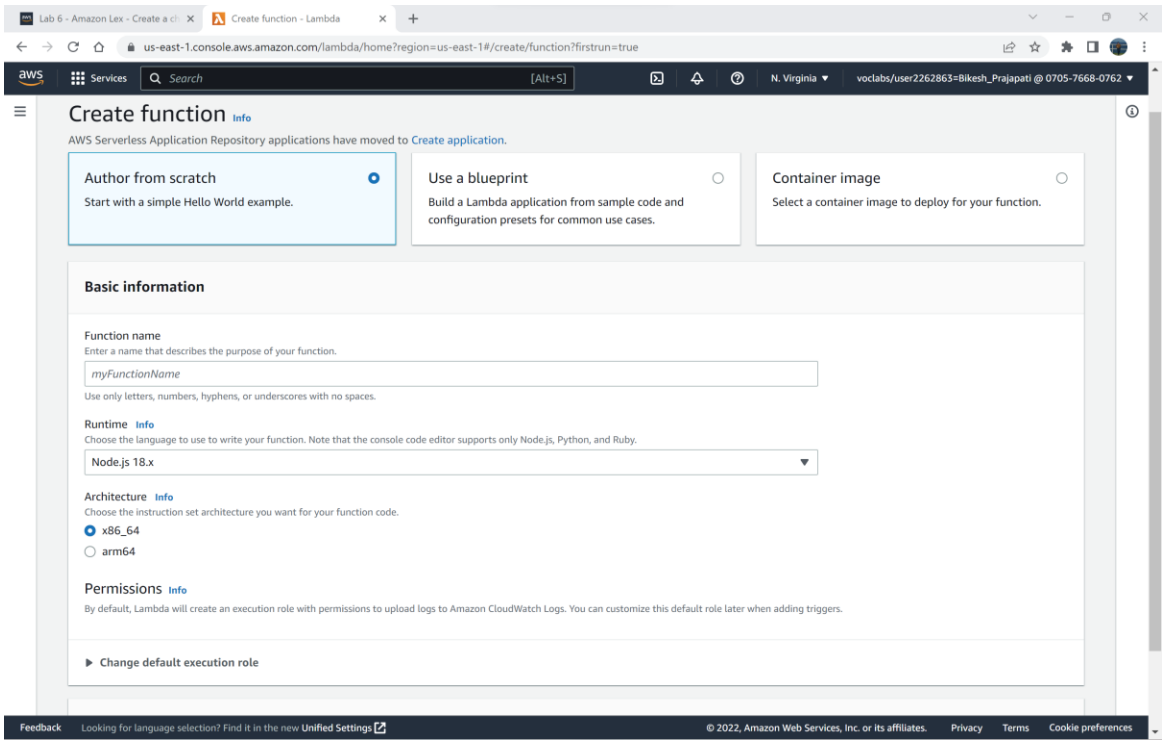
Here is the completion message that we created the schedule Appointment bot system in amazon lex. Now we can test the text that are present in lab exercise.



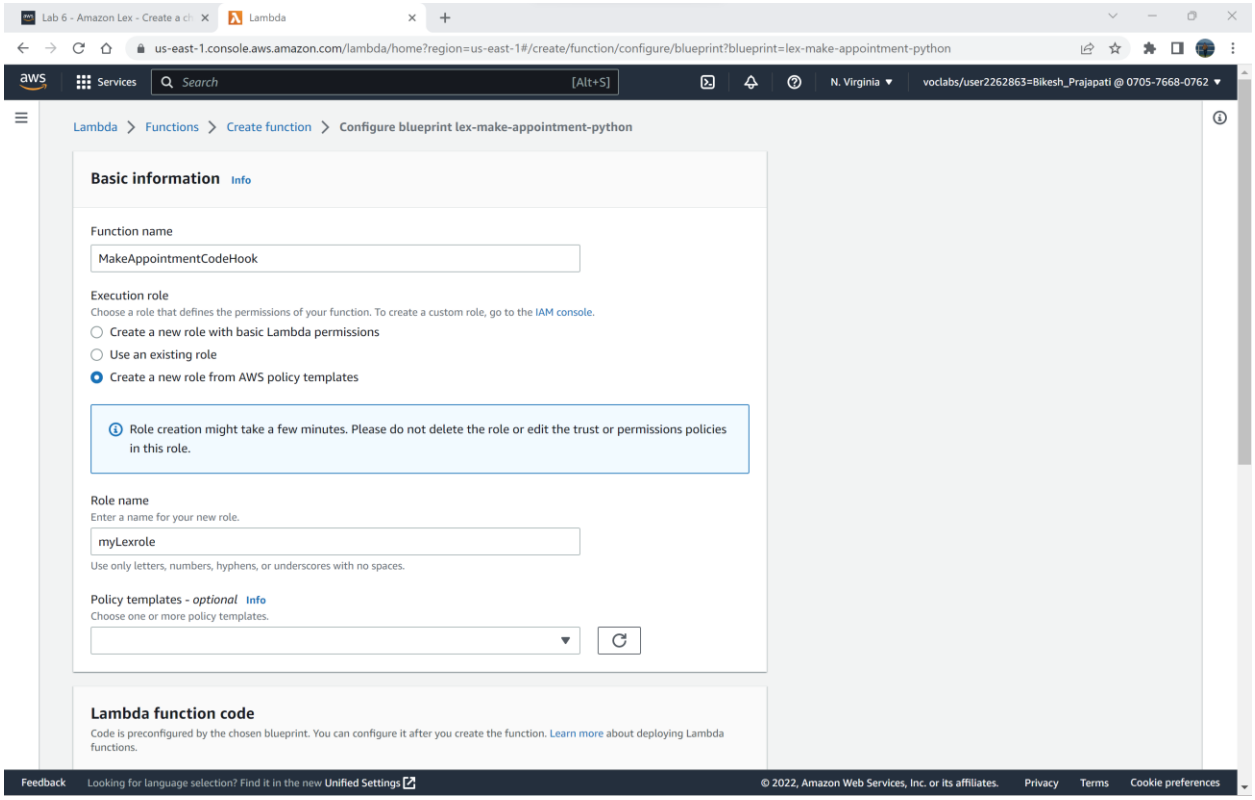


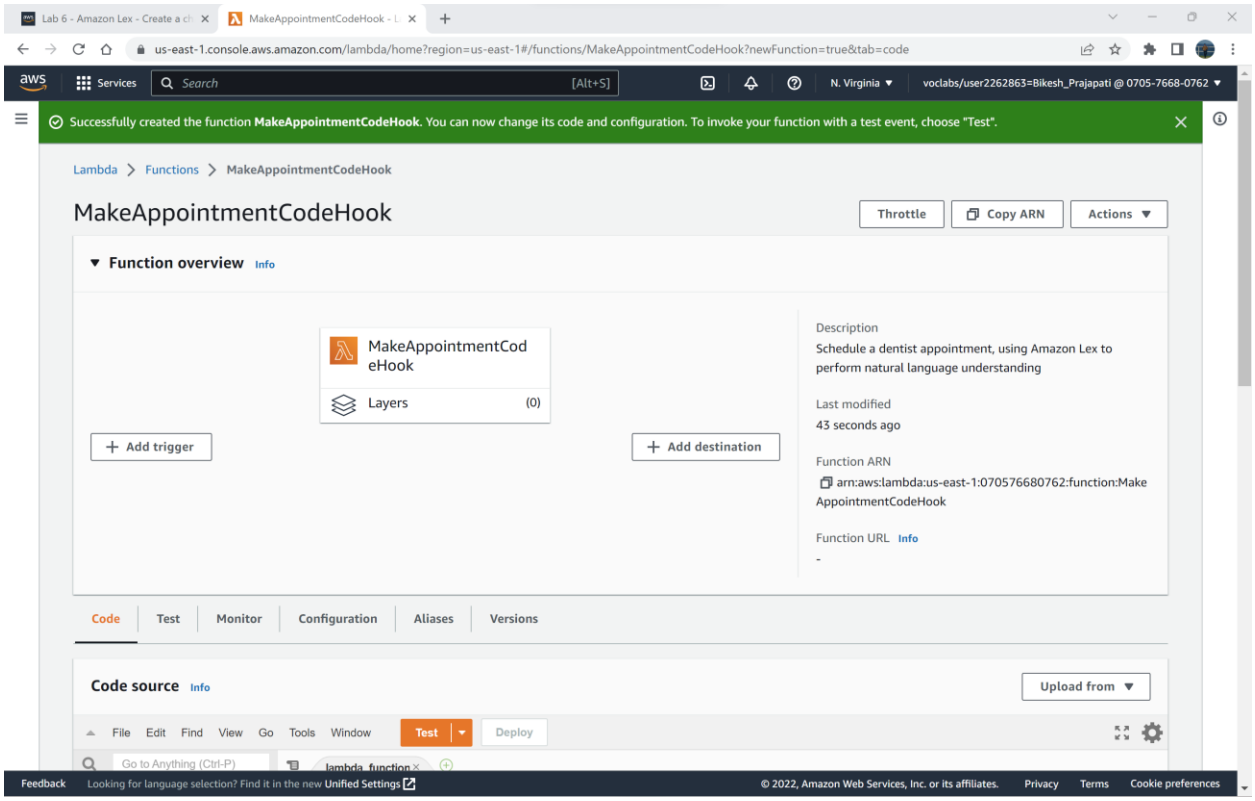


As you can see the above response from the chat box on left side that we have the following question about the schedule appointment, and it work very well. After that we will be creating lambda function.

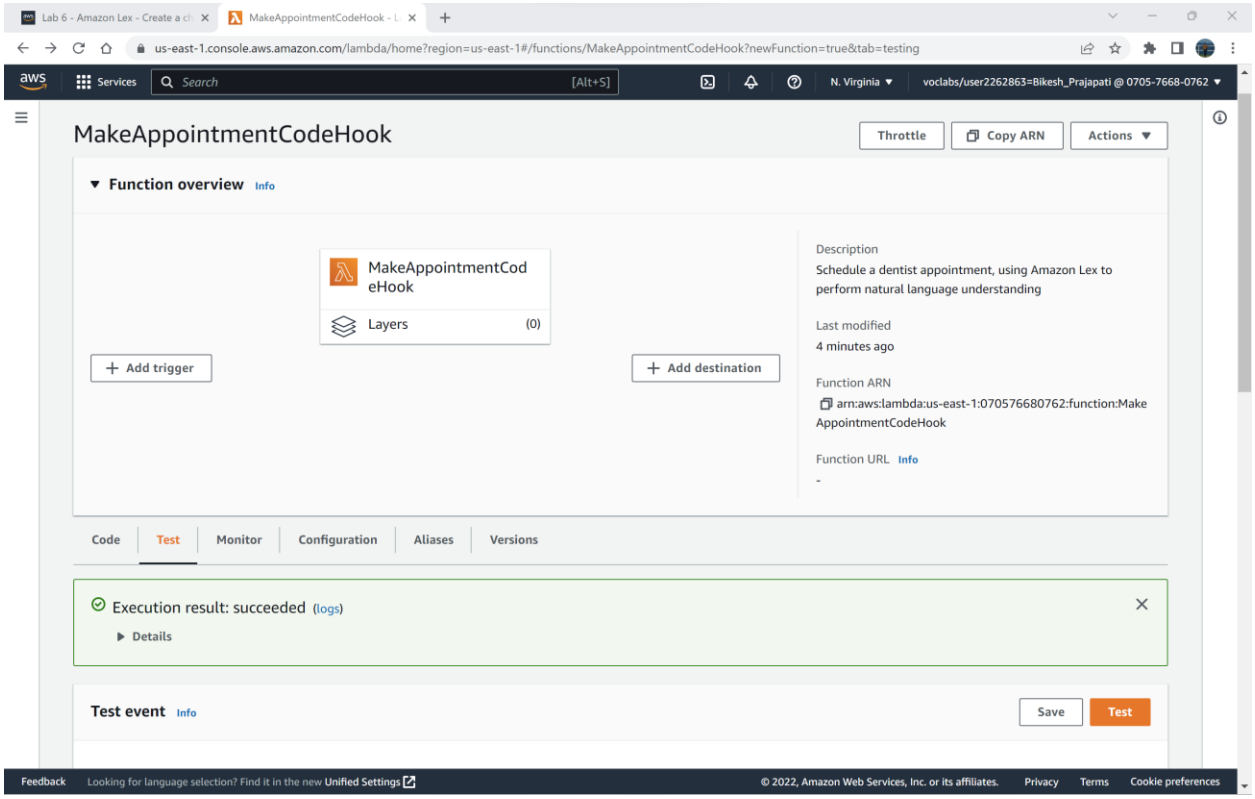


Now we create the lambda function. We go to amazon console then search lambda then click on lambda function. We fill out all the information that we need for this creation an click create.

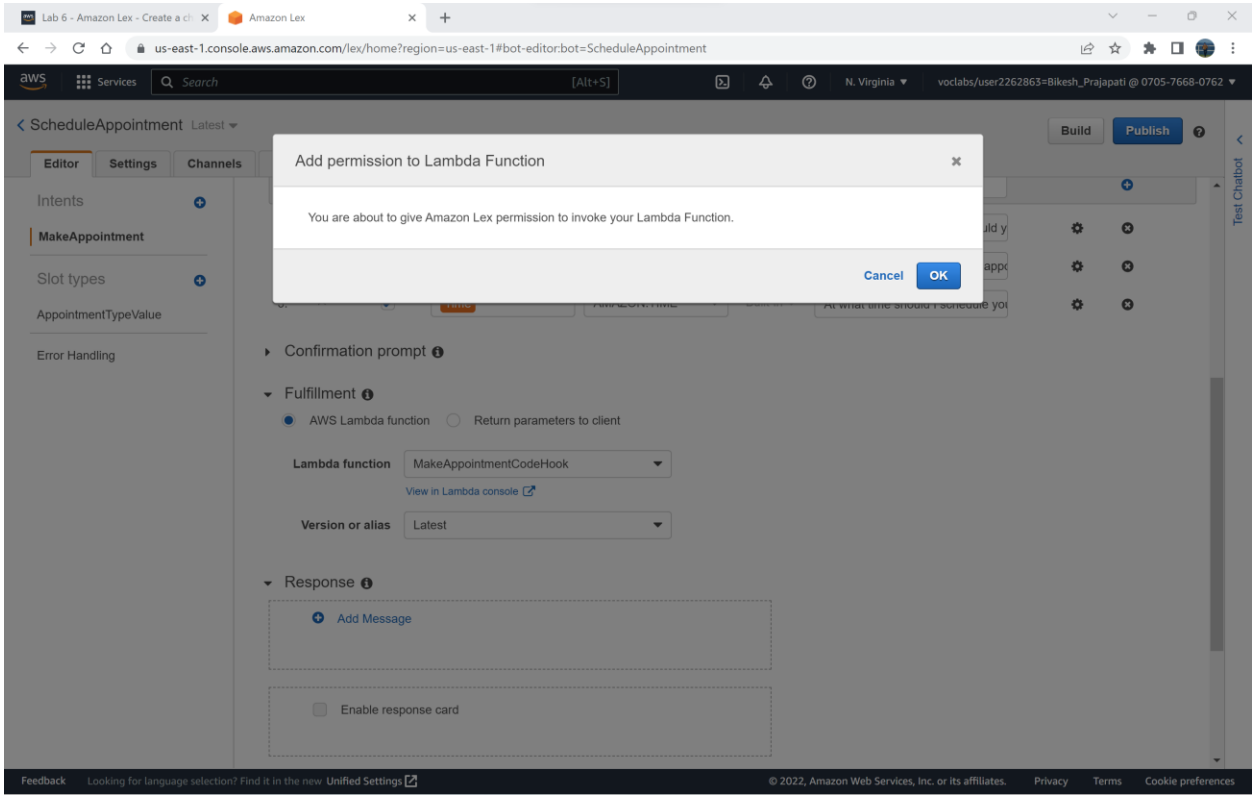


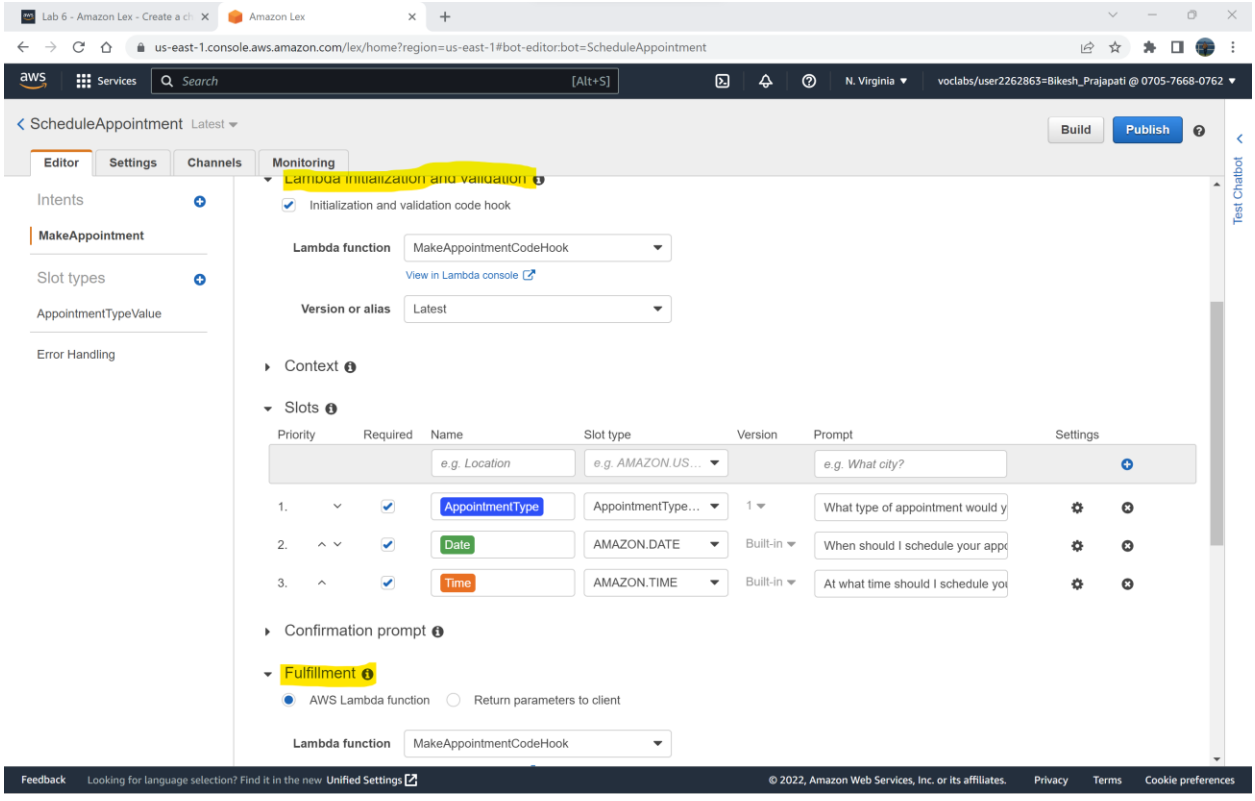


Here is the Lambda Designer window which is ready to use.

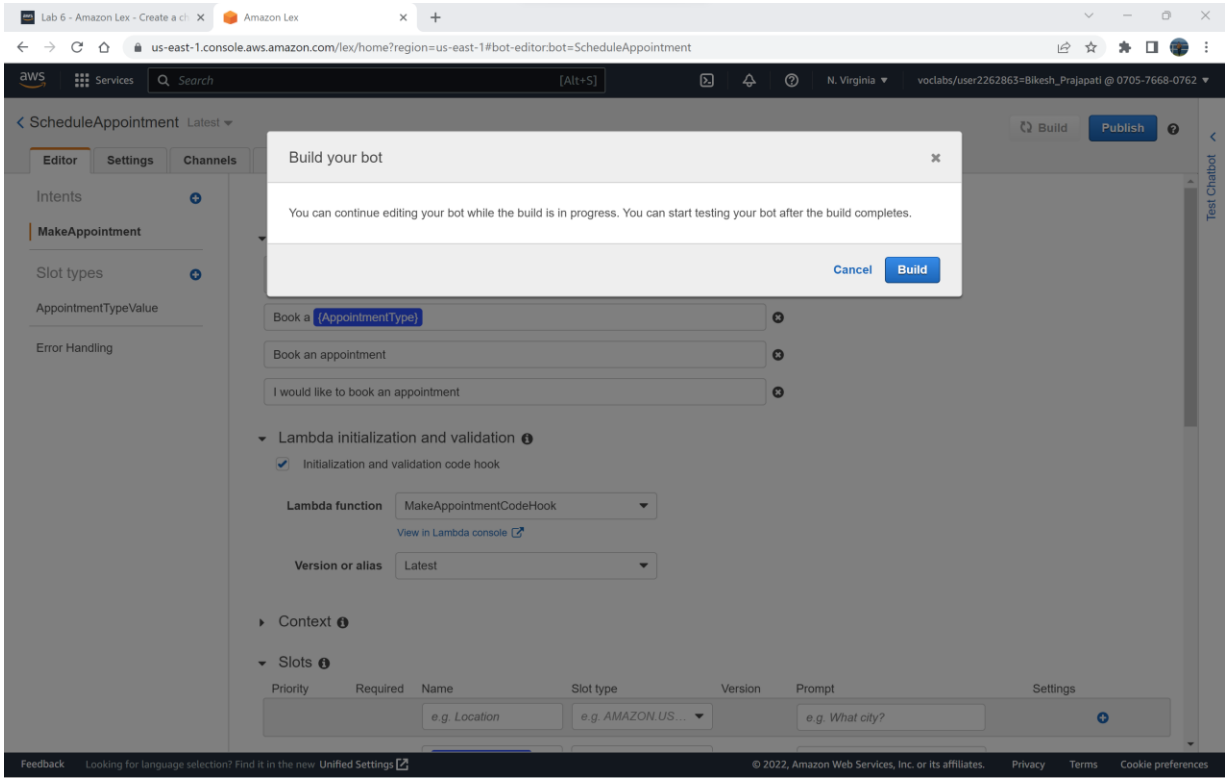


On the designer page, we choose test and provide the event name that we have create. The click test of confirm. we got succeeded message.

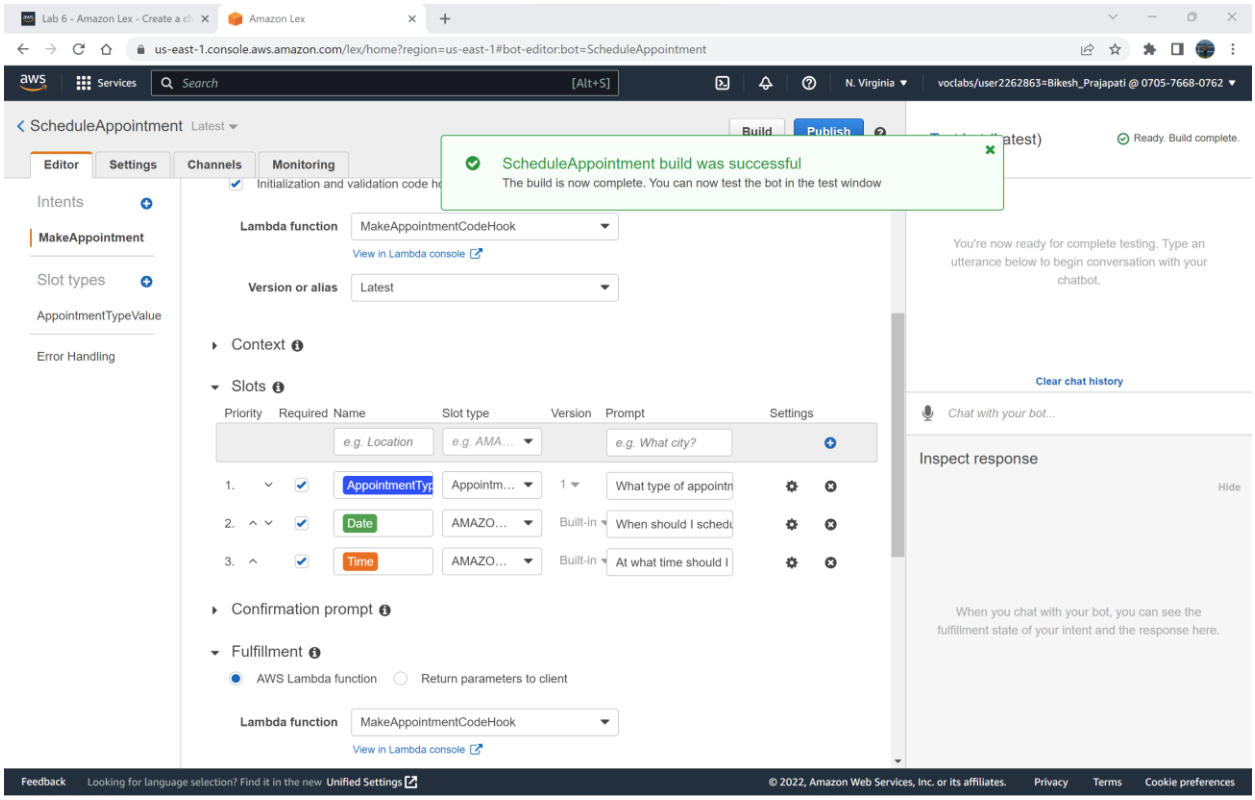


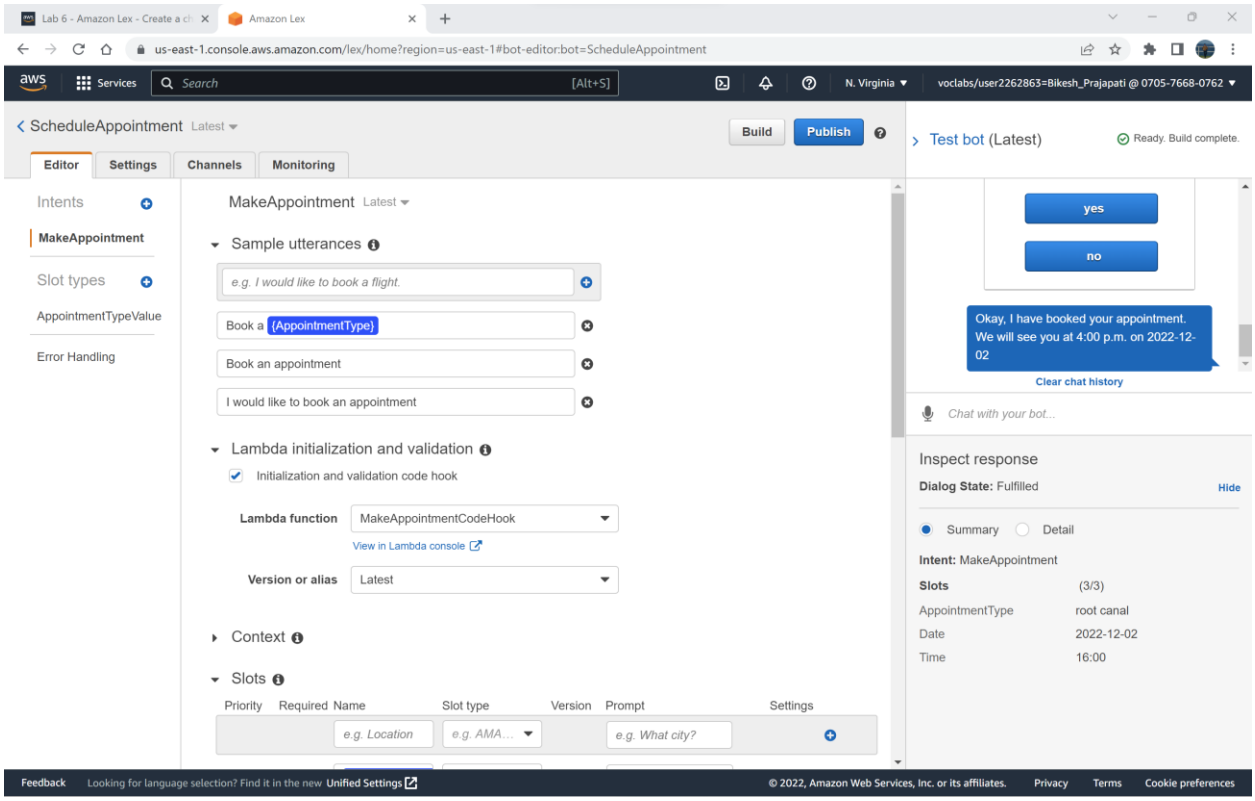


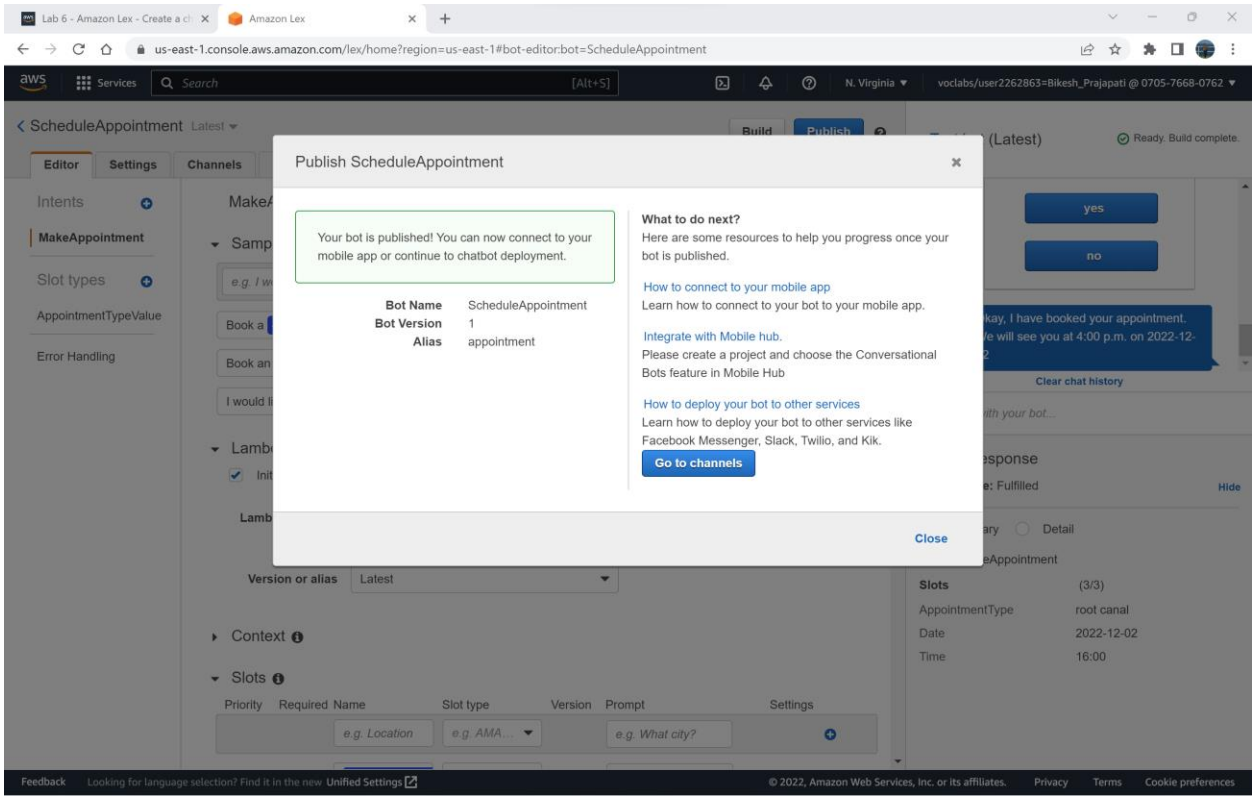
Now, In our previous bot that we have created, we will be selected the lambda function that we have created for both lambda initialization and validation and fulfillment and save indent.

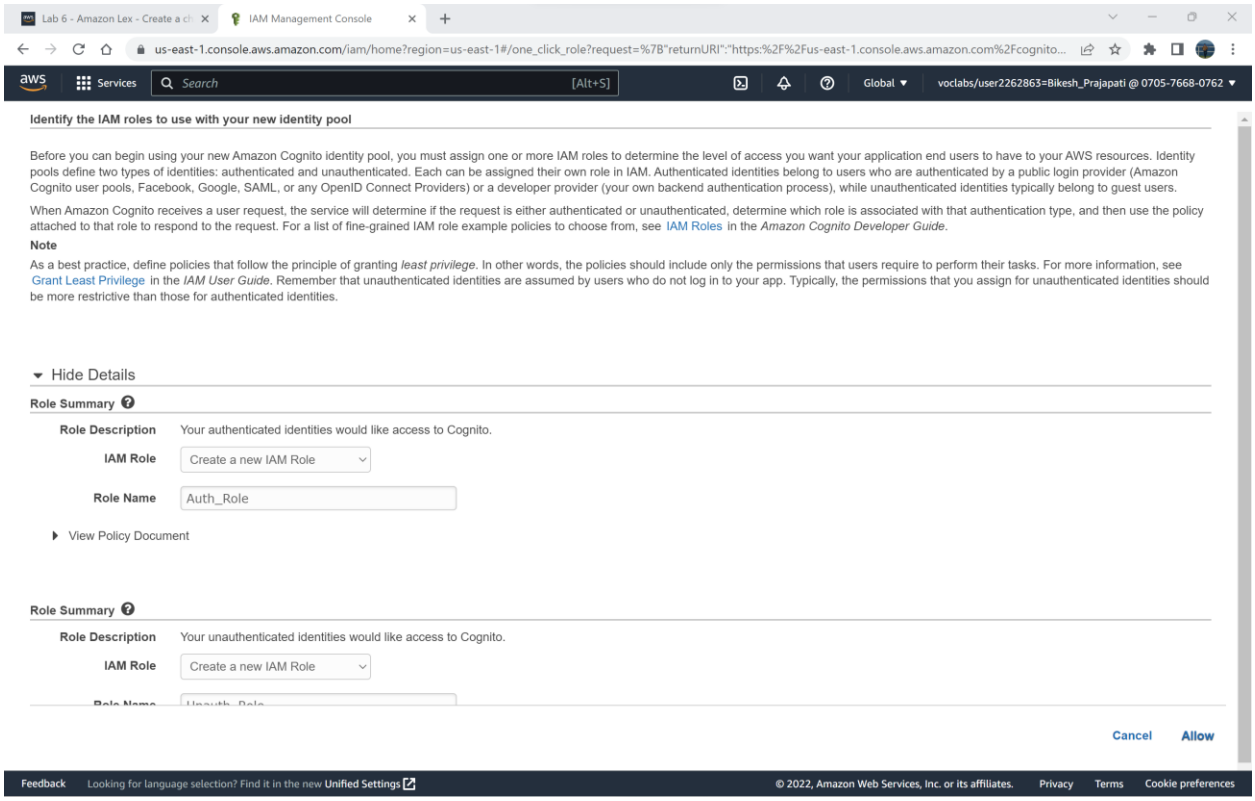


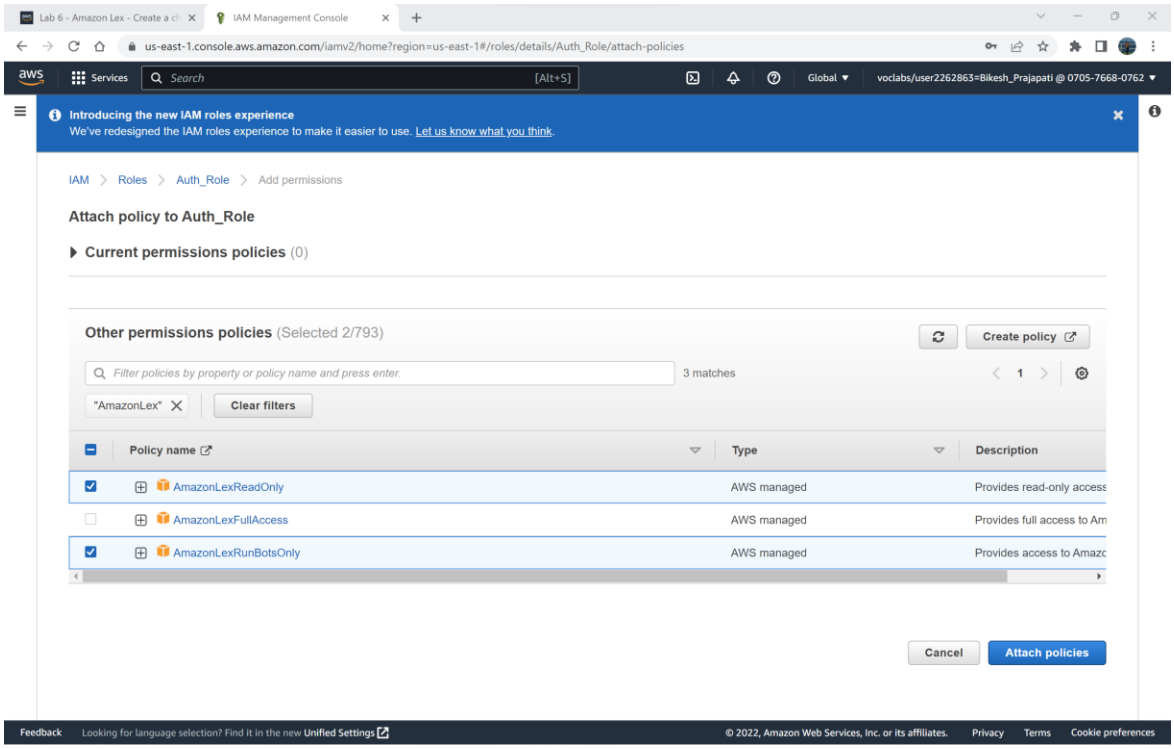
We select build to test out bot system that we have added lambda fucntion.



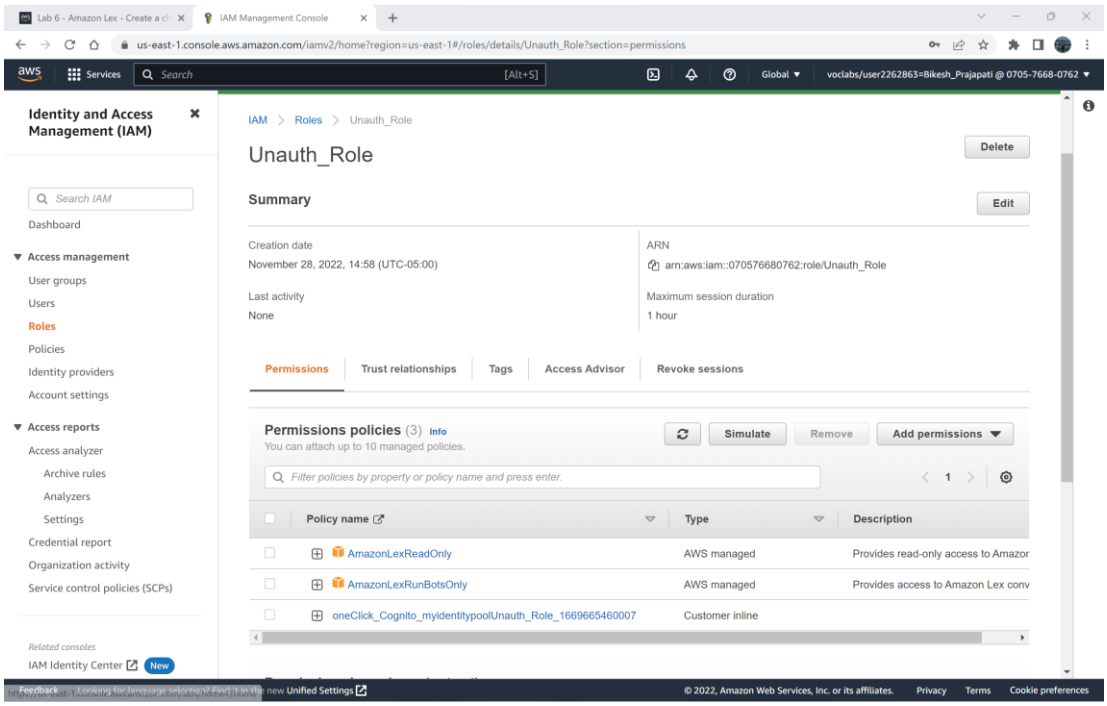

Lab 6 - Amazon Lex - Create a ch
Amazon Lex
us-east-1.console.aws.amazon.com/lex/home?region=us-east-1#bot-editor:bot=ScheduleAppointment
Services
Search
[Alt+S]
N. Virginia
voclabs/user2262863=Bikesh_Prajapati @ 0705-7668-0762
ScheduleAppointment
Latest
Build
Publish
Editor
Settings
Channels
Monitoring
Intents
MakeAppointment
Slot types
AppointmentTypeValue
Error Handling
MakeAppointment
Latest
Sample utterances
e.g. I would like to book a flight.
Book a {AppointmentType}
Book an appointment
I would like to book an appointment
Lambda initialization and validation
Initialization and validation code hook
Lambda function
MakeAppointmentCodeHook
View in Lambda console
Version or alias
Latest
Context
Slots
Priority
Required
Name
Slot type
Version
Prompt
Settings
e.g. Location
e.g. AMA...
e.g. What city?
Test bot (Latest)
Ready. Build complete.
yes
no
Okay, I have booked your appointment.
We will see you at 4:00 p.m. on 2022-12-02
Clear chat history
Chat with your bot...
Inspect response
Dialog State: Fulfilled
Hide
Summary
Detail
Intent: MakeAppointment
Slots
(3/3)
AppointmentType
root canal
Date
2022-12-02
Time
16:00
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Publish ScheduleAppointment
Your bot is published! You can now connect to your mobile app or continue to chatbot deployment.
Bot Name
ScheduleAppointment
Bot Version
1
Alias
appointment
What to do next?
Here are some resources to help you progress once your bot is published.
How to connect to your mobile app
Learn how to connect to your bot to your mobile app.
Integrate with Mobile hub.
Please create a project and choose the Conversational Bots feature in Mobile Hub
How to deploy your bot to other services
Learn how to deploy your bot to other services like Facebook Messenger, Slack, Twilio, and Kik.
Go to channels
Close

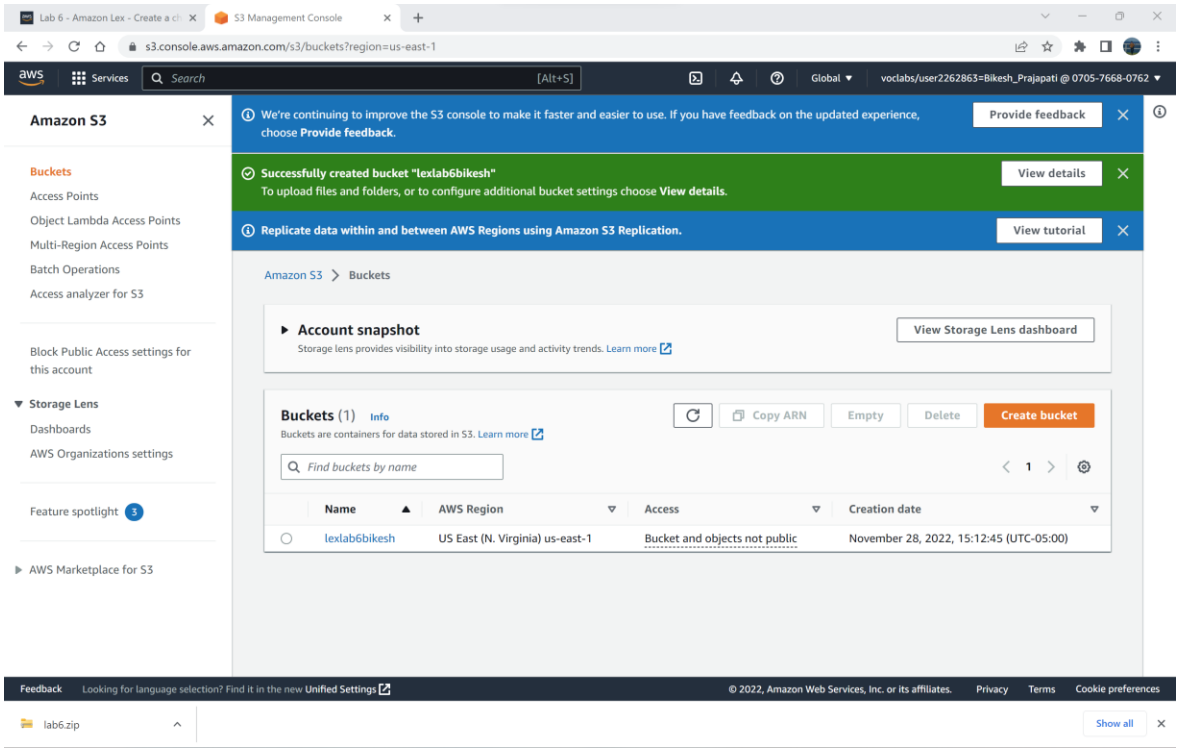


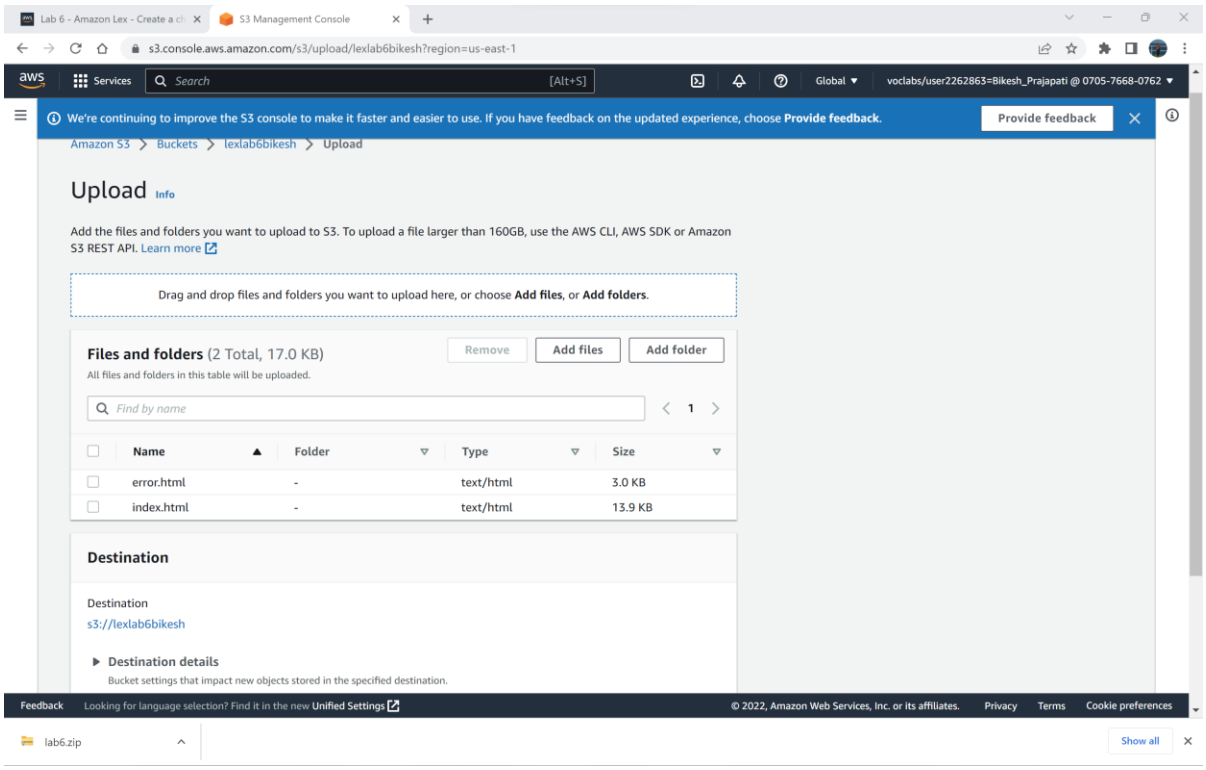


Here we select the appropriate setting for the permission police which we jus select amazon lex read only and amazon lex run bots only.

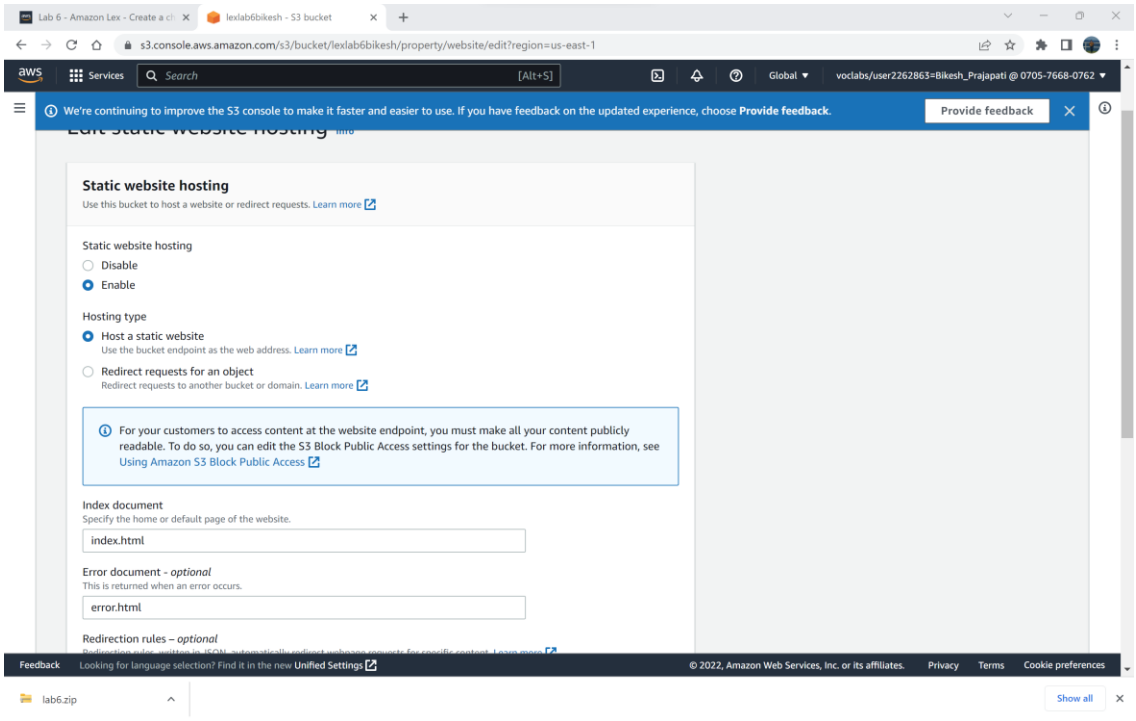


We do that permission policies setting for both auth_role and unauth_role.

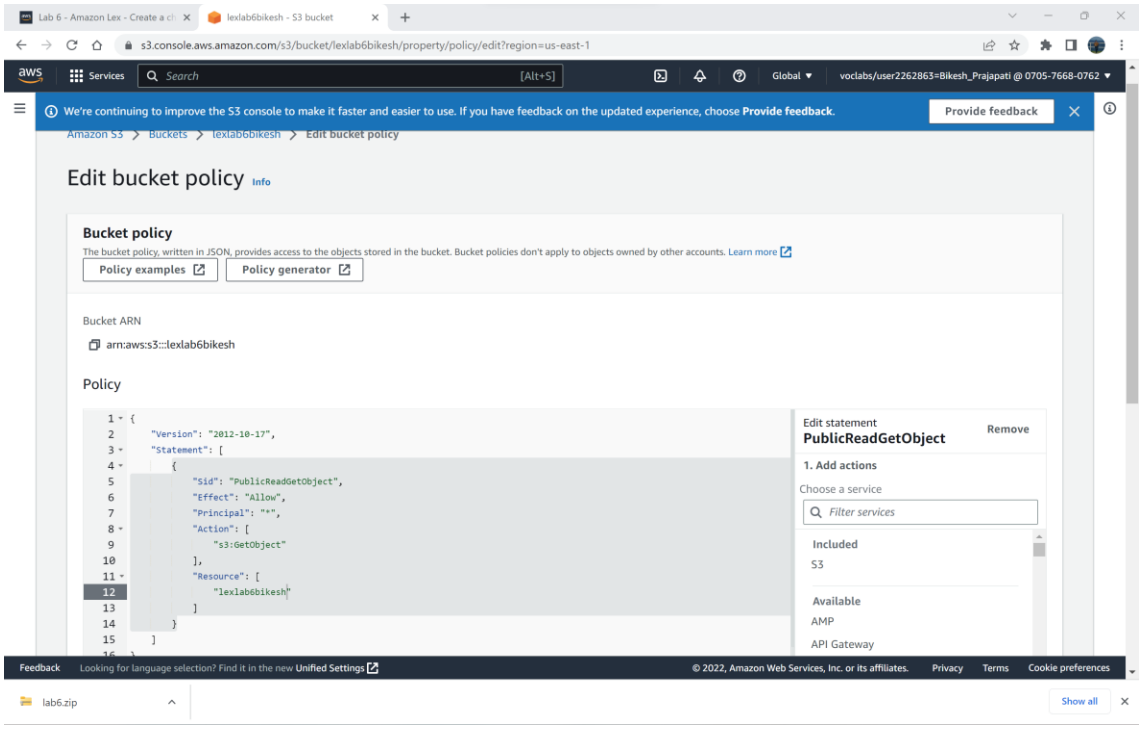




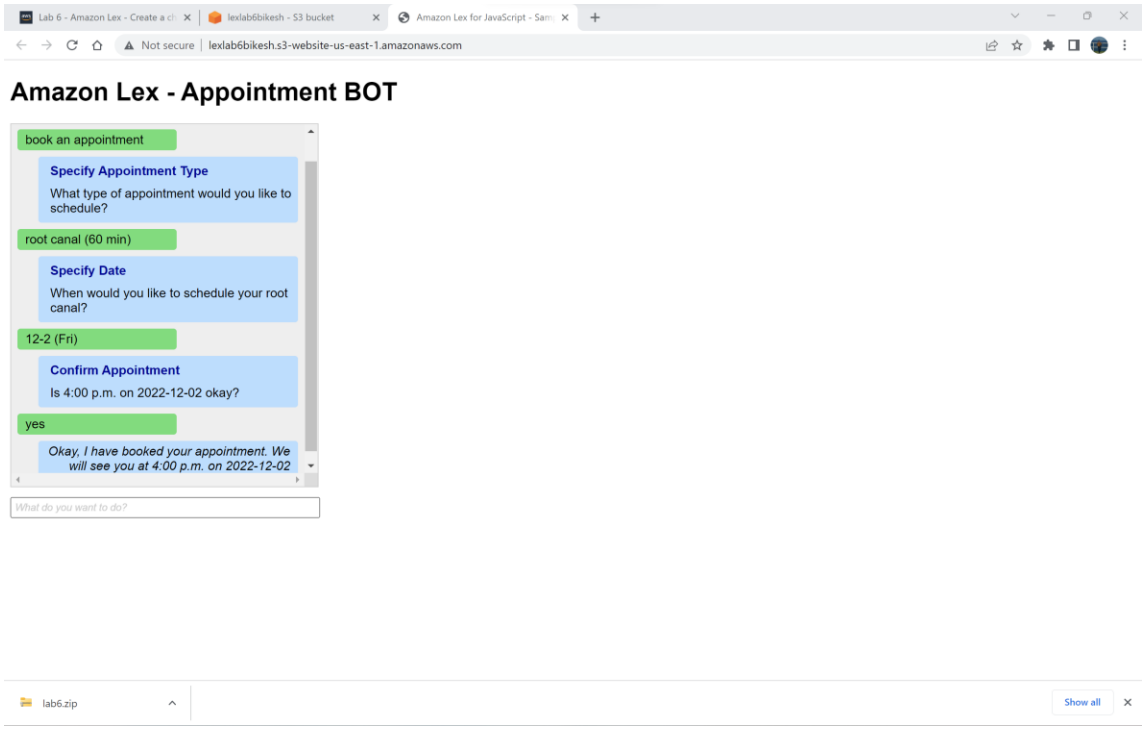
After that, here we upload the hard coded html files that we create the chat bot. files names are error.htm and index.html.



These are the required setting for hosting static website in S3.



Here we edit some bucket policy. We change our resource name to the name of bucket that we have created before which is lexlab6bikesh.



Finally, we lunch our chat bot website. Here we test the same thing as booking the appointment. Chat bot repones as we expected and even booked the data we wanted. And send us confirmation text. This is how we create the website chat bot.

At the end, we have managed to complete all the step for this lab. We successfully complete all the lab for this module.