

2022

# Final Project - AWS Machine Learning

BDM 1034 APPLICATION DESIGN FOR BIG DATA  
GROUP E

## Table of Contents

Group Members who have shown participation .....	2
LAB 3.....	3
Lab 3.1 - Amazon Sage Maker - Creating and importing data.....	3
Lab 3.2 Evaluate the data .....	7
Lab 3.3 Amazon Sage Maker - Encoding Categorical Data .....	12
Lab 3.4 Amazon Sage Maker - Training a Machine Learning Model .....	18
Lab 3.5 Machine Learning Pipeline with Amazon Sage Maker.....	22
Lab 3.6 - Amazon Sage Maker - Generating model performance metrics .....	26
Lab 3.7 - Tuning Hyperparameters.....	33
Lab 4 - Creating a forecast with Amazon Forecast .....	40
Lab 5 Guided Lab: Facial Recognition.....	55
LAB 6 Guided Lab: Natural Language Processing – Create chatbot .....	61

## Group Members who have shown participation

<b>Student Name</b>	<b>Student ID</b>
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

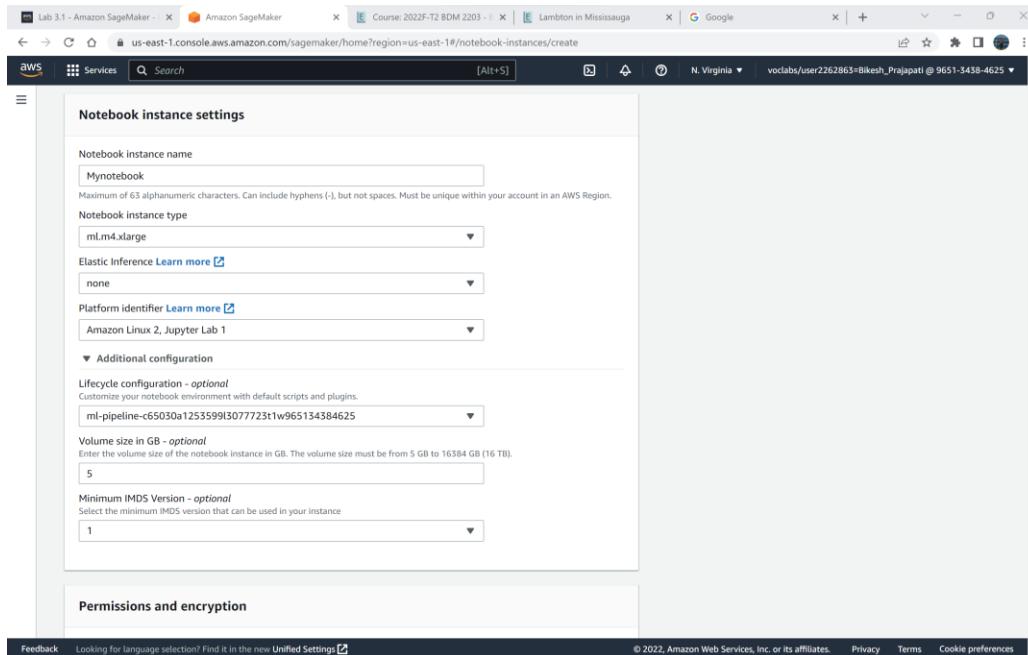
The screenshot shows a browser window with multiple tabs open. The active tab is titled 'Lab 3.1 - Amazon SageMaker'. The page displays a 'Start Lab' dialog box. Inside the dialog, the following information is visible:

- Region: us-east-1
- Lab ID: arn:aws:cloudformation:us-east-1:965134384625:stack/c65030a125359913077723t1w965134384625@e520cb0-6077-11ed-9449-12387d27f6df
- Creation Time: 2022-11-09T13:43:32-0800
- Start session at: 2022-11-09T13:43:33-0800
- Remaining session time: 02:00:00 (120 minutes)
- Lab status: ready

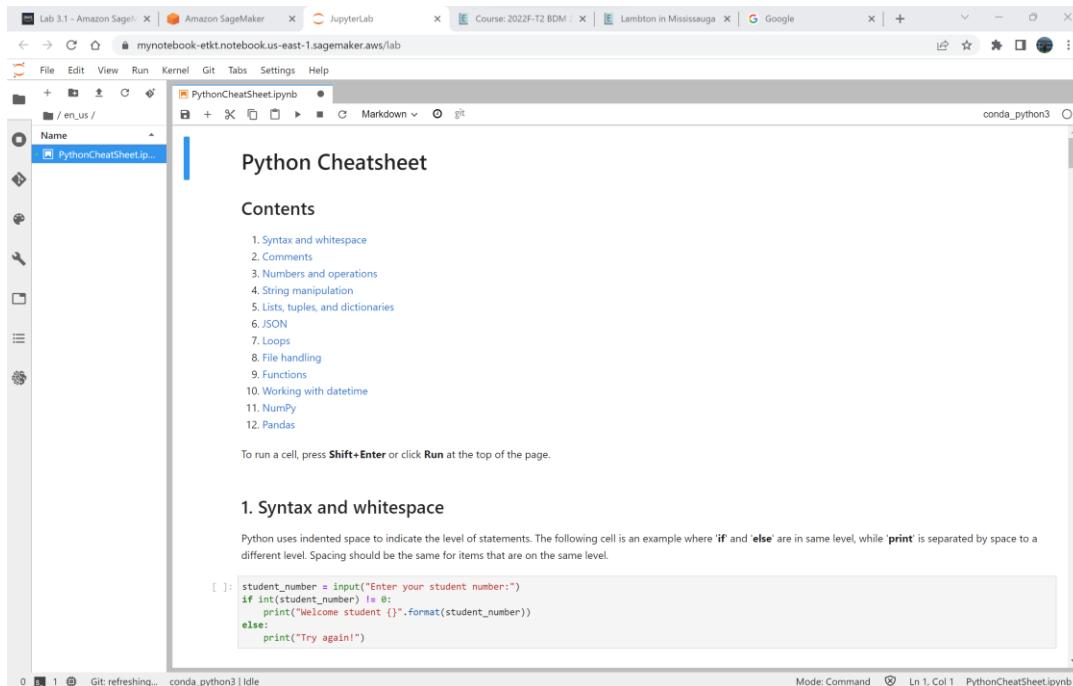
Below the dialog, a message reads: "windows. Select the banner or icon and then choose Allow pop ups." Navigation buttons at the bottom include '< Previous' and 'Next >'.

The screenshot shows the Amazon SageMaker console with the URL [us-east-1.console.aws.amazon.com/sagemaker/home?region=us-east-1#notebook-instances](https://us-east-1.console.aws.amazon.com/sagemaker/home?region=us-east-1#notebook-instances). The left sidebar contains navigation links such as 'Getting started', 'Control panel', 'Studio', 'Studio Lab', 'Canvas', 'RStudio', 'SageMaker dashboard', 'Images', 'Lifecycle configurations', 'Search', 'Ground Truth', 'Notebook' (with 'Notebook instances' selected), 'Processing', 'Training', 'Inference', and 'Edge Manager'. The main content area is titled 'Notebook instances' and includes a search bar and a table with columns: Name, Instance, Creation time, Status, and Actions. A message at the bottom of the table says 'There are currently no resources.' Navigation buttons at the bottom right include '< 1 >'. The top of the page shows standard browser controls and a header with the AWS logo and 'Services'.

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.

The screenshot shows a JupyterLab environment with multiple tabs at the top. The active tab is 'linear\_learner\_mnist.ipynb'. The left sidebar displays 'TERMINAL SESSIONS' and 'KERNEL SESSIONS', both of which have three entries: 'PythonCheatSheet.ipynb', 'linear\_learner\_mnist.ipynb', and 'Untitled.ipynb', all marked as 'SHUT DOWN'. The main content area contains the following text:

## An Introduction to Linear Learner with MNIST

**Making a Binary Prediction of 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 ( $\mathbf{x}$ ,  $y$ ).  $\mathbf{x}$  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  $\mathbf{x}$  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.

The screenshot shows a JupyterLab environment with multiple tabs at the top. The active tab is 'Untitled.ipynb'. The left sidebar displays 'TERMINAL SESSIONS' and 'KERNEL SESSIONS', both of which have two entries: 'PythonCheatSheet.ipynb' and 'linear\_learner\_mnist.ipynb', both marked as 'SHUT DOWN'. A 'Select Kernel' dialog box is open in the center of the screen, prompting the user to select a kernel for the 'Untitled.ipynb' notebook. The dialog box contains the following text:

Select Kernel  
Select kernel for: "Untitled.ipynb"  
conda\_python3  
Select

The screenshot shows a Jupyter Notebook interface with multiple tabs at the top. The active tab is 'Lab3.1\_bikesh.ipynb'. The notebook contains the following code:

```

import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

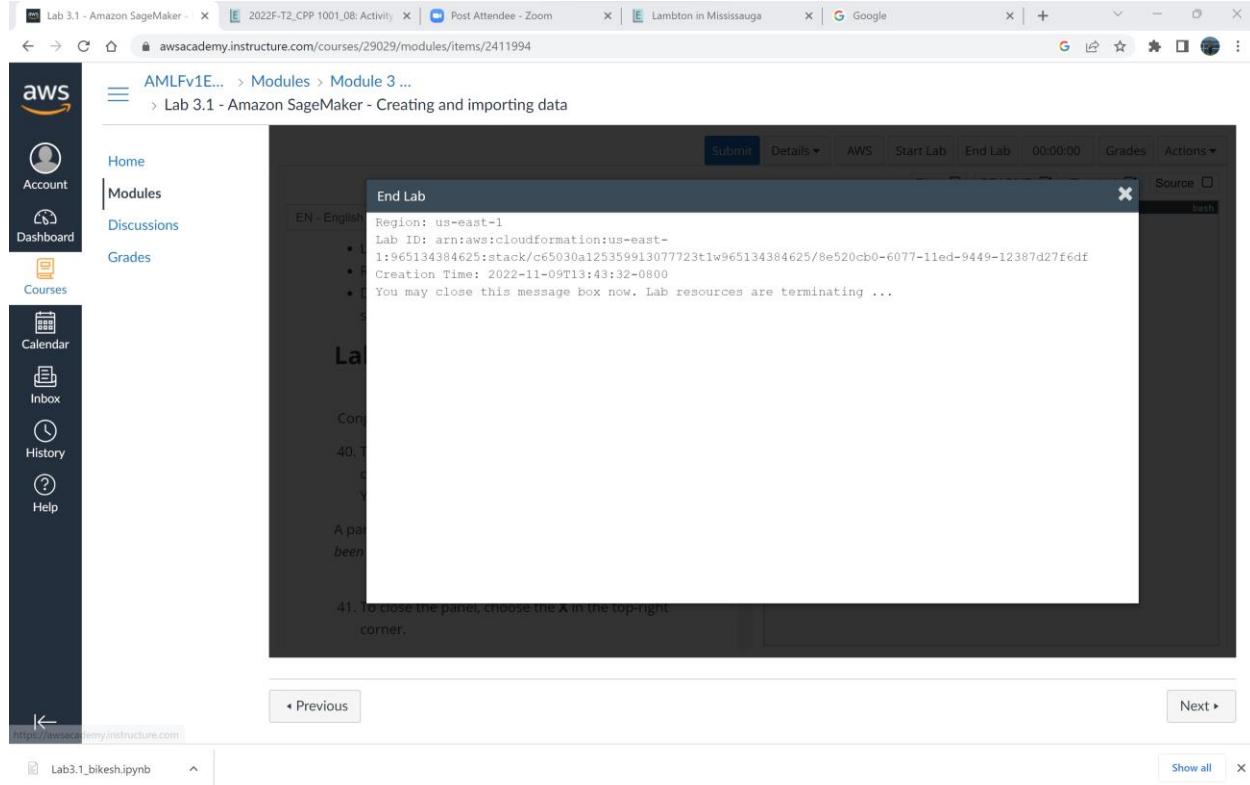
data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])
df.head()

```

Below the code, a table shows the first five rows of the DataFrame:

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254400	b'Abnormal'
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	b'Abnormal'
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	b'Abnormal'
3	69.297000	24.652878	44.311238	44.644130	101.868495	11.211523	b'Abnormal'
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	b'Abnormal'

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 our first lab and we completed our lab successfully.



## Lab 3.2 Evaluate the data

The screenshot shows a browser window for 'awsacademy.instructure.com' with a sub-path 'courses/29029/modules/items/2411998'. The main content area displays a 'Start Lab' panel titled 'EN - English'. It contains information about the lab's region ('us-east-1'), ID ('arn:aws:cloudformation:us-east-1:819985883415:stack/c65030a125360113112496t1w819985883415/8fd397c0-6455-11ed-bec6-0a9c666f2c27'), and creation time ('2022-11-14T11:50:16-0800'). It also shows the start session time ('2022-11-14T11:50:17-0800') and remaining session time ('02:00:00 (120 minutes)'). A note states 'Lab status: ready'. Below the panel, a caption reads: 'A Start Lab panel opens, which displays the lab status.' Navigation buttons for 'Previous' and 'Next' are visible at the bottom.

The screenshot shows a JupyterLab interface with a tab bar including 'Lab 3.2 - Amazon SageMaker', 'Amazon SageMaker', and 'JupyterLab'. The main area is titled 'Lab setup' and contains instructions: 'Because this solution is split across several labs in this module, you must run the following cells so that you can load the data:'. Below this, a code cell shows the following Python code:

```

import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

f_zin = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
r = requests.get(f_zin, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

data = arff.loadarff('column_0C_weka.arff')
df = pd.DataFrame(data[0])

```

Following this, a section titled 'Step 1: Exploring the data' provides instructions: 'You will start by looking at the data in the dataset.' It includes a note: 'To get the most out of this lab, carefully read the instructions and code before you run the cells. Take time to experiment!' and a tip: 'First, you will use `shape` to examine the number of rows and columns'. The code cell for this step is:

```

df.shape

```

Further down, another code cell is shown:

```

df.columns

```

A note next to it says: 'You can see the six biomechanical features, and the target column is named `class`'.

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.

Lab 3.2 - Amazon SageMaker - E | mynotebook-2uh9.notebook.us-east-1.sagemaker.aws/lab | JupyterLab

File Edit View Run Kernel Git Tabs Settings Help

/ en\_us /

Name Last Modified

- 3\_2-machinelearning.ipynb 6 minutes ago
- column\_2C\_weka.arff seconds ago
- column\_2C.dat seconds ago
- column\_3C\_weka.arff seconds ago
- column\_3C.dat seconds ago

**Step 1: Exploring the data**

You will start by looking at the data in the dataset.

To get the most out of this lab, carefully read the instructions and code before you run the cells. Take time to experiment!

First, you will use `shape` to examine the number of rows and columns

```
[4]: df.shape
[4]: (310, 7)
```

You will now get a list of the columns.

```
[5]: df.columns
[5]: Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class'],
       dtype='object')
```

You can see the six biomechanical features, and the target column is named `class`.

What column types do you have?

```
[6]: df.dtypes
[6]: pelvic_incidence    float64
pelvic_tilt            float64
lumbar_lordosis_angle float64
sacral_slope           float64
pelvic_radius          float64
degree_spondylolisthesis   object
dtype: object
```

You have six floats for the biomechanical features, but the target is a class.

To look at the statistics for the first column, you can use the `describe` function.

Mode: Command | Ln 1, Col 1 | 3\_2-machinelearning.ipynb

Lab 3.2 - Amazon SageMaker - E | mynotebook-2uh9.notebook.us-east-1.sagemaker.aws/lab | JupyterLab

File Edit View Run Kernel Git Tabs Settings Help

/ en\_us /

Name Last Modified

- 3\_2-machinelearning.ipynb 2 minutes ago
- column\_2C\_weka.arff 3 minutes ago
- column\_2C.dat 3 minutes ago
- column\_3C\_weka.arff 3 minutes ago
- column\_3C.dat 3 minutes ago

`df.describe()`

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
count	310.000000	310.000000	310.000000	310.000000	310.000000	310.000000
mean	60.496653	17.542822	51.930930	42.953831	117.920655	26.296694
std	17.236526	10.008330	18.554064	13.423102	13.317377	37.559027
min	26.147921	-6.554948	14.000000	13.366931	70.082575	-11.058179
25%	46.430294	10.667069	37.000000	33.347122	110.709196	1.603727
50%	58.691038	16.357688	49.562398	42.404912	118.268178	11.767934
75%	72.877696	22.120395	63.000000	52.695888	125.467674	41.287352
max	129.834041	49.431864	125.742385	121.429566	163.071041	418.543082

\*\*Question:\*\* Are there any features that aren't well-distributed? Are there any features with outliers that you want to look at? Does it look like there might be any correlations between features?

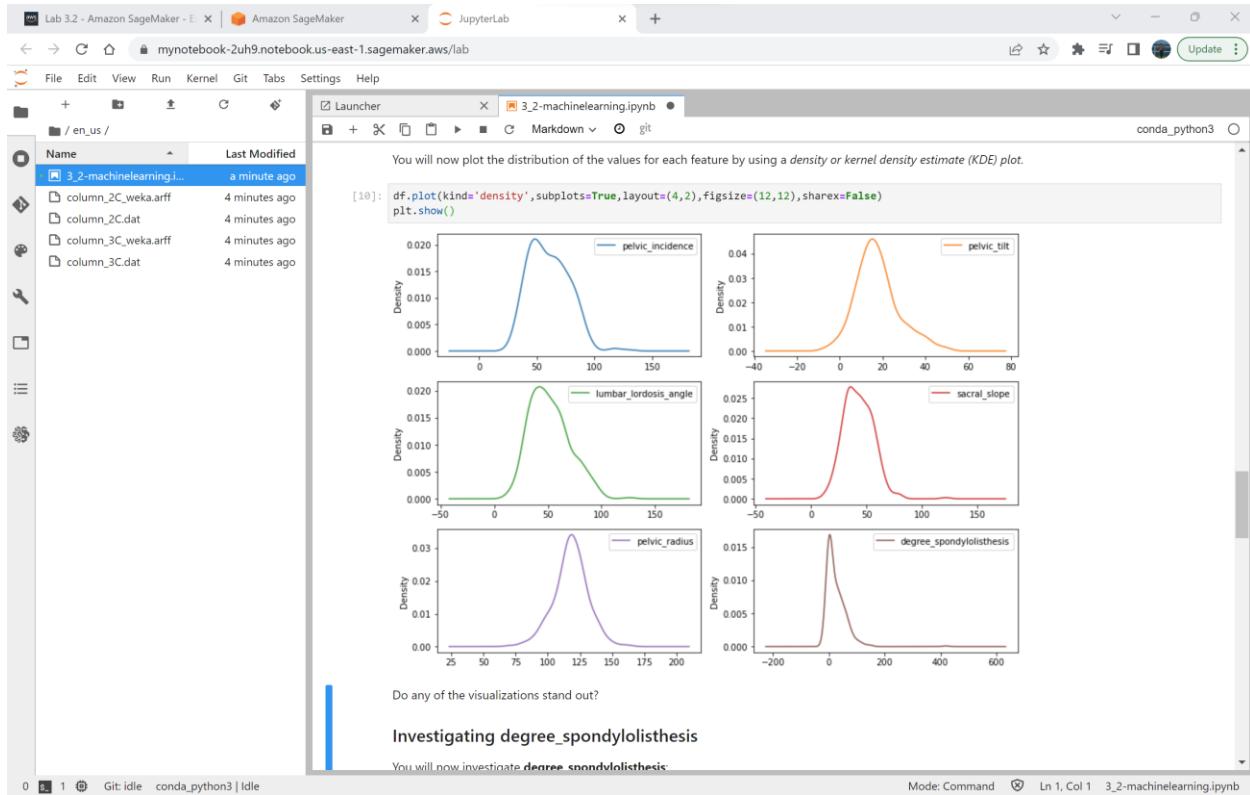
It's not always easy to make observations when you look only at numbers, so you will now plot these values.

```
[9]: import matplotlib.pyplot as plt
%matplotlib inline
df.plot()
```

[9]: <AxesSubplot:>

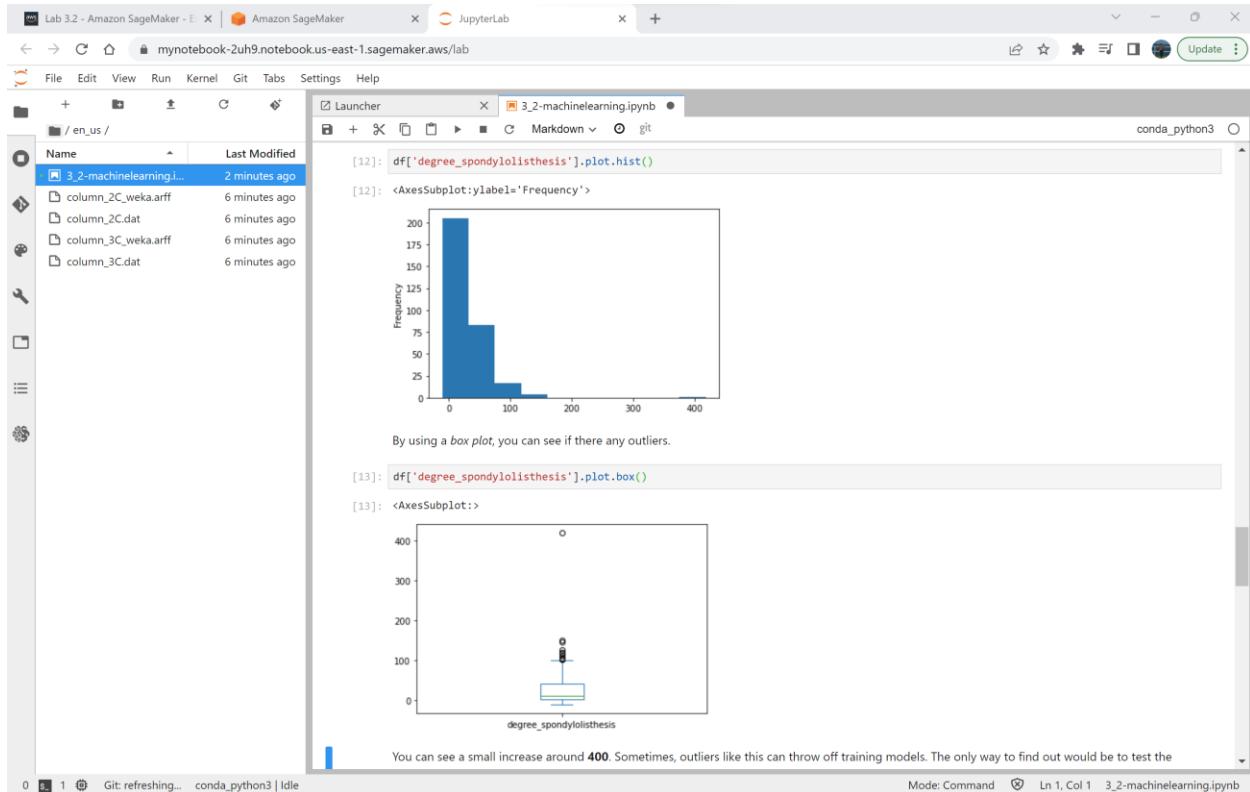
Mode: Command | Ln 1, Col 1 | 3\_2-machinelearning.ipynb

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.

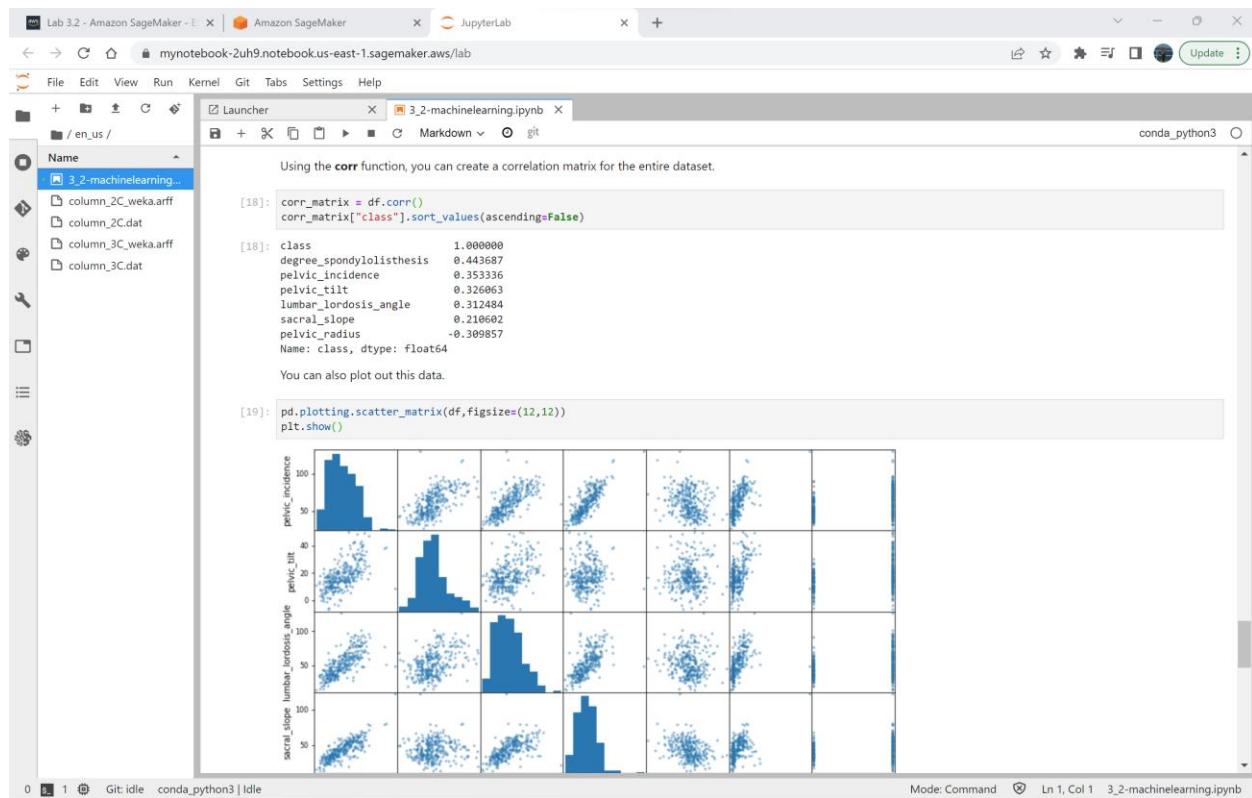
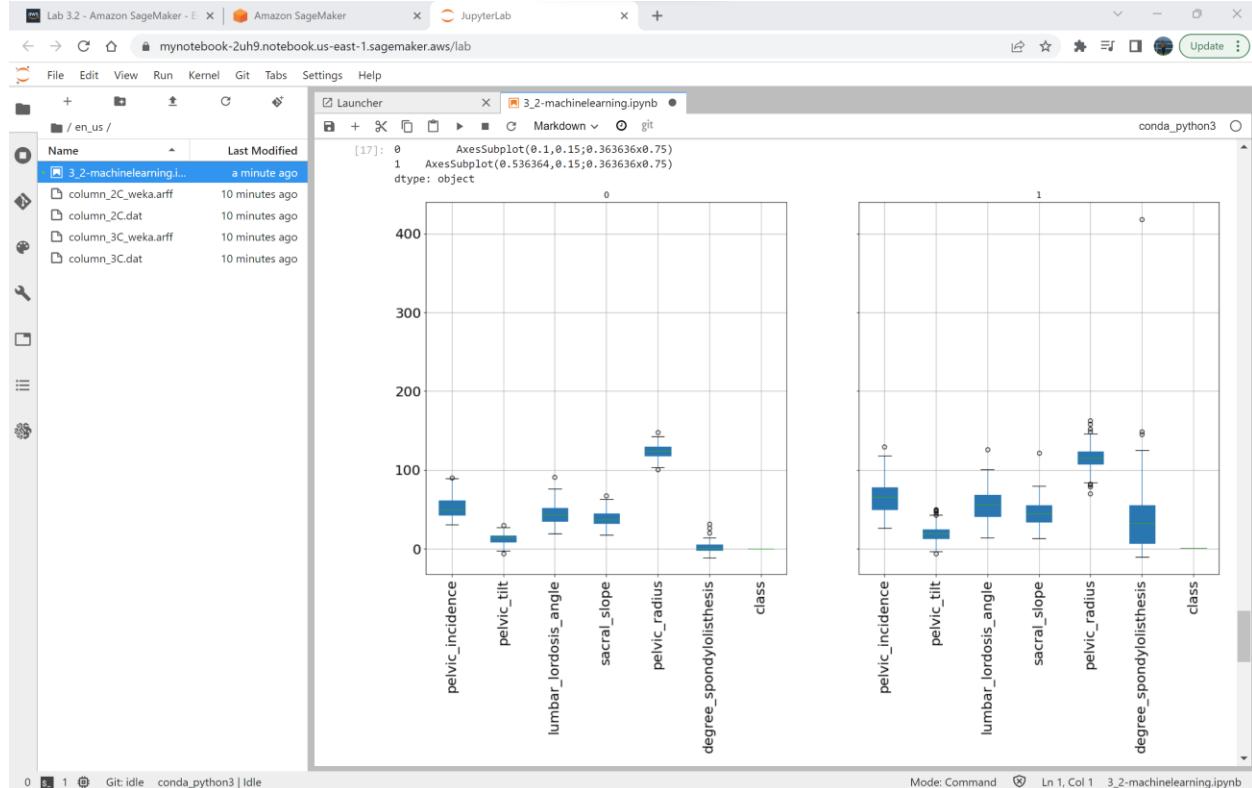


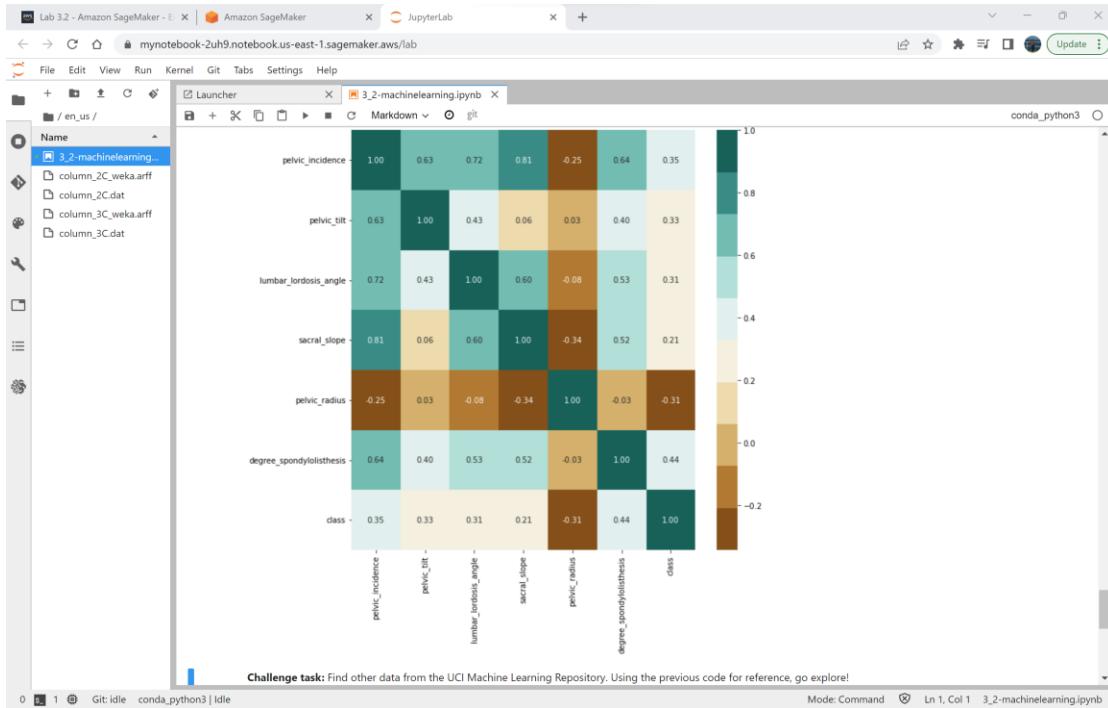
Yes, the visualization are stand out.

Yes, there is an outlier in degree.

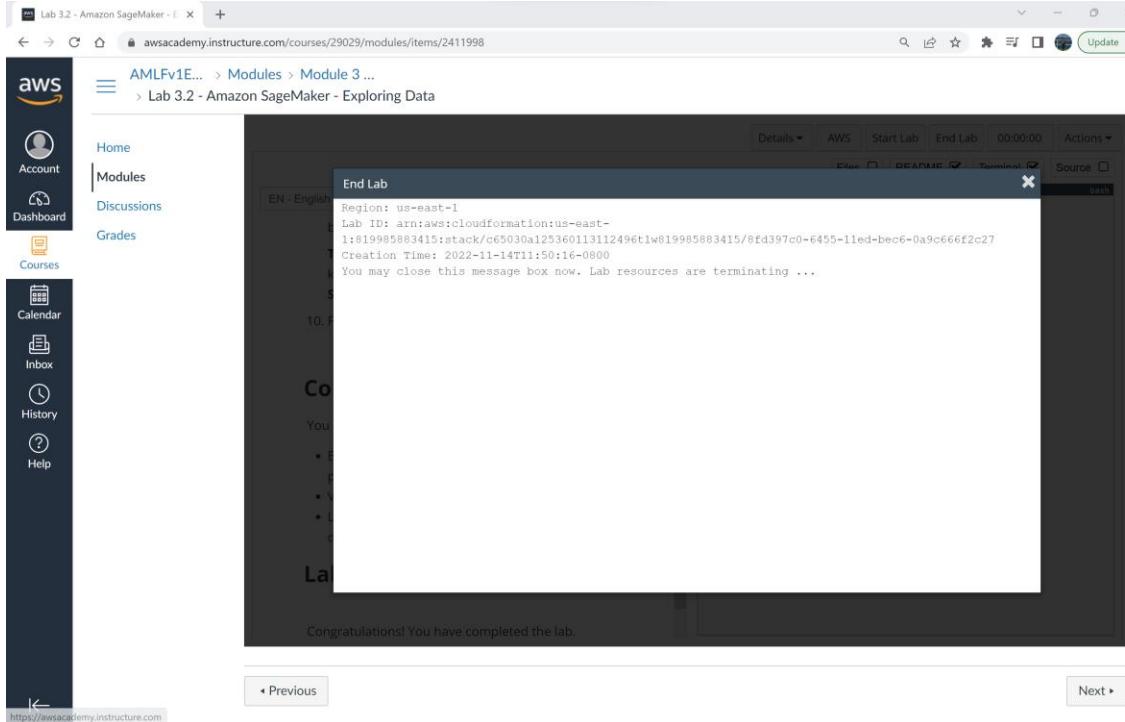


Here are some visualization that we have create to see the outlier and using box plot. As we can see, we can find some outlier as some data are out side the box plot.





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

The screenshot shows a browser window for 'awsacademy.instructure.com'. The URL is 'awsacademy.instructure.com/courses/29029/modules/items/2412002'. The page title is 'AMLFv1E... > Modules > Module 3... > Lab 3.3 - Amazon SageMaker - Encoding Categorical Data'. On the left, there's a sidebar with 'Account', 'Dashboard', 'Courses', 'Calendar', 'Inbox', 'History', and 'Help'. The main content area has a 'Start Lab' dialog box open. The dialog contains the following information:

- Region: us-east-1
- Lab ID: arn:aws:cloudformation:us-east-1:990522007975:stack/c65030a12536031112677t1w990522007975/0ae4a980-645c-11ed-8f3a-0a01120ff4a9
- Creation Time: 2022-11-14T12:36:40-0800
- Start session at: 2022-11-14T12:36:40-0800
- Remaining session time: 02:00:00 (120 minutes)
- Lab status: ready

Below the dialog, the page content includes a 'Prerequisites' section and navigation buttons for 'Previous' and 'Next'.

The screenshot shows a JupyterLab interface with the tab '3.3-machinelearning.ipynb' selected. The left sidebar shows a file tree with 'en\_us/' and two files: '3.3-machinelearning.ipynb' (4 minutes ago) and 'imports-85.csv' (2 years ago). The main content area displays the following code and instructions:

### Dataset attributions

This dataset was obtained from: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository (<http://archive.ics.uci.edu/ml>). Irvine, CA: University of California, School of Information and Computer Science.

### Step 1: Importing and exploring the data

You will start by examining the data in the dataset.

To get the most out of this lab, read the instructions and code before you run the cells. Take time to experiment!

Start by importing the pandas package and setting some default display options.

```
[ ]: import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

Next, load the dataset into a pandas DataFrame.

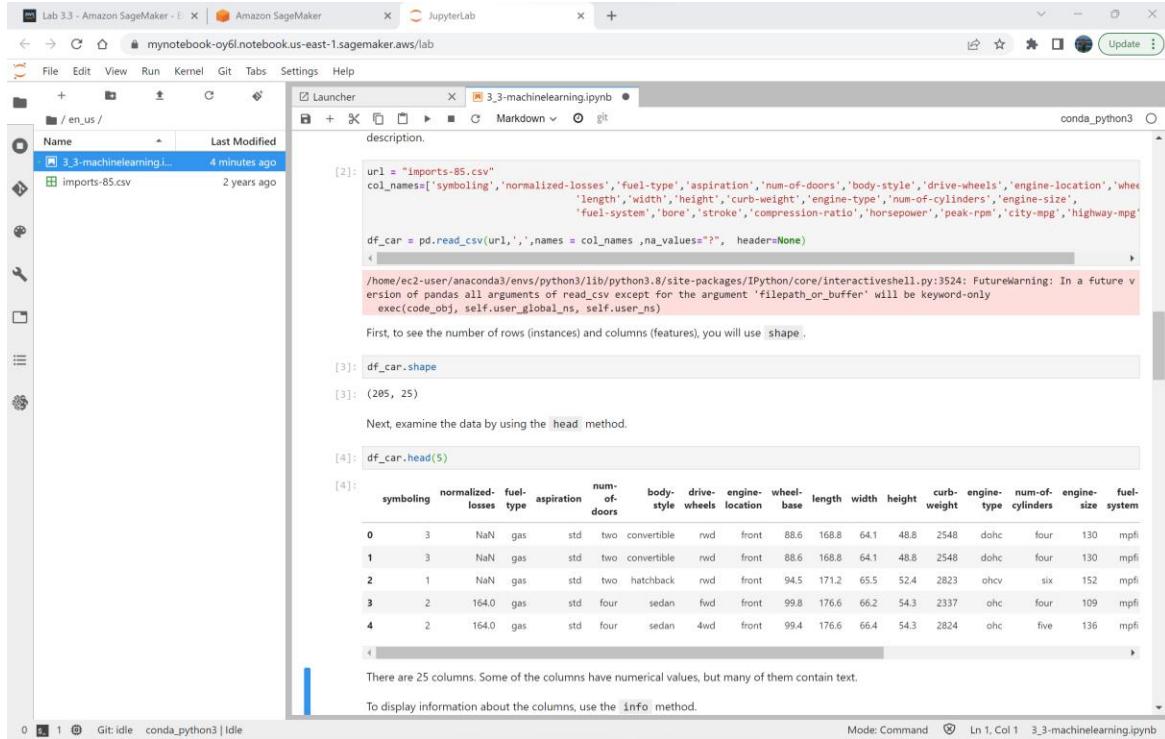
The data doesn't contain a header, so you will define those column names in a variable that's named `col_names` to the attributes listed in the dataset description.

```
[ ]: url = "imports-85.csv"
col_names=['symboling','normalized-losses','fuel-type','aspiration','num-of-doors','body-style','drive-wheels','engine-location','wheel-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)
```

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

```
[ ]: df_car.shape
```

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.



The screenshot shows a JupyterLab interface with a notebook titled "3\_3-machinelearning.ipynb". In cell [2], Python code is run to load a CSV file "imports-85.csv" into a DataFrame "df\_car". A warning message from pandas is displayed, indicating that the 'filepath\_or\_buffer' argument will be keyword-only in a future version. In cell [3], the shape of the DataFrame is checked, resulting in the output "(285, 25)". In cell [4], the first 5 rows of the DataFrame are displayed using the `head(5)` method, showing columns like 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, and fuel-system.

```

[2]: url = "imports-85.csv"
col_names=['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']
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 version 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]: (285, 25)

Next, examine the data by using the .head() method.

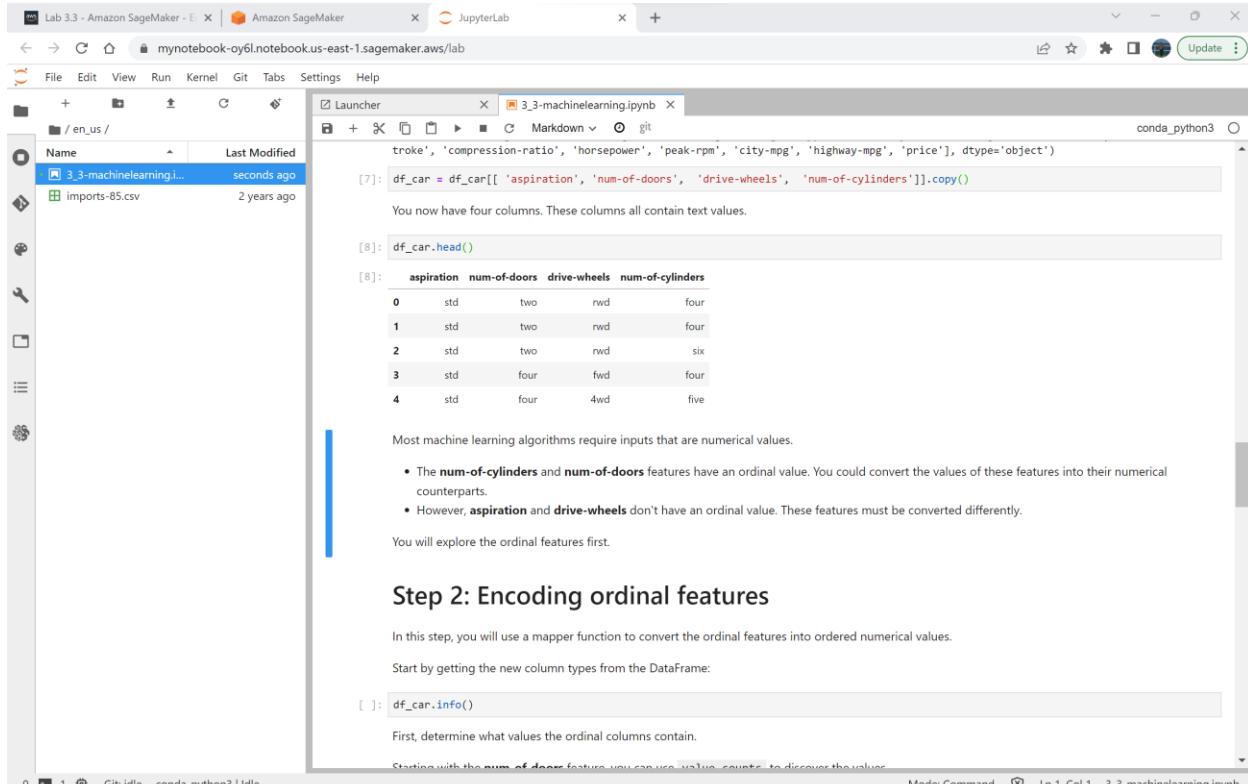
[4]: df_car.head(5)
[4]:

```

	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.



The screenshot shows a JupyterLab interface with a notebook titled "3\_3-machinelearning.ipynb". In cell [7], the user copies four specific columns from the DataFrame "df\_car" into a new DataFrame "df\_car\_cpy": "aspiration", "num-of-doors", "drive-wheels", and "num-of-cylinders". A note states that these columns contain text values. In cell [8], the first 5 rows of the copied DataFrame are displayed using the `head(5)` method, showing the copied columns.

```

[7]: df_car_cpy = 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_cpy.head(5)
[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.

Starting with the `num_of_doors` feature, you can use `value_counts` to discover the values.

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

The screenshot shows a JupyterLab interface with a sidebar containing a file tree and a central workspace. In the workspace, a terminal tab shows the command `conda_python3`. A code editor tab displays the notebook `3_3-machinelearning.ipynb`. The code cell [10] contains:

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

The output shows the value counts for the 'num-of-doors' column:

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

A note states: "This feature only has two values: four and two. You can create a simple mapper that contains a dictionary." The code cell [11] contains:

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

A note states: "You can then use the `replace` method from pandas to generate a new numerical column based on the `num-of-doors` column." The code cell [12] contains:

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

A note states: "When you display the DataFrame, you should see the new column on the right. It contains a numerical representation of the number of doors." The code cell [13] contains:

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

The output shows the first five rows of the DataFrame with the 'doors' column converted:

	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.

The screenshot shows a JupyterLab interface with a sidebar containing a file tree and a central workspace. In the workspace, a terminal tab shows the command `conda_python3`. A code editor tab displays the notebook `3_3-machinelearning.ipynb`. The code cell [14] contains:

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

The output shows the value counts for the 'num-of-cylinders' column:

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

A note states: "Next, create the mapper." The code cell [15] contains:

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

A note states: "Apply the mapper by using the `replace` method." The code cell [16] contains:

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

The code cell [17] contains:

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

The output shows the first five rows of the DataFrame with the 'cylinders' column converted:

	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

Git: idle    conda\_python3 | Idle    Mode: Command    Ln 1, Col 1    3\_3-machinelearning.ipynb

Here, we repeat 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.

The screenshot shows a JupyterLab interface within an Amazon SageMaker notebook. The left sidebar displays a file tree with a CSV file named 'imports-85.csv'. The main area shows a code cell [18] with the command `df_car['drive-wheels'].value_counts()`, which outputs:

```
fwd    120
rwd     76
4wd      9
Name: drive-wheels, dtype: int64
```

Below it, another cell [19] contains `df_car = pd.get_dummies(df_car,columns=['drive-wheels'])`. A third cell [20] shows the result of `df_car.head()`:

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

Text annotations explain the process: "According to the attribute description, `drive-wheels` has three possible values.", "Use the `get_dummies` method to add new binary features to the DataFrame.", "When you examine the dataset, you should see three new columns on the right:", and a bulleted list: • `drive-wheels_4wd`, • `drive-wheels_fwd`, • `drive-wheels_rwd`. A note states: "The encoding was straightforward. If the value in the `drive-wheels` column is `4wd`, then a `1` is the value in the `drive-wheels_4wd` column. A `0` is the value for the other columns that were generated. If the value in the `drive-wheels` column is `fwd`, then a `1` is the value in the `drive-wheels_fwd` column, and so on." Another note says: "These binary features enable you to express the information in a numerical way, without implying any order." The status bar at the bottom indicates "Mode: Command" and "Ln 1, Col 1".

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.

The screenshot shows a JupyterLab interface with a sidebar containing a file tree and a central workspace. The workspace displays a notebook titled '3\_3-machinelearning.ipynb'. The code cell [21] contains the command `df\_car['aspiration'].value\_counts()`, which outputs a series of counts for 'std' (168) and 'turbo' (37). Cell [22] shows the use of `pd.get\_dummies` to encode the 'aspiration' column. Cell [23] shows the resulting head of the DataFrame, which includes columns for aspiration ('std', 'turbo') and drive-wheels ('4wd', 'fwd', 'rwd'). A challenge task is mentioned at the bottom of the notebook.

```
[21]: df_car['aspiration'].value_counts()
[22]: df_car = pd.get_dummies(df_car,columns=['aspiration'], drop_first=True)
[23]: df_car.head()
```

	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.

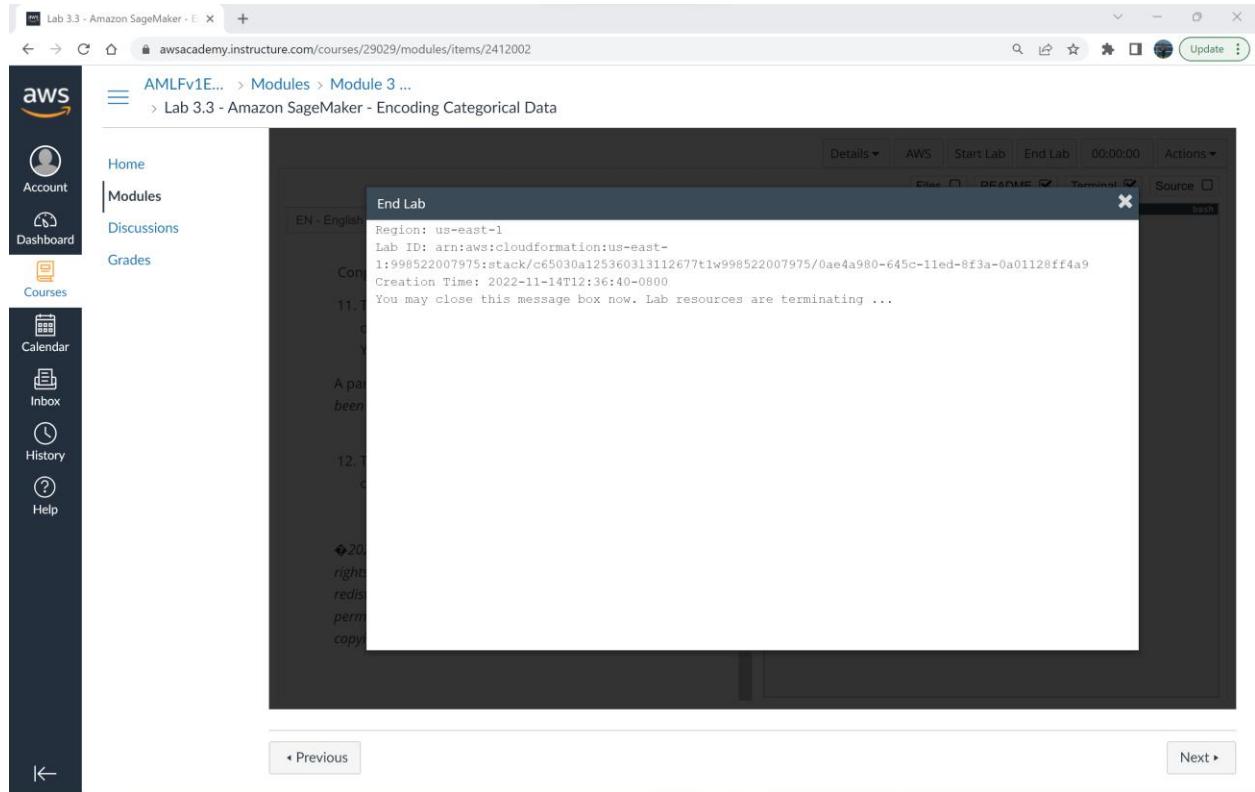
This screenshot shows the same JupyterLab interface as the previous one, but the workspace content has changed. The notebook '3\_3-machinelearning.ipynb' now includes code to drop the 'body-style' column and encode it using get\_dummies. The resulting DataFrame is shown with columns for 'aspiration', 'num-of-doors', 'drive-wheels', 'num-of-cylinders', and the newly encoded 'body-style' column. The 'body-style' column is represented as binary values (0 or 1) corresponding to the categories 'sedan', 'hatchback', 'wagon', 'hardtop', and 'convertible'.

```
[25]: df_car = df_car[['aspiration', 'num-of-doors', 'drive-wheels', 'num-of-cylinders','body-style']].copy()
[26]: df_car.head()
[27]: df_car['body-style'].value_counts()
[28]: df_car = pd.get_dummies(df_car,columns=['body-style'])
[29]: df_car.head()
```

	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

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

The screenshot shows the AWS Academy interface. On the left is a sidebar with icons for Account, Dashboard, Courses, Calendar, Inbox, History, and Help. The main area shows a navigation path: AMLFv1E... > Modules > Module 3... > Lab 3.4 - Amazon SageMaker - Training a Machine Learning Model. A central modal window titled "Start Lab" displays the following information:

```

Region: us-east-1
Lab ID: arn:aws:cloudformation:us-east-1:336316976938:stack/c65030a125360513112813t1w336316976938:c180f610-6460-11ed-acdf-0e69374a54e3
Creation Time: 2022-11-14T13:10:24-0800

Start session at: 2022-11-14T13:10:25-0800
Remaining session time: 02:00:00 (120 minutes)
Lab status: ready
  
```

Below the modal, a note says: "After completing this lab, you will be able to..." followed by a bulleted list: "Split data into training, validation and test". At the bottom are "Previous" and "Next" buttons.

The screenshot shows a JupyterLab interface with a tab bar for "Lab 3.4 - Amazon SageMaker" and "JupyterLab". The main area has a file browser on the left showing "en\_us/" folder with files "3\_4-machinelearning.ipynb" and "PythonCheatSheet.ipynb". The right pane is titled "Lab setup" and contains the following text and code snippets:

**Importing the data**

By running the following cells, the data will be imported and ready for use.

**Note:** The following cells represent the key steps in the previous labs.

```

[ ]: import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff
import boto3

[ ]: f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

[ ]: data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])

[ ]: class_mapper = {'Abnormal':1,'Normal':0}
df['class']=df['class'].replace(class_mapper)
  
```

**Step 1: Exploring the data**

You will start with a quick reminder of the data in the dataset.

To get the most out of this lab, carefully read the instructions and code before you run the cells. Take time to experiment!

First, use **shape** to examine the number of rows and columns.

At the bottom, it says "Mode: Command" and "Ln 1, Col 1 3\_4-machinelearning.ipynb".

Loading the file from 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.

The screenshot shows the JupyterLab interface. On the left is a file browser for the directory `/en_us/`, listing files like `3.4-machinelearning.ipynb`, `column_2C_weka.arff`, `column_2C.dat`, etc. The main area is a code editor for `3.4-machinelearning.ipynb`. The code is as follows:

```
[9]: from sklearn.model_selection import train_test_split
train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42, stratify=df['class'])

[10]: test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])

[11]: print(train.shape)
print(test.shape)
print(validate.shape)

(248, 7)
(31, 7)
(31, 7)
```

Examine the three datasets.

```
[ ]: print(train['class'].value_counts())
print(test['class'].value_counts())
print(validate['class'].value_counts())
```

Now, check the distribution of the classes.

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

The screenshot shows the JupyterLab interface. On the left is a file browser for the directory `/en_us/`, listing files like `3.4-machinelearning.ipynb`, `column_2C_weka.arff`, `column_2C.dat`, etc. The main area is a code editor for `3.4-machinelearning.ipynb`. The code is as follows:

```
[ ]: bucket='c65030a125360513112813t1w336316976938-labbucket-1argwvxq21a18'
prefix='lab3'

train_file='vertebral_train.csv'
test_file='vertebral_test.csv'
validate_file='vertebral_validate.csv'

import os

s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False)
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).put(Body=csv_buffer.getvalue())
```

Use the function that you created to upload the three datasets.

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.

```

[15]: import boto3
from sagemaker.image_uris import retrieve
container = retrieve('xgboost', boto3.Session().region_name, '1.0-1')

[16]: hyperparams={"num_round": "42",
                 "eval_metric": "auc",
                 "objective": "binary:logistic"}

[17]: import sagemaker
s3_output_location="s3://{}(/output)".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                         sagemaker.get_execution_role(),
                                         instance_count=1,
                                         instance_type='ml.m4.xlarge',
                                         output_path=s3_output_location,
                                         hyperparameters=hyperparams,
                                         sagemaker_session=sagemaker.Session())

```

The estimator needs *channels* to feed data into the model. For training, the *train\_channel* and *validate\_channel* will be used.

```

[18]: train_channel = sagemaker.inputs.TrainingInput(
        "s3://{}(/train)".format(bucket,prefix,train_file),
        content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
        "s3://{}(/validate)".format(bucket,prefix,validate_file),
        content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}

```

Running *fit* will train the model.

**Note:** This process can take up to 5 minutes.

```

[*]: xgb_model.fit(inputs=data_channels, logs=False)

```

2022-11-14 21:24:37 Starting - Starting the training job...

After the training is complete, you are ready to test and evaluate the model. However, you will do testing and validation in later labs.

Congratulations!

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

```

[17]: import sagemaker
s3_output_location="s3://{}(/output)".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                         sagemaker.get_execution_role(),
                                         instance_count=1,
                                         instance_type='ml.m4.xlarge',
                                         output_path=s3_output_location,
                                         hyperparameters=hyperparams,
                                         sagemaker_session=sagemaker.Session())

```

The estimator needs *channels* to feed data into the model. For training, the *train\_channel* and *validate\_channel* will be used.

```

[18]: train_channel = sagemaker.inputs.TrainingInput(
        "s3://{}(/train)".format(bucket,prefix,train_file),
        content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
        "s3://{}(/validate)".format(bucket,prefix,validate_file),
        content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}

```

Running *fit* will train the model.

**Note:** This process can take up to 5 minutes.

```

[*]: xgb_model.fit(inputs=data_channels, logs=False)

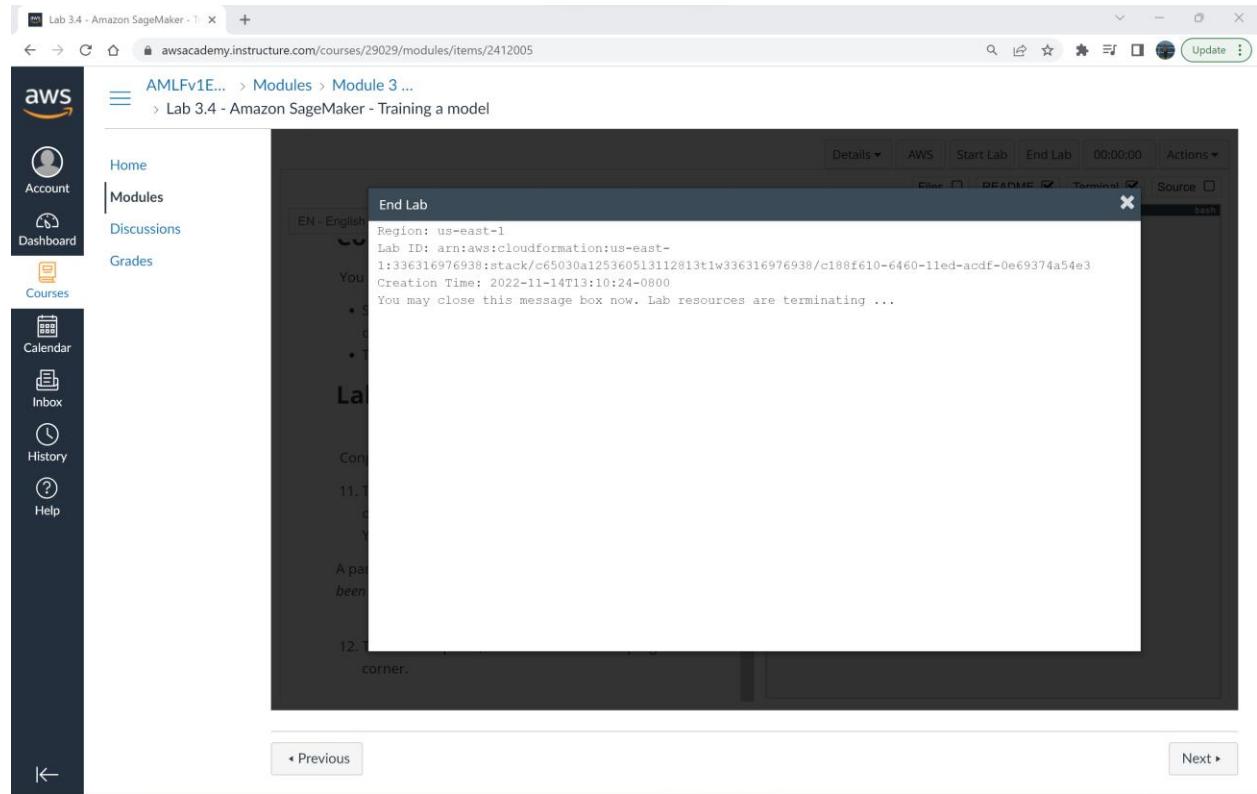
```

2022-11-14 21:24:37 Starting - Starting the training job...

After the training is complete, you are ready to test and evaluate the model. However, you will do testing and validation in later labs.

Congratulations!

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

The screenshot shows the AWS Academy interface. On the left, there's a sidebar with navigation links: Home, Modules (which is selected), Discussions, Grades, Calendar, Inbox, History, and Help. The main content area has a title "AMLFv1E... > Modules > Module 3 ... > Lab 3.5 - Amazon SageMaker - Deploying a model". A central modal window titled "Start Lab" provides session details: Region: us-east-1, Lab ID: arn:aws:cloudformation:us-east-1:193383414844:stack/c65030a125360713112916t1w193383414844/00e49140-6464-11ed-a351-0aa04cd3cc75, Creation Time: 2022-11-14T13:33:39+00:00, Start session at: 2022-11-14T13:33:40-0800, Remaining session time: 02:00:00 (120 minutes), and Lab status: ready. Below this, a message says "You now have successfully: Deployed a machine learning model". At the bottom, there are "Previous" and "Next" buttons.

The second part of the screenshot shows a JupyterLab session titled "3\_5-machinelearning.ipynb". The code cell contains the following Python script:

```

bucket='c65030a125360713112916t1w193383414844-labbucket-b2iekn15p9xs'

import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

import os
import boto3
import sagemaker
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split

f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])

class_mapper = {'Abnormal':1,'Normal':0}
df['class']=df['class'].replace(class_mapper)

cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]

train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42, stratify=df['class'])
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])

prefix='lab3'

```

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)
```

	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,:]
row.head()
```

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	88.024499	39.844669	81.774473	48.179830	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)
```

```
88.024499,39.844669,81.774473,48.179830,116.601538,56.766083
```

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

```
[9]: xgb_predictor.predict(test_row)
```

```
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)
```

	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.

**Step 3: Terminating the deployed model**

To delete the endpoint, use the `delete_endpoint` function on the predictor.

```
[11]: xgb_predictor.delete_endpoint(delete_endpoint_config=True)
```

**Step 4: Performing a batch transform**

When you are in the training-testing-feature engineering cycle, you want to test your holdout or test sets against the model. You can then use those results to calculate metrics. You could deploy an endpoint as you did earlier, but then you must remember to delete the endpoint. However, there is a more efficient way.

You can use the transformer method of the model to get a transformer object. You can then use the transform method of this object to perform a prediction on the entire test dataset. SageMaker will:

- Spin up an instance with the model
- Perform a prediction on all the input values
- Write those values to Amazon Simple Storage Service (Amazon S3)
- Finally, terminate the instance

You will start by turning your data into a CSV file that the transformer object can take as input. This time, you will use `iloc` to get all the rows, and all columns except the first column.

```
[12]: batch_X = test.iloc[:,1:];
batch_X.head()
```

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	52.204693	17.212673	78.094699	34.992020	136.972517	54.939134
130	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697

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 get all the rows, and all column except the first columns

```
[2022-11-14:22:09:29:INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14:22:09:29:INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14:22:09:29:INFO] Determined delimiter of CSV input is ','
[2022-11-14:22:09:29:INFO] POST /invocations HTTP/1.1 200 598 "-" "Go-http-client/1.1"
2022-11-14T22:09:29.729:[sagemaker logs]: MaxConcurrentTransforms=4, MaxPayloadInMB=6, BatchStrategy=MULTI_RECORD
```

After the transform completes, you can download the results from Amazon S3 and compare them with the input.

First, download the output from Amazon S3 and load it into a pandas Dataframe.

```
[15]: s3 = boto3.client('s3')
obj = s3.get_object(Bucket=Bucket, Key="{}/batch-out{}".format(prefix, batch_in.csv.out))
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()), names=['class'])
target_predicted.head(5)
```

class	
0	0.996607
1	0.777283
2	0.994641
3	0.993690
4	0.939139

You can use a function to convert the probability into either a 0 or a 1.

The first table output will be the *predicted values*, and the second table output is the *original test data*.

```
[16]: 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)
```

```
[16]: 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	0.993600	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	124.126001	-4.083298
134	1	52.204693	17.212673	78.094669	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352500	16.577364	30.706191	24.775141	113.266675	-4.497958
135	1	77.121344	30.349874	77.481083	46.7711470	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.

AMLFv1E... > Modules > Module 3 ...  
Lab 3.5 - Amazon SageMaker - Deploying a model

EN - English

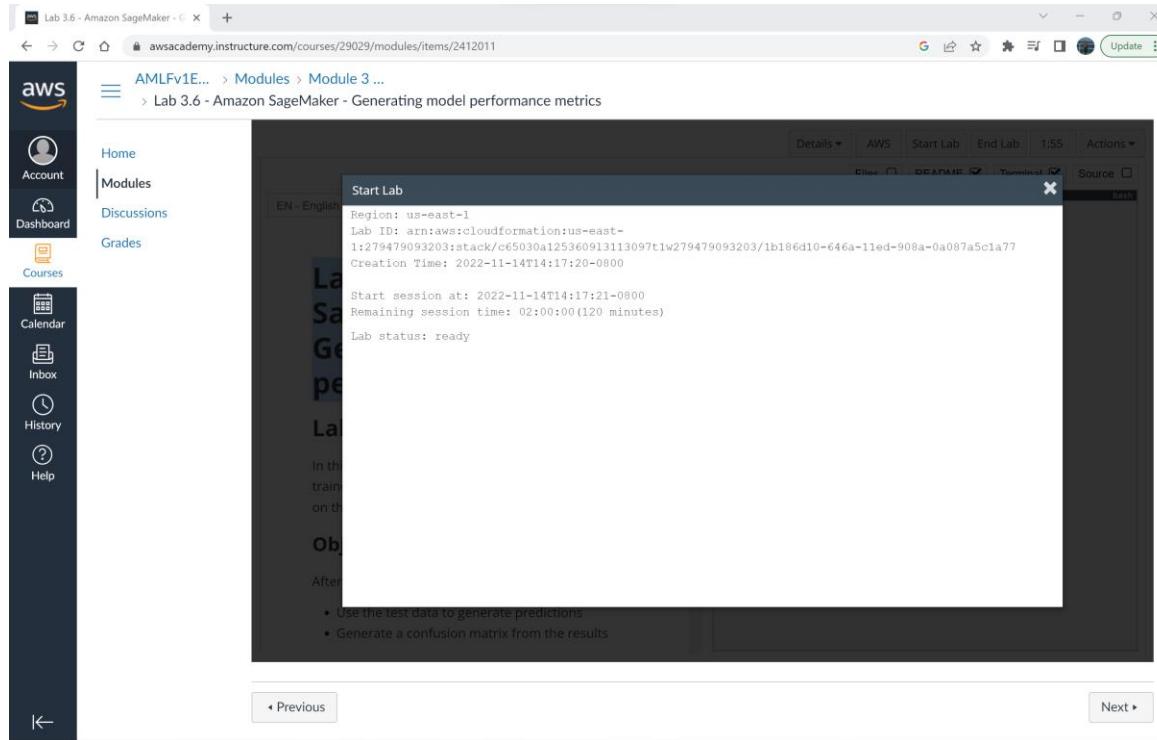
Region: us-east-1  
Lab ID: arn:aws:cloudformation:us-east-1:193303414844:stack/c65030a125360713112916t1w193383414844/00e49140-6464-11ed-a351-0aa04cd3cc75  
Creation Time: 2022-11-14T13:33:39-0800  
You may close this message box now. Lab resources are terminating ...

9. Open the `en_us/3_5-machinelearning.ipynb` file by choosing it.

◀ Previous      Next ▶

<https://awsacademy.instructure.com>

## Lab 3.6 - Amazon Sage Maker - Generating model performance metrics



```

import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

import os
import boto3
import sagemaker
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split

[*]: f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])

class_mapper = {b'Abnormal':1,b'Normal':0}
df['class']=df['class'].replace(class_mapper)

cols = df.columns.tolist()
cols = cols[1:] + cols[:1]
df = df[cols]

train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42, stratify=df['class'])
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])

prefix='lab3'

```

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.

```

2022-11-14 22:26:05 Starting - Starting the training job.....
2022-11-14 22:26:44 Starting - Preparing the instances for training.....
2022-11-14 22:29:35 Downloading - Downloading input data.....
2022-11-14 22:29:59 Training - Downloading the training image.....
2022-11-14 22:30:29 Training - Training image download completed. Training in progress.....
2022-11-14 22:30:55 Uploading - Uploading generated training model.
2022-11-14 22:31:06 Completed - Training job completed
.....
[2022-11-14:22:36:53:INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14:22:36:53:INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14:22:36:53:INFO] nginx config:
worker_processes auto;
[2022-11-14:22:36:53:INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14:22:36:53:INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14:22:36:53:INFO] nginx config:
worker_processes auto;
daemon off;
pid /tmp/nginx.pid;
error_log /dev/stderr;
worker_limit_nofile 4096;
events {
    worker_connections 2048;
}
http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    access_log /dev/stdout combined;
    upstream unicorn {
        server unix:/tmp/unicorn.sock;
    }
    server {
        listen 8080 deferred;
        client_max_body_size 0;
        keepalive_timeout 3;
        location ~ ^/(ping|invocations|execution-parameters) {
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $http_host;
            proxy_redirect off;
            proxy_read_timeout 60s;
            proxy_pass http://unicorn;
        }
        location /
}

```

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.

```

Step 1: Exploring the results

The output from the model will be a probability. You must first convert that probability into one of the two classes, either 0 or 1. To do this, you can create a function to perform the conversion. Note the use of the threshold in the function.

[4]: def binary_convert(x):
    threshold = 0.3
    if x > threshold:
        return 1
    else:
        return 0

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

print(target_predicted_binary.head(5))
test.head(5)

0    1
1    1
2    1
3    1
4    1
Name: class, dtype: int64

```

	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

Based on these results, you can see that the initial model might not be that good. It's difficult to tell by comparing a few values.

Next, you will generate some metrics to see how well the model performs.

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.

```
[5]: test_labels = test.iloc[:,0]
test_labels.head()

[5]: 136    1
230    0
134    1
130    1
47    1
Name: class, dtype: int64

[6]: from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(test_labels, target_predicted_binary)
df_confusion = pd.DataFrame(matrix, index=['Normal','Abnormal'],columns=['Normal','Abnormal'])

df_confusion
```

	Normal	Abnormal
Normal	7	3
Abnormal	2	19

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`.

```
[7]: import seaborn as sns
import matplotlib.pyplot as plt

cmap = sns.color_palette("BrBG", 10)
sns.heatmap(df_confusion, annot=True, cbar=None, cmap=cmap)
plt.title("Confusion Matrix")
plt.tight_layout()
plt.ylabel("True Class")
plt.xlabel("Predicted Class")
plt.show()
```

	Normal	Abnormal
Normal	7	3
Abnormal	2	19

**Tip:** If the chart doesn't display the first time, try running the cell again.

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: Calculating performance statistics

If you want to compare this model to the next model that you create, you need some metrics that you can record. For a binary classification problem, the confusion matrix data can be used to calculate various metrics.

To start, extract the values from the confusion matrix cells into variables.

[8]: from sklearn.metrics import roc_auc_score, roc_curve, auc
TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).ravel()

print(f"True Negative (TN) : {TN}")
print(f"False Positive (FP) : {FP}")
print(f"False Negative (FN) : {FN}")
print(f"True Positive (TP) : {TP}")

True Negative (TN) : 7
False Positive (FP) : 3
False Negative (FN) : 2
True Positive (TP) : 19

You can now calculate some statistics.

Sensitivity

Sensitivity is also known as hit rate, recall, or true positive rate (TPR). It measures the proportion of the actual positives that are correctly identified.

In this example, the sensitivity is the probability of detecting an abnormality for patients with an abnormality.
```

```

[9]: # Sensitivity, hit rate, recall, or true positive rate
Sensitivity = float(TP)/(TP+FN)*100
print(f"Sensitivity or TPR: {Sensitivity}%")
print(f"There is a {Sensitivity}% chance of detecting patients with an abnormality have an abnormality")

Sensitivity or TPR: 90.47619847619048%
There is a 90.47619847619048% chance of detecting patients with an abnormality have an abnormality

Question: Is the sensitivity good enough for this scenario?
```

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.

```

In this example, the specificity is the probability of detecting normal, for patients who are normal.
```

```

[10]: # Specificity or true negative rate
Specificity = float(TN)/(TN+FP)*100
print(f"Specificity or TNR: {Specificity}%")
print(f"There is a {Specificity}% chance of detecting normal patients are normal.")

Specificity or TNR: 70.0%
There is a 70.0% chance of detecting normal patients are normal.

Question: Is this specificity too low, exactly right, or too high? What value would you want to see here, given the scenario?
```

#### Positive and negative predictive values

The precision, or positive predictive value, is the proportion of positive results.

In this example, the positive predictive value is *the probability that subjects with a positive screening test truly have an abnormality*.

```

[11]: # Precision or positive predictive value
Precision = float(TP)/(TP+FP)*100
print(f"Precision: {Precision}%")
print(f"You have an abnormality, and the probability that is correct is {Precision}%")


Precision: 86.3683636363636%
You have an abnormality, and the probability that is correct is 86.3683636363636%

The negative predictive value is the proportion of negative results.

In this example, the negative predictive value is the probability that subjects with a negative screening test truly have an abnormality.
```

```

[12]: # Negative predictive value
NPV = float(FN)/(TN+FN)*100
print(f"Negative Predictive Value: {NPV}%")
print(f"You don't have an abnormality, but there is a {NPV}% chance that is incorrect")

Negative Predictive Value: 77.7777777777779%
You don't have an abnormality, but there is a 77.7777777777779% chance that is incorrect

Think about the impact of these values. If you were a patient, how worried should you be if the test for an abnormality was positive? On the opposite side, how reassured should you be if you tested negative?
```

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**.

The screenshot shows a JupyterLab environment with several tabs open. The main tab is titled '3.6-machinelearning.ipynb'. On the left, there's a file browser showing local files like '3.6-machinelearning.ipynb' and various 'column\_X.weka.arff' files. The right side displays code cells and their outputs.

**False positive rate**

The false positive rate (*FPR*) is the probability that a false alarm will be raised, or that a positive result will be given when the true value is negative.

```
[13]: # Fall out or false positive rate
FPR = float(FP)/(TP+FP)*100
print(f"False Positive Rate: {FPR}%")
print(f"There is a {FPR}% chance that this positive result is incorrect.")

False Positive Rate: 30.0%
There is a 30.0% chance that this positive result is incorrect.
```

**False negative rate**

The false negative rate -- or miss rate -- is the probability that a true positive will be missed by the test.

```
[14]: # False negative rate
FNR = float(FN)/(TN+FN)*100
print(f"False Negative Rate: {FNR}%")
print(f"There is a {FNR}% chance that this negative result is incorrect.")

False Negative Rate: 9.523809523809524%
There is a 9.523809523809524% chance that this negative result is incorrect.
```

**False discovery rate**

In this example, the false discovery rate is the probability of predicting an abnormality when the patient doesn't have one.

```
[15]: # False discovery rate
FDR = float(FP)/(TP+FP)*100
print(f"False Discovery Rate: {FDR}%")
print(f"You have an abnormality, but there is a {FDR}% chance this is incorrect.")

False Discovery Rate: 13.636363636363635%
You have an abnormality, but there is a 13.636363636363635% chance this is incorrect.
```

**Overall accuracy**

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%**.

Lab 3.6 - Amazon SageMaker x | Amazon SageMaker x | JupyterLab x +

mynotebook-4km6.notebook.us-east-1.sagemaker.aws/lab

File Edit View Run Kernel Git Tabs Settings Help

Launcher x 3\_6-machinelearning.ipynb x

File Edit View Insert Cell Kernel Help

conda\_python3

Name Last Modified

- 3\_6-machinelearning.ipynb 10 minutes ago
- column\_2C.weka.arff 23 minutes ago
- column\_2C.dat 23 minutes ago
- column\_3C.weka.arff 23 minutes ago
- column\_3C.dat 23 minutes ago
- PythonCheatsheet.ipynb 28 minutes ago

FDR = float(FP)/(TP+FP)\*100  
print("False Discovery Rate: {FDR}%)  
print("You have an abnormality, but there is a {FDR}% chance this is incorrect.")

False Discovery Rate: 13.636363636363635%  
You have an abnormality, but there is a 13.6363636363635% chance this is incorrect.

## Overall accuracy

How accurate is your model?

```
[16]: # Overall accuracy  
ACC = float(TP+TN)/(TP+FP+FN+TN)*100  
print("Accuracy: (ACC)%")
```

Accuracy: 83.87896774193549%

In summary, you calculated the following metrics from your model:

```
[17]: print("Sensitivity or TPR: (Sensitivity)%")  
print("Specificity or TNR: (Specificity)%")  
print("Precision: (Precision)%")  
print("Negative Predictive Value: (NPV)%")  
print("False Positive Rate: (FPR)%")  
print("False Negative Rate: (FNR)%")  
print("False Discovery Rate: {FDR}%)  
print("Accuracy: (ACC)%")
```

Sensitivity or TPR: 90.47619847619048%  
Specificity or TNR: 70.0%  
Precision: 86.36363636363636%  
Negative Predictive Value: 77.7777777777779%  
False Positive Rate: 30.0%  
False Negative Rate: 9.523809523889524%  
False Discovery Rate: 13.636363636363635%  
Accuracy: 83.87896774193549%

**Challenge task:** Record the previous values, then go back to step 1 and change the value used for the threshold. Values you should try are .25 and .75.

Did those threshold values make a difference?

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

Lab 3.6 - Amazon SageMaker - C | Amazon SageMaker | JupyterLab | +

File Edit View Run Kernel Git Tabs Settings Help

/ en\_us /

Name Last Modified

- 3\_6-machinelearning.ipynb 12 minutes ago
- column\_2C\_weka.arff 26 minutes ago
- column\_2C.dat 26 minutes ago
- column\_3C\_weka.arff 26 minutes ago
- column\_3C.dat 26 minutes ago
- PythonCheatSheet.ipynb 30 minutes ago

**Step 4: Calculating the AUC-ROC Curve**

The scikit-learn library has functions that can help you compute the *area under the receiver operating characteristic curve (AUC-ROC)*.

- The ROC is a probability curve.
- The AUC tells you how well the model can distinguish between classes.

The AUC can be calculated. As you will see in the next lab, it can be used to measure the performance of the model.

In this example, the higher the AUC, the better the model is at distinguishing between abnormal and normal patients.

Depending on the value you set for the threshold, the AUC can change. You can plot the AUC by using the probability instead of your converted class.

```
[18]: test_labels = test.iloc[:,0];
print("Validation AUC", roc_auc_score(test_labels, target_predicted))

Validation AUC 0.8904761904761904
```

Typically, the ROC curve is plotted with the TPR against the FPR, where the TPR is on the y-axis and the FPR is on the x-axis.

scikit-learn has the `roc_curve` function to help generate those values to plot.

```
[19]: fpr, tpr, thresholds = roc_curve(test_labels, target_predicted)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

# create the axis of thresholds (scores)
ax2 = plt.gca().twinx()
ax2.plot(fpr, thresholds, markeredgcolor='r', linestyle='dashed', colors='r')
ax2.set_ylabel('Threshold')
ax2.set_yticks([thresholds[-1], thresholds[0]])
ax2.set_xlim([fpr[0], fpr[-1]])

print(plt.figure())
Figure(432x288)
```

Mode: Command | Ln 1, Col 1 | 3\_6-machinelearning.ipynb

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

Lab 3.6 - Amazon SageMaker - C | Amazon SageMaker | JupyterLab | +

File Edit View Run Kernel Git Tabs Settings Help

/ en\_us /

Name Last Modified

- 3\_6-machinelearning.ipynb 13 minutes ago
- column\_2C\_weka.arff 27 minutes ago
- column\_2C.dat 27 minutes ago
- column\_3C\_weka.arff 27 minutes ago
- column\_3C.dat 27 minutes ago
- PythonCheatSheet.ipynb 32 minutes ago

```
[19]: fpr, tpr, thresholds = roc_curve(test_labels, target_predicted)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

# create the axis of thresholds (scores)
ax2 = plt.gca().twinx()
ax2.plot(fpr, thresholds, markeredgcolor='r', linestyle='dashed', colors='r')
ax2.set_ylabel('Threshold')
ax2.set_yticks([thresholds[-1], thresholds[0]])
ax2.set_xlim([fpr[0], fpr[-1]])

print(plt.figure())
Figure(432x288)
```

Mode: Command | Ln 1, Col 1 | 3\_6-machinelearning.ipynb

That's all for this lab, we completed all the step that are required for this lab sucessfully.

The screenshot shows a web browser window for AWS Academy. The URL is [awsacademy.instructure.com/courses/29029/modules/items/2412011](https://awsacademy.instructure.com/courses/29029/modules/items/2412011). The page title is "AMLFv1E... > Modules > Module 3 ... > Lab 3.6 - Amazon SageMaker - Generating model performance metrics". On the left, there's a sidebar with navigation links: Home, Modules (selected), Discussions, Grades, Courses, Calendar, Inbox, History, and Help. The main content area has a dark background with white text. A modal dialog box is open, titled "End Lab". It displays the following information:  
Region: us-east-1  
Lab ID: arn:aws:cloudformation:us-east-1:279479093203:stack/c65030a125360913113097t1w279479093203/lbl06d10-646a-11ed-900a-0a087a5c1a77  
Creation Time: 2022-11-14T14:17:20-0800  
You may close this message box now. Lab resources are terminating ...  
Below this, there's a section titled "In this lab you will learn how to generate predictions from a trained machine learning model using Amazon SageMaker. You will also learn how to evaluate the quality of the generated predictions using a confusion matrix." followed by a bulleted list:

- Use the test data to generate predictions
- Generate a confusion matrix from the results

At the bottom of the modal, there are "Previous" and "Next" buttons. The "Next" button is highlighted with a blue border.

## Lab 3.7 - Tuning Hyperparameters

Start Lab

Region: us-east-1  
Lab ID: arn:aws:cloudformation:us-east-1:896046633533:stack/c65030a125361113113269t1w896046633533/759e75e0-646f-11ed-ad93-0e9b55720b6f  
Creation Time: 2022-11-14T14:55:39-0800

Start session at: 2022-11-14T14:55:40-0800  
Remaining session time: 02:00:00 (120 minutes)

Lab status: ready

- Test the tuned model by using performance metrics

**Importing the data, and training, testing and validating the model**

By running the following cells, the data will be imported, and the model will be trained, tested and validated and ready for use.

**Note:** The following cells represent the key steps in the previous labs.

In order to tune the model it must be ready, then you can tweak the mdoel with hyperparameters later in step 2.

```
[1]: bucket='c65030a125361113113269t1w896046633533-labbucket-1n52van7atqto'

[2]: import time
start = time.time()
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

import os
import boto3
import sagemaker
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split

from sklearn.metrics import roc_auc_score, roc_curve, auc, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

**Note:** The following cell takes approximately 10 minutes to complete. Observe the code and how it processes, this will help you to better understand what is going on in the background. Keep in mind that this cell completes all the steps you did in previous labs in this module including:

- Importing the data
- Loading the data into a dataframe
- Splitting the data into training, test and validation datasets
- Uploading the split datasets to S3
- Training, testing and validating the model with the datasets

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

The screenshot shows a JupyterLab interface with a file tree on the left and a code editor on the right. The code editor contains Python code for training an XGBoost model using a batch-in, batch-out strategy. The execution log at the bottom of the code cell shows the job starting, instances preparing, input downloading, training image download completed, upload initiated, and finally completed.

```

xgb_model.fit(inputs=data_channels, logs=False)

batch_X = test.iloc[:,1:];

batch_X_file='batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)

batch_output = "s3://{}//{}//batch-out/".format(bucket,prefix)
batch_input = "s3://{}//{}//batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = xgb_model.transformer(instance_count=1,
                                       instance_type='ml.m5.2xlarge',
                                       strategy='MultiRecord',
                                       assemble_with='Line',
                                       output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                         data_type='S3Prefix',
                         content_type='text/csv',
                         split_type='Line')

xgb_transformer.wait(logs=False)

```

2022-11-14 23:03:57 Starting - Starting the training job...
2022-11-14 23:04:15 Starting - Preparing the instances for training.....
2022-11-14 23:05:57 Downloading - Downloading Input data...
2022-11-14 23:06:18 Training - Training image download completed. Training in progress..
2022-11-14 23:06:43 Uploading - Uploading generated training model..
2022-11-14 23:06:43 Completed - Training job completed
.....
[2022-11-14 23:11:07 INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14 23:11:07 INFO] No GPUs detected (normal if no gpus installed)
[2022-11-14 23:11:07 INFO] nginx config:
worker\_processes auto;
daemon off;
pid /tmp/nginx.pid;
error\_log /dev/stderr;
worker\_rlimit\_nofile 4096;
events {
 worker\_connections 2048;
}
http {
 include /etc/nginx/mime.types;
}

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

The screenshot shows a JupyterLab interface with a file tree on the left and a code editor on the right. The code editor contains Python code for reading S3 objects and applying a binary conversion function to target predicted values. It then plots a confusion matrix and an ROC curve.

### Step 1: Getting model statistics

Before you tune the model, re-familiarize yourself with the current model's metrics.

The setup performed a batch prediction, so you must read in the results from Amazon Simple Storage Service (Amazon S3).

```

s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}//batch-out/{}".format(prefix,'batch-in.csv.out'))
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),',',names=['class'])

def binary_convert(x):
    threshold = 0.5
    if x > threshold:
        return 1
    else:
        return 0

target_predicted_binary = target_predicted[['class']].apply(binary_convert)
test_labels = test.iloc[:,0]

```

Plot the confusion matrix and the receiver operating characteristic (ROC) curve for the original model.

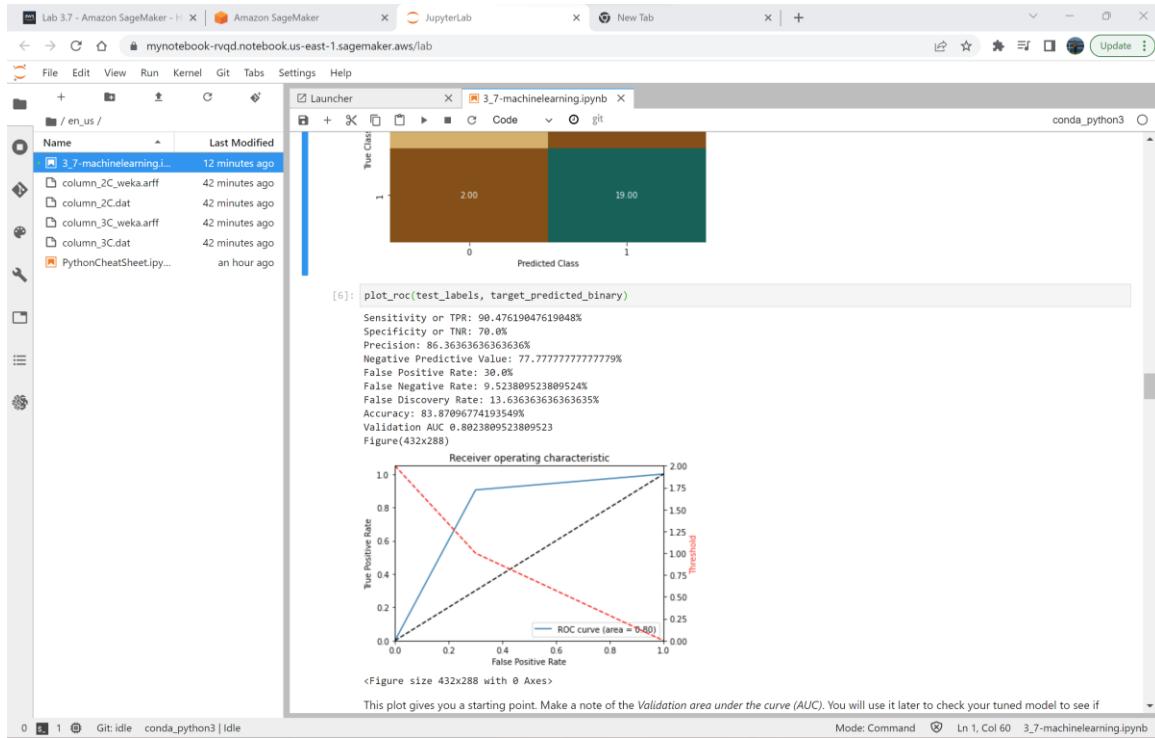
```

plot_confusion_matrix(test_labels, target_predicted_binary)

```

Confusion Matrix

	0	1
0	7.00	3.00
1	2.00	19.00



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%

File Edit View Run Kernel Git Tabs Settings Help

Name Last Modified

- 3\_7-machinelearning.ipynb 13 minutes ago
- column\_2C\_weka.arff 42 minutes ago
- column\_2C.dat 42 minutes ago
- column\_3C\_weka.arff 42 minutes ago
- column\_3C.dat 42 minutes ago
- PythonCheatsheet.ipynb... an hour ago

## 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.

```
[7]: #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://[REDACTED]/output'.format(bucket, prefix),
                                     sagemaker_session=sagemaker.Session())

xgb.set_hyperparameters(eval_metrics='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.
```

```
File Edit View Run Kernel Git Tabs Settings Help
+ - _ ○
mynotebook-rvqd.notebook.us-east-1.sagemaker.aws/lab
Launcher 3_7-machinelearning.ipynb x
Name Last Modified
3_7-machinelearning.ipynb 14 minutes ago
column_2c_weka.arff 44 minutes ago
column_2c.dat 44 minutes ago
column_3c_weka.arff 44 minutes ago
column_3c.dat 44 minutes ago
PythonCheatSheet.ipynb... an hour ago
conda_python3
Name
Last Modified
3_7-machinelearning.ipynb 14 minutes ago
column_2c_weka.arff 44 minutes ago
column_2c.dat 44 minutes ago
column_3c_weka.arff 44 minutes ago
column_3c.dat 44 minutes ago
PythonCheatSheet.ipynb... an hour ago
3_7-machinelearning.ipynb x
'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 config.
No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config.
...
!
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**.

```
[9]: from pprint import pprint
from sagemaker.analytics import HyperparameterTuningJobAnalytics

tuner_analytics = HyperparameterTuningJobAnalytics(tuner.latest_tuning_job.name, sagemaker_session=sagemaker.Session())

df_tuning_job_analytics = tuner_analytics.dataframe()

# Sort the tuning job analytics by the final metrics value
df_tuning_job_analytics.sort_values(
    by=['FinalObjectiveValue'],
    inplace=True,
    ascending=False if tuner.objective_type == "Maximize" else True)

# Show detailed analytics for the top 20 models
df_tuning_job_analytics.head(20)
```

	alpha	eta	min_child_weight	num_round	subsample	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime
0	0.000000	0.198089	4.556176	12.0	0.670806	sagemaker-xgboost-221114-2311-010-a0f86d06	Completed	0.09677	2022-11-14 23:25:05+00:00	2022-11-14 23:25:32+00:00
2	6.326859	0.159529	4.161218	46.0	0.833988	sagemaker-xgboost-221114-2311-008-89970d9	Completed	0.09677	2022-11-14 23:23:03+00:00	2022-11-14 23:23:35+00:00
1	0.000000	0.220703	3.607412	41.0	0.571210	sagemaker-xgboost-221114-2311-009-f27e03e	Completed	0.12903	2022-11-14 23:23:54+00:00	2022-11-14 23:24:26+00:00
						sagemaker-xgboost-221114-2311-010-a0f86d06			2022-11-14 23:25:44+00:00	2022-11-14 23:25:54+00:00

```
[10]: attached_tuner = HyperparameterTuningJobAnalytics(tuner.latest_tuning_job.name, sagemaker_session=sagemaker.Session())
best_training_job = attached_tuner.best_training_job

Now, you must attach to the best training job and create the model.

[11]: from sagemaker.estimator import Estimator
algo_estimator = Estimator.attach(best_training_job)

best_algo_model = algo_estimator.create_model(env={'SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT': 'text/csv'})
```

```
2022-11-14 23:25:03 Starting - Found matching resource for reuse
2022-11-14 23:25:03 Downloading - Downloading input data
2022-11-14 23:25:03 Training - Training image download completed. Training in progress.
2022-11-14 23:25:03 Uploading - Uploading generated training model
2022-11-14 23:25:03 Completed - Resource reused by training job: sagemaker-xgboost-221114-2311-010-a0f86d06
```

Then, you can use the transform method to perform a batch prediction by using your testing data. Remember that the testing data is data that the model has never seen before.

```
[12]: batch_output = "s3://{}//{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}//{}/batch-in/".format(bucket,prefix,batch_X_file)

xgb_transformer = best_algo_model.transformer(instance_count=1,
                                             instance_type='ml.m4.xlarge',
                                             strategy='MultiRecord',
                                             assemble_with='Line',
                                             output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                         data_type='S3Prefix',
                         content_type='text/csv',
                         split_type='Line')
xgb_transformer.wait(logs=False)
```

.....[2022-11-14:23:31:08:INFO] No GPUs detected (normal if no gpus installed)  
[2022-11-14:23:31:08:INFO] No GPU detected (normal if no gpus installed)  
[2022-11-14:23:31:08:INFO] nginx config:  
worker\_processes auto;

Step 3 is to Investigate the tuning job results. Here we load 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 our testing data. Since testing data is data that the model has never seen before.

Lab 3.7 - Amazon SageMaker -> Amazon SageMaker > JupyterLab > New Tab > +

File Edit View Run Kernel Git Tabs Settings Help

Launcher > 3.7-machinelearning.ipynb > conda\_python3

```
[13]: s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in.csv.out'))
best_target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),',',names=['class'])

def binary_convert(x):
    threshold = 0.5
    if x > threshold:
        return 1
    else:
        return 0

best_target_predicted_binary = best_target_predicted['class'].apply(binary_convert)
test_labels = test.loc[:,0]
```

Get the predicted target and the test labels of the model.

Plot a confusion matrix for your best\_target\_predicted and test\_labels.

[14]: plot\_confusion\_matrix(test\_labels, best\_target\_predicted\_binary)

Mode: Command > Ln 1, Col 60 3.7-machinelearning.ipynb

Lab 3.7 - Amazon SageMaker -> Amazon SageMaker > JupyterLab > New Tab > +

File Edit View Run Kernel Git Tabs Settings Help

Launcher > 3.7-machinelearning.ipynb > conda\_python3

```
[15]: plot_roc(test_labels, best_target_predicted_binary)
```

Sensitivity or TPR: 95.23809523809523%  
Specificity or TNR: 60.0%  
Precision: 83.3333333333334%  
Negative Predictive Value: 85.71428571428571%  
False Positive Rate: 40.0%  
False Negative Rate: 4.761904761904762%  
False Discovery Rate: 16.666666666666664%  
Accuracy: 83.87095774193549%  
Validation AUC: 0.7761904761904761

Figure(432x288)

Receiver operating characteristic

<Figure size 432x288 with 2 Axes>

Question: How do these results differ from the original? Are these results better or worse?

You might not always see an improvement. There are a few reasons for this result:

- The model might already be good from the initial pass (what counts as *good* is subjective).
- You don't have a large amount of data to train with.
- You are using a subset of the hyperparameter tuning ranges to save time in this lab.

Increasing the hyperparameter ranges (as recommended by the documentation) and running more than 30 jobs will typically improve the model. However, this process will take 2-3 hours to complete.

Mode: Command > Ln 1, Col 60 3.7-machinelearning.ipynb

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

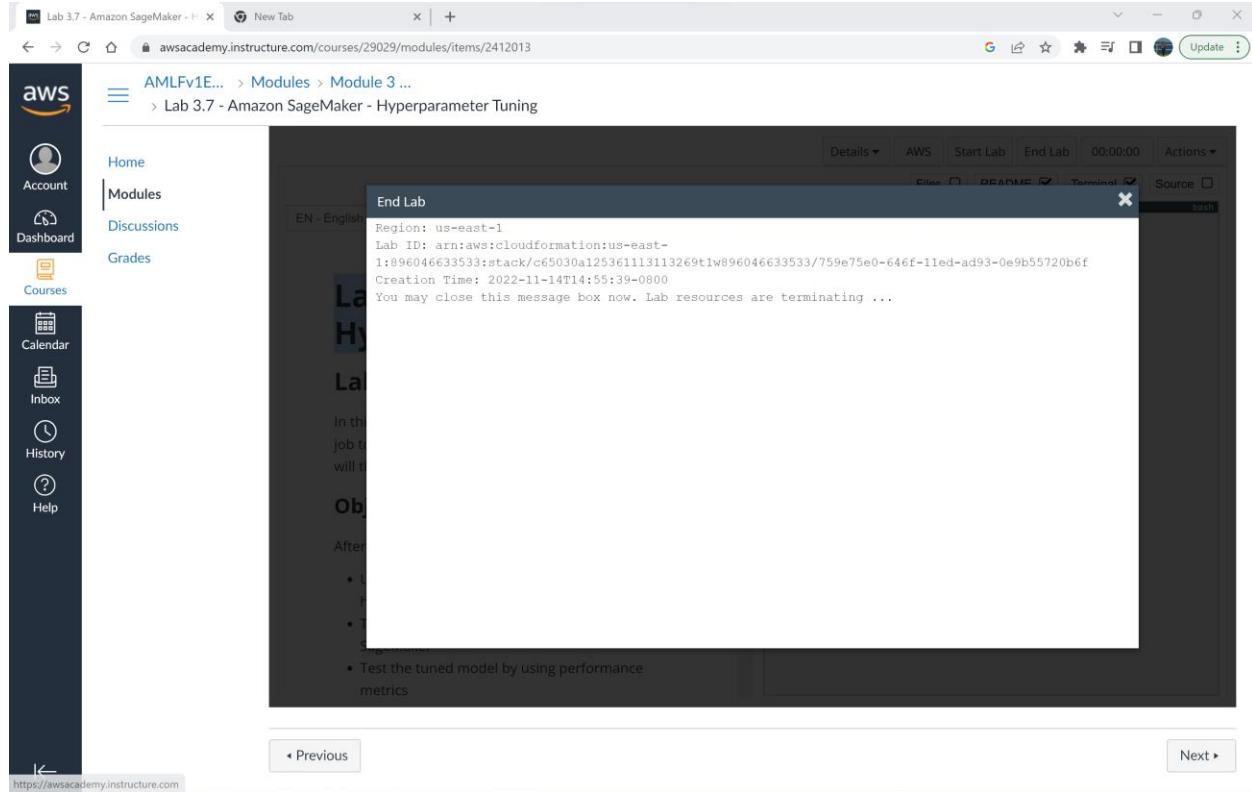
Negative Predictive Value: 85.71

**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.

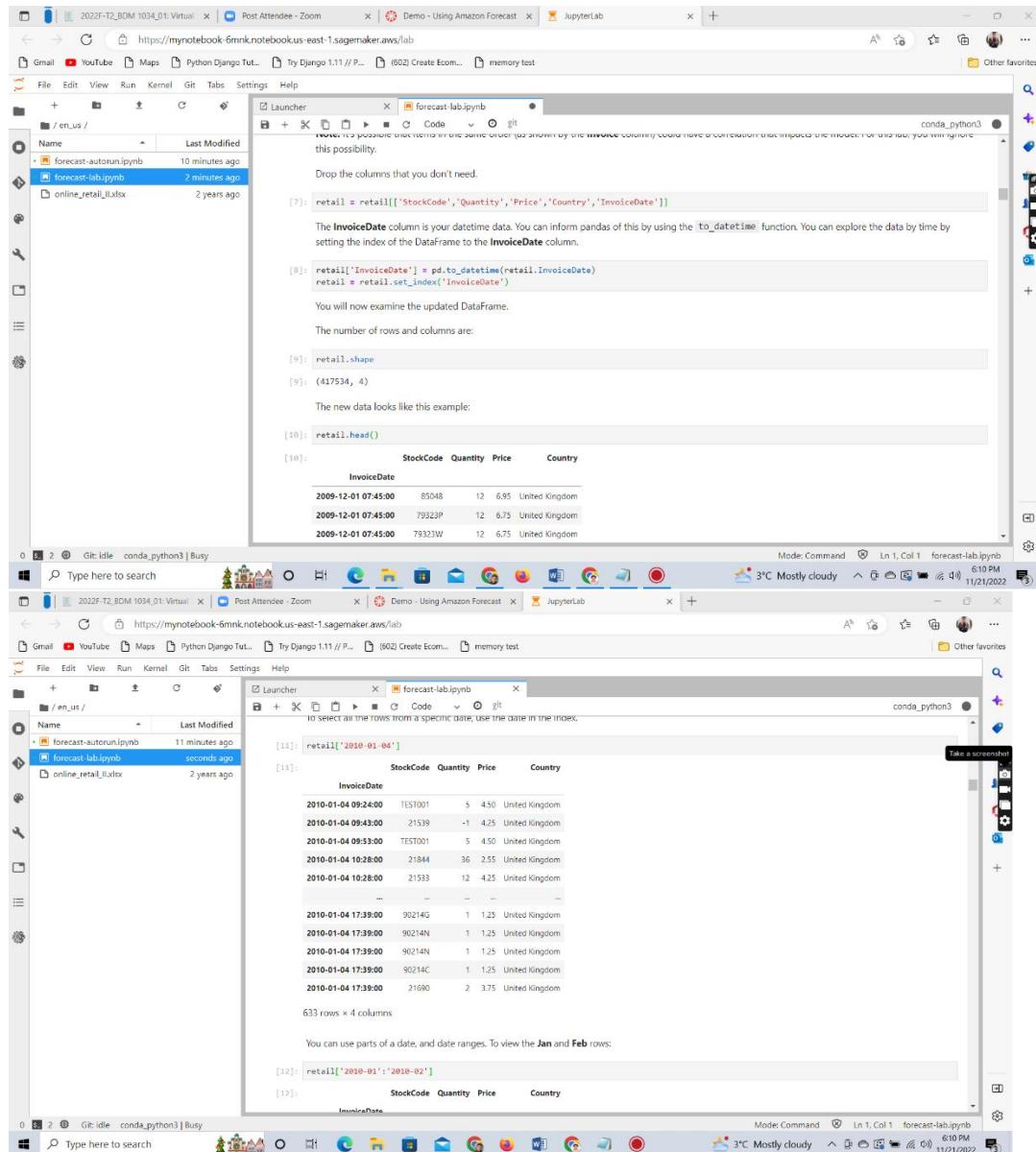


## Lab 4 - Creating a forecast with Amazon Forecast

The screenshot displays three vertically stacked JupyterLab interfaces, each showing a different step in the forecasting process:

- Top Window:** Shows the initial setup with imports for boto3, pandas, matplotlib, and xgboost. It includes a section titled "Task 2: Exploring the data". The code reads an Excel file named "online\_retail\_II.xlsx" using pd.read\_excel. A note states that some values are missing and suggests removing them.
- Middle Window:** Shows the exploration phase. The code prints the data types of the DataFrame "retail" using retail.dtypes. The output shows various columns like Invoice, StockCode, Description, Quantity, InvoiceDate, Price, CustomerID, and Country, all with object dtype except for Quantity and InvoiceDate which are int64 and datetime64 respectively.
- Bottom Window:** Shows the first few rows of the dataset using retail.head(20). The resulting table has 8 columns: Invoice, StockCode, Description, Quantity, InvoiceDate, Price, CustomerID, and Country. The data includes items like "15CM CHRISTMAS GLASS BALL 20 LIGHTS", "PINK CHERRY LIGHTS", and "RECORD FRAME 7" SINGLE SIZE".

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, Descriprion, Quantity, InvoiceDate, Price, CustomerID and Country,



The screenshot shows a JupyterLab interface with two tabs open: 'forecast-lab.ipynb' and 'memory test'. The 'forecast-lab.ipynb' tab contains Python code and its corresponding output.

```
[7]: retail = retail[['StockCode','Quantity','Price','Country','InvoiceDate']]
The InvoiceDate column is your datetime data. You can inform pandas of this by using the to_datetime function. You can explore the data by time by setting the index of the Dataframe to the InvoiceDate column.

[8]: retail['InvoiceDate'] = pd.to_datetime(retail.InvoiceDate)
retail = retail.set_index('InvoiceDate')

You will now examine the updated DataFrame.

The number of rows and columns are:
```

```
[9]: retail.shape
[9]: (417534, 4)

The new data looks like this example:
```

```
[10]: retail.head()
[10]:
   StockCode Quantity Price Country
InvoiceDate
2009-12-01 07:45:00 85048    12  6.95 United Kingdom
2009-12-01 07:45:00 79323P   12  6.75 United Kingdom
2009-12-01 07:45:00 79323W   12  6.75 United Kingdom
```

The second screenshot shows a continuation of the session with more code execution and data preview.

```
[11]: retail["2010-01-04"]
[11]:
   StockCode Quantity Price Country
InvoiceDate
2010-01-04 09:24:00 TEST001     5  450 United Kingdom
2010-01-04 09:43:00 21539    -1  425 United Kingdom
2010-01-04 09:53:00 TEST001     5  450 United Kingdom
2010-01-04 10:28:00 21844     36  2.55 United Kingdom
2010-01-04 10:28:00 21533    12  425 United Kingdom
...
2010-01-04 17:39:00 90214G     1  1.25 United Kingdom
2010-01-04 17:39:00 90214N     1  1.25 United Kingdom
2010-01-04 17:39:00 90214N     1  1.25 United Kingdom
2010-01-04 17:39:00 90214C     1  1.25 United Kingdom
2010-01-04 17:39:00 21690     2  3.75 United Kingdom
```

Output: 633 rows × 4 columns

```
You can use parts of a date, and date ranges. To view the Jan and Feb rows:
```

```
[12]: retail["2010-01":"2010-02"]
[12]:
   StockCode Quantity Price Country
InvoiceDate
```

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.

Screenshot of a JupyterLab session showing code execution and a bar chart.

```

[13]: retail.index.min()
[13]: Timestamp('2009-12-01 07:45:00')

The date range ends at:

[14]: retail.index.max()
[14]: Timestamp('2010-12-09 20:01:00')

With pandas, you can extract date information easily. You might extract date information to explore the data further and look for time-related trends.

Extract the year, month, and day of the week.

[15]: retail['Year'] = retail.index.year
retail['Month'] = retail.index.month
retail['weekday_name'] = retail.index.day_name()

[16]: retail.head()

```

	StockCode	Quantity	Price	Country	Year	Month	weekday_name
2009-12-01 07:45:00	85048	12	6.95	United Kingdom	2009	12	Tuesday
2009-12-01 07:45:00	79323P	12	6.75	United Kingdom	2009	12	Tuesday
2009-12-01 07:45:00	79323W	12	6.75	United Kingdom	2009	12	Tuesday
2009-12-01 07:45:00	22041	48	2.10	United Kingdom	2009	12	Tuesday
2009-12-01 07:45:00	21232	24	1.25	United Kingdom	2009	12	Tuesday

Mode: Command | Ln 1, Col 1 | forecast-lab.ipynb | 6:11 PM | 11/21/2022

Type here to search

Here to search

2022F-T2\_BDM 2053\_01: Virtual | Launch Meeting - Zoom | Post Attendee - Zoom | Demo - Using Amazon Forecast | JupyterLab | +

File Edit View Run Kernel Git Tabs Settings Help

Name Last Modified

- forecast-autorun.ipynb 12 minutes ago
- forecast\_lab.ipynb** a minute ago
- online\_retail\_II.xlsx 2 years ago

Launcher | forecast-lab.ipynb | conda\_python3 | Take a screenshot

```

[17]: retail.Month.value_counts(sort=False).plot(kind='bar')
[17]: <AxesSubplot: >

```

seasonality in this data. You will now investigate whether there is seasonality.

From the chart, you could deduce some seasonality:

1. November and December seem to be higher than the rest of the year.
2. Q4 seems to be higher than other quarters.
3. For Q1, Q2, and Q3: The last month of the quarter (months 3, 6, and 9) seem to have spikes.

Do you notice any other seasonal patterns?

Now, investigate whether there is any seasonality during the week.

Mode: Command | Ln 1, Col 1 | forecast-lab.ipynb | 6:12 PM | 11/21/2022

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.

The screenshot shows a JupyterLab environment with multiple tabs open at the top, including "2022f-T2\_BDM 2053\_01: Virtual", "Launch Meeting - Zoom", "Post Attendee - Zoom", "Demo - Using Amazon Forecast", and "JupyterLab". The main area displays a code cell and its output. The code cell contains Python code to sort a dataset by day of the week and plot it as a bar chart:

```
day_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
retail.weekday_name.value_counts(sort=False).loc[day_order].plot(kind='bar')
```

The resulting bar chart shows the number of retail orders for each day of the week. The y-axis ranges from 0 to 80,000. The x-axis lists the days: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday. The bars show values approximately: Monday (~65,000), Tuesday (~70,000), Wednesday (~68,000), Thursday (~75,000), Friday (~55,000), Saturday (~70,000), and Sunday (~72,000).

Below the chart, a note states: "Saturday shows very few orders. Why might this be the case?"

**Task 3: Cleaning and reducing the size of the data**

In this task, you will reduce the size of the data. You will also remove any anomalies, such as negative prices, outliers, and country data.

Reducing the countries

Mode: Command 3°C 6:12 PM 11/21/2022

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

Examine the `Country` data:

```
[19]: retail.Country.unique()
[19]: array(['United Kingdom', 'France', 'USA', 'Belgium', 'Australia', 'EIRE',
   'Germany', 'Portugal', 'Japan', 'Denmark', 'Netherlands', 'Poland',
   'Spain', 'Channel Islands', 'Italy', 'Cyprus', 'Greece', 'Norway',
   'Austria', 'Sweden', 'United Arab Emirates', 'Finland',
   'Switzerland', 'Unspecified', 'Nigeria', 'Malta', 'RSA',
   'Singapore', 'Bahrain', 'Thailand', 'Israel', 'Lithuania',
   'West Indies', 'Korea', 'Brazil', 'Canada', 'Iceland'],
  dtype='object')

[20]: retail.Country.value_counts()
[20]: United Kingdom    379423
      EIRE              8710
      Germany            8129
      France             5710
      Netherlands        2779
      Spain              1278
      Switzerland         1187
      Belgium             1054
      Portugal            1024
      Channel Islands     906
      Sweden              883
      Italy               731
      Australia            654
      Cyprus              554
      Austria              537
      Greece              517
      Denmark              428
      Norway              369
      Finland              354
      United Arab Emirates 318
      Unspecified          288
```

Most of the data seems to be for the United Kingdom. To make your job easier, filter the data by `United Kingdom`.

```
[21]: country_filter = ['United Kingdom']
      retail = retail[retail.Country.isin(country_filter)]
```

Because the `Country` column only contains the same value, you can drop it.

```
[22]: retail = retail[['StockCode', 'Quantity', 'Price']]

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

```
[23]:
```

	StockCode	Quantity	Price
0	InvoiceDate		
1	2009-12-01 07:45:00	85048	12 6.95
2	2009-12-01 07:45:00	79323P	12 6.75
3	2009-12-01 07:45:00	79323W	12 6.75
4	2009-12-01 07:45:00	22041	48 2.10
5	2009-12-01 07:45:00	21232	24 1.25

Examining StockCode and removing anomalies

Examine the distribution of the `StockCode` column:

```
[24]: retail.StockCode.describe()
[24]: count    379423
      unique   4015
      top     65123A
```

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

The screenshot shows a JupyterLab interface with two windows side-by-side. Both windows have the title 'forecast-lab.ipynb' and are running in a 'conda\_python3' kernel.

**Left Window:**

- File menu: File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help.
- Launcher pane: Shows files like 'forecast-autorun.ipynb', 'forecast-lab.ipynb', and 'online\_retail\_II.xlsx'.
- Code cell [25]: `retail.StockCode.value_counts().plot()`. The output is a histogram showing a long tail of high-frequency stock codes. The x-axis lists stock codes such as 85123A, 21137, 21011, 21808, 22765, 21374, 84464, 84493, 85218. The y-axis ranges from 0 to 3000, with the highest frequency around 3000.
- Text cell [26]:

There are 4,015 unique values for **StockCode**. A quick plot of the counts might give you some insight into how the values are distributed.

It seems that there are a few high-selling products, with a long tail behind them. You could investigate this situation further. However, for now, examine **Quantity**.
- Code cell [26]: `retail.Quantity.describe()`. The output shows descriptive statistics for the 'Quantity' column.

	count	mean	std	min	25%	50%	75%
count	379423.000000	11.451517	68.370769	-9360.000000	2.000000	4.000000	12.000000

**Right Window:**

- File menu: File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help.
- Launcher pane: Shows files like 'forecast-autorun.ipynb', 'forecast-lab.ipynb', and 'online\_retail\_II.xlsx'.
- Code cell [28]: `retail = retail[retail.Quantity>0]`. The output indicates that negative and zero quantities could impact the forecast.
- Text cell [29]:

Negative and zero quantities could impact the forecast if you don't know why these values exist. To make things easier for now, you will remove negative and zero quantities.

Now, examine **Price**.
- Code cell [29]: `retail.Price.describe()`. The output shows descriptive statistics for the 'Price' column.

	count	mean	std	min	25%	50%	75%	max
count	370551.000000	3.145220	30.551482	0.000000	1.290000	1.950000	3.750000	10953.500000

- Code cell [30]: `retail.Price.plot()`. The output is a histogram of prices, showing several sharp peaks at specific price levels.

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.

The StockCode value of M looks unusual. If you had access to a domain expert, you could learn about the importance of M. Because you can't ask a domain expert for this lab, you will drop everything that has a StockCode value of M.

```
[31]: retail[retail.Price>500].head()
```

	StockCode	Quantity	Price
2009-12-10 11:50:00	M	1	1213.02
2010-01-29 11:04:00	M	1	8985.60
2010-03-23 15:22:00	M	1	10953.50
2010-06-08 16:39:00	M	1	849.45
2010-06-11 15:54:00	M	1	1000.63

This result is better, but the max value is still high. You will now investigate this situation further.

```
[32]: retail = retail[retail.StockCode!="M"]
```

```
[33]: retail.Price.describe()
```

```
[34]: retail[retail.Price>300].head(20)
```

	StockCode	Quantity	Price
2010-01-26 16:29:00	ADJUST	1	342.80
2010-01-26 17:28:00	ADJUST	1	387.54
2010-06-25 14:15:00	ADJUST2	1	300.13
2010-06-25 14:15:00	ADJUST2	1	358.47
2010-08-04 11:38:00	POST	1	334.88

It seems that some adjustments occurred. You will also drop any data that shows these adjustments.

```
[35]: stockcodes = ['ADJUST', 'ADJUST2', 'POST']
retail = retail[~retail.StockCode.isin(stockcodes)]
```

```
[36]: retail.Price.describe()
```

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.

```
[37]: retail[retail['Price'] > 0].count()
[37]: <bound method DataFrame.count of
      InvoiceDate          StockCode  Quantity  Price
      2009-12-02 13:13:49:00     22076       12    0.0
      2009-12-03 11:19:00       48185       2    0.0
      2009-12-08 15:25:00       22065       1    0.0
      2009-12-08 15:25:00       22142       12    0.0
      2009-12-15 13:49:00       85042       8    0.0
      2009-12-18 14:22:00       21143       12    0.0
      2010-01-08 14:54:00       79320       24    0.0
      2010-01-11 12:43:00       21533       12    0.0
      2010-01-11 12:43:00       21533        5    0.0
      2010-03-12 15:47:00       TES7091       5    0.0
      2010-03-04 11:44:00       21652       1    0.0
      2010-04-01 17:13:00       22459       8    0.0
      2010-04-01 17:13:00       22458       8    0.0
      2010-06-11 11:12:00       21765       1    0.0
      2010-06-17 10:12:00       20914       2    0.0
      2010-06-24 12:34:00       22423       5    0.0
      2010-07-19 13:13:00       22698       6    0.0
      2010-09-27 16:59:00       46000M       648    0.0
      2010-09-30 12:19:00       22218       2    0.0
      2010-10-18 15:13:00       22121       1    0.0
      2010-11-07 14:26:00       21843       2    0.0>
There aren't many values in these results, so you can drop zero-priced items.

[38]: retail = retail[retail['Price'] > 0]
```

**Splitting the data**

The timeseries data that you need to create a forecast requires a `timestamp`, an `item id`, and a `demand`. These features will map to the `InvoiceDate`.

```
[39]: df_time_series = retail[['StockCode','Quantity']]
df_related_time_series = retail[['StockCode','Price']]

[40]: df_time_series[df_time_series['StockCode'] == 21232]['2009-12-01']
```

**Downsampling**

You will now examine a single item.

```
[40]: <class 'pandas.core.frame.DataFrame'>
```

InvoiceDate	StockCode	Quantity
2009-12-01 07:45:00	21232	24
2009-12-01 10:49:00	21232	48
2009-12-01 12:13:00	21232	3
2009-12-01 12:14:00	21232	20
2009-12-01 13:33:00	21232	4
2009-12-01 13:37:00	21232	12
2009-12-01 13:43:00	21232	24
2009-12-01 14:19:00	21232	12
2009-12-01 15:26:00	21232	12
2009-12-01 16:18:00	21232	12

You can see multiple orders for each day. You want to create a forecast that predicts demand at a daily level.  
You must downsample the data from the individual orders into a daily total.

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.

The screenshot displays a JupyterLab environment with two code cells and their resulting data frames.

**Code Cell 1:**

```
[41]: df_time_series = df_time_series.groupby('StockCode').resample('D').sum().reset_index()
[42]: df_time_series['InvoiceDate'] = pd.to_datetime(df_time_series.InvoiceDate)
df_time_series = df_time_series.set_index(['InvoiceDate'])
df_time_series.head()
```

**Data Output 1:**

InvoiceDate	StockCode	Quantity
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

```
[43]: df_time_series = df_time_series.groupby('StockCode').resample('D').sum().reset_index().set_index(['InvoiceDate'])
Examine the new DataFrame.
```

```
[44]: df_time_series[df_time_series.StockCode==21232]
```

**Data Output 2:**

InvoiceDate	StockCode	Quantity
2009-12-01	21232	171
2009-12-02	21232	164
2009-12-03	21232	192

**Code Cell 2:**

```
[45]: df_related_time_series.head()
```

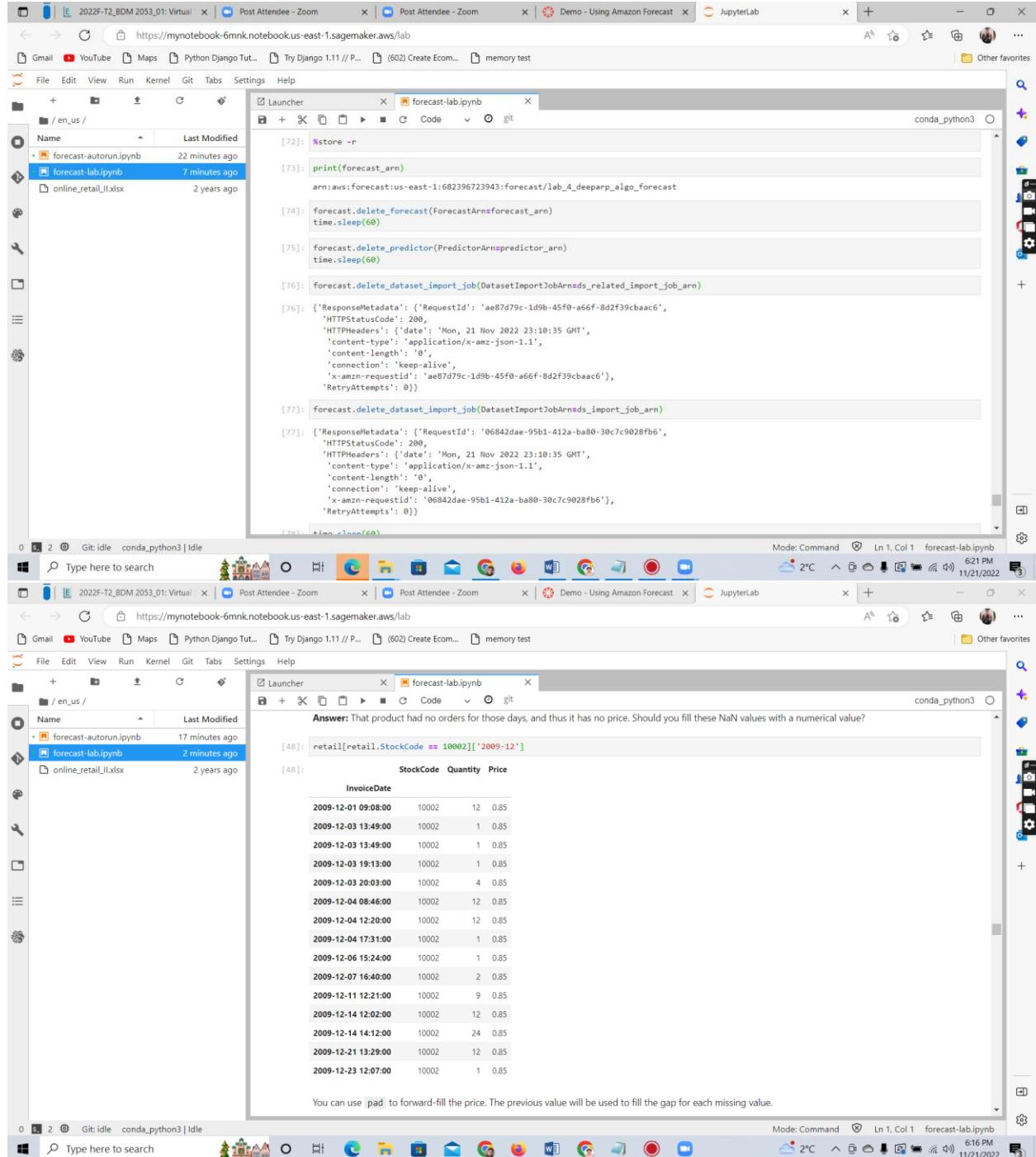
**Data Output 3:**

InvoiceDate	StockCode	Price
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

```
[46]: df_related_time_series2 = df_related_time_series.groupby('StockCode').resample('D').mean().reset_index().set_index(['InvoiceDate','StockCode'])
[47]: df_related_time_series2.head(20)
```

**Data Output 4:**

InvoiceDate	StockCode	Price
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



The screenshot shows a JupyterLab interface with multiple tabs open. The active tab is 'forecast-lab.ipynb' in the 'forecast-lab' workspace. The code cell at the bottom contains:

```
[72]: !store -r
[73]: print(forecast_arn)
arn:aws:forecast:us-east-1:682396723943:forecast/lab_4_deeparp_algo_forecast
[74]: forecast.delete_forecast(ForecastArns=forecast_arn)
time.sleep(60)
[75]: forecast.delete_predictor(PredictorArns=predictor_arn)
time.sleep(60)
[76]: forecast.delete_dataset_import_job(DataSetImportJobArns=ds_related_import_job_arn)
[77]: {'ResponseMetadata': {'RequestId': 'ae87d79c-1d9b-45f0-a66f-8d2f39cbaac6',
   'HTTPStatusCode': 200,
   'HTTPHeaders': {'date': 'Mon, 21 Nov 2022 23:10:35 GMT',
      'content-type': 'application/x-amz-json-1.1',
      'content-length': '0',
      'connection': 'keep-alive',
      'x-amzn-requestid': 'ae87d79c-1d9b-45f0-a66f-8d2f39cbaac6'},
   'RetryAttempts': 0}}
[78]: forecast.delete_dataset_import_job(DataSetImportJobArns=ds_import_job_arn)
[79]: {'ResponseMetadata': {'RequestId': '06842dae-95b1-412a-ba80-30c7c9028fb6',
   'HTTPStatusCode': 200,
   'HTTPHeaders': {'date': 'Mon, 21 Nov 2022 23:10:35 GMT',
      'content-type': 'application/x-amz-json-1.1',
      'content-length': '0',
      'connection': 'keep-alive',
      'x-amzn-requestid': '06842dae-95b1-412a-ba80-30c7c9028fb6'},
   'RetryAttempts': 0}}
```

The code cell above the data frame displays the following message:

Answer: That product had no orders for those days, and thus it has no price. Should you fill these NaN values with a numerical value?

The data frame 'retail' is shown with the following content:

	StockCode	Quantity	Price
2009-12-01 09:08:00	10002	12	0.85
2009-12-01 13:49:00	10002	1	0.85
2009-12-03 13:49:00	10002	1	0.85
2009-12-03 19:13:00	10002	1	0.85
2009-12-03 20:03:00	10002	4	0.85
2009-12-04 08:46:00	10002	12	0.85
2009-12-04 12:20:00	10002	12	0.85
2009-12-04 17:31:00	10002	1	0.85
2009-12-05 15:24:00	10002	1	0.85
2009-12-07 16:00:00	10002	2	0.85
2009-12-11 12:21:00	10002	9	0.85
2009-12-14 12:02:00	10002	12	0.85
2009-12-21 13:29:00	10002	24	0.85
2009-12-23 12:07:00	10002	12	0.85
2009-12-23 12:07:00	10002	1	0.85

A note below the table states: You can use `pad` to forward-fill the price. The previous value will be used to fill the gap for each missing value.

Here we fill all the products that have 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.

The screenshot shows a JupyterLab environment with a file browser on the left and a code editor on the right. The code editor displays the following code and output:

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

Output:

Price		
InvoiceDate	StockCode	
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
2009-12-15	10002	0.85

Below this, another code cell is shown:

```
[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.

```

[52]: buckets='mlf-lab4-Forecastbucket-12sb9sjex9iv'
session = boto3.Session()
forecast = session.client(service_name='forecast')
forecast_query = session.client(service_name='forecastquery')

You will read the variables from the store, and check whether the forecast was defined. After the forecast is defined, you will wait until its status becomes active.

[53]: print('Waiting for the predictor arn to be available')
while True:
    if 'store' in locals():
        if 'forecast_arn' in locals():
            break
    print('.')
    time.sleep(10)

print('Waiting for the predictor to be available')
status_indicator_predictor = StatusIndicator()
while True:
    status = forecast.describe_predictor(PredictorArn=predictor_arn)['Status']
    status_indicator_predictor.update(status)
    if status in ('ACTIVE', 'CREATE_FAILED'): break
    time.sleep(10)

status_indicator_predictor.end()

print('Waiting for forecast to be available')
status_indicator_forecast = StatusIndicator()
while True:
    status = forecast.describe_forecast(ForecastArn=forecast_arn)['Status']
    status_indicator_forecast.update(status)
    if status in ('ACTIVE', 'CREATE_FAILED'): break
    time.sleep(10)

```

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

```

At this point, there should be a forecast that's ready to be queried.

Check that you get data for the following test stock code: 21232

[54]: print()
forecast_response = forecast_query.query_forecast(
    ForecastArn=forecast_arn,
    Filters=[{"Item_Id": "21232"}]
)
print(forecast_response)

{
    "Forecast": {
        "Predictions": [
            {
                "Timestamp": "2010-11-01T00:00:00",
                "Value": -7.5065078735,
                "Time": "2010-11-02T00:00:00"
            },
            {
                "Timestamp": "2010-11-03T00:00:00",
                "Value": 47.7141838074,
                "Time": "2010-11-04T00:00:00"
            },
            {
                "Timestamp": "2010-11-05T00:00:00",
                "Value": -12.5096263885,
                "Time": "2010-11-06T00:00:00"
            },
            {
                "Timestamp": "2010-11-07T00:00:00",
                "Value": 3.8610399246,
                "Time": "2010-11-08T00:00:00"
            },
            {
                "Timestamp": "2010-11-09T00:00:00",
                "Value": 69.0687332153,
                "Time": "2010-11-09T00:00:00"
            },
            {
                "Timestamp": "2010-11-10T00:00:00",
                "Value": 132.3821716309,
                "Time": "2010-11-10T00:00:00"
            },
            {
                "Timestamp": "2010-11-11T00:00:00",
                "Value": 164.4648132324,
                "Time": "2010-11-11T00:00:00"
            },
            {
                "Timestamp": "2010-11-12T00:00:00",
                "Value": 161.2157897949,
                "Time": "2010-11-12T00:00:00"
            },
            {
                "Timestamp": "2010-11-13T00:00:00",
                "Value": 90.8256301883,
                "Time": "2010-11-13T00:00:00"
            },
            {
                "Timestamp": "2010-11-14T00:00:00",
                "Value": 73.7878646851,
                "Time": "2010-11-14T00:00:00"
            },
            {
                "Timestamp": "2010-11-15T00:00:00",
                "Value": 93.5387006836,
                "Time": "2010-11-15T00:00:00"
            },
            {
                "Timestamp": "2010-11-16T00:00:00",
                "Value": 141.2473449707,
                "Time": "2010-11-16T00:00:00"
            },
            {
                "Timestamp": "2010-11-17T00:00:00",
                "Value": 131.2735137393,
                "Time": "2010-11-17T00:00:00"
            },
            {
                "Timestamp": "2010-11-18T00:00:00",
                "Value": 115.9074554443,
                "Time": "2010-11-18T00:00:00"
            },
            {
                "Timestamp": "2010-11-19T00:00:00",
                "Value": 81.2667999268,
                "Time": "2010-11-19T00:00:00"
            },
            {
                "Timestamp": "2010-11-20T00:00:00",
                "Value": 68.2911911011,
                "Time": "2010-11-20T00:00:00"
            },
            {
                "Timestamp": "2010-11-21T00:00:00",
                "Value": 37.4108085632,
                "Time": "2010-11-21T00:00:00"
            },
            {
                "Timestamp": "2010-11-22T00:00:00",
                "Value": 50.3027954102,
                "Time": "2010-11-22T00:00:00"
            },
            {
                "Timestamp": "2010-11-23T00:00:00",
                "Value": 66.2919192139,
                "Time": "2010-11-23T00:00:00"
            },
            {
                "Timestamp": "2010-11-24T00:00:00",
                "Value": 56.5740509933,
                "Time": "2010-11-24T00:00:00"
            },
            {
                "Timestamp": "2010-11-25T00:00:00",
                "Value": 59.0319404602,
                "Time": "2010-11-25T00:00:00"
            },
            {
                "Timestamp": "2010-11-26T00:00:00",
                "Value": 43.9337844849,
                "Time": "2010-11-26T00:00:00"
            },
            {
                "Timestamp": "2010-11-27T00:00:00",
                "Value": -16.284259961,
                "Time": "2010-11-27T00:00:00"
            },
            {
                "Timestamp": "2010-11-28T00:00:00",
                "Value": 46.3264732361,
                "Time": "2010-11-28T00:00:00"
            },
            {
                "Timestamp": "2010-11-29T00:00:00",
                "Value": 68.4265594482,
                "Time": "2010-11-29T00:00:00"
            },
            {
                "Timestamp": "2010-11-30T00:00:00",
                "Value": 69.2885023193,
                "Time": "2010-11-30T00:00:00"
            },
            {
                "Timestamp": "2010-11-31T00:00:00",
                "Value": 62.0142593384,
                "Time": "2010-11-31T00:00:00"
            },
            {
                "Timestamp": "2010-12-01T00:00:00",
                "Value": 57.7246742249,
                "Time": "2010-12-01T00:00:00"
            },
            {
                "Timestamp": "2010-12-02T00:00:00",
                "Value": 72.5650634766,
                "Time": "2010-12-02T00:00:00"
            },
            {
                "Timestamp": "2010-12-03T00:00:00",
                "Value": 92.651693726,
                "Time": "2010-12-03T00:00:00"
            },
            {
                "Timestamp": "2010-12-04T00:00:00",
                "Value": 65.4985046587,
                "Time": "2010-12-04T00:00:00"
            },
            {
                "Timestamp": "2010-12-05T00:00:00",
                "Value": 47.3432312012,
                "Time": "2010-12-05T00:00:00"
            }
        ]
    }
}

```

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

The screenshot shows a JupyterLab environment with two code cells displayed in a notebook named "forecast-lab.ipynb".

**Plotting the actual results:**

```
[55]: actual1_df = pd.read_csv(test, names=['InvoiceDate','StockCode','Quantity'])
actual1_df['InvoiceDate'] = pd.to_datetime(actual1_df['InvoiceDate'])
actual1_df = actual1_df.set_index('InvoiceDate')
actual1_df.head()
```

StockCode	Quantity
21232	0
21232	60
21232	130
21232	255
21232	24

Check that you only have data for the 21232 stock code.

```
[56]: stockcode_filter = ['21232']
actual1_df = actual1_df[actual1_df['StockCode'].isin(stockcode_filter)]
```

```
[57]: actual1_df.head()
```

StockCode	Quantity
21232	0

**Plotting the prediction:**

```
[63]: # Generate DF
prediction_df_p10 = pd.DataFrame.From_dict(forecast_response['Forecast']['Predictions']['p10'])
prediction_df_p10.head()
```

Timestamp	Value
0	-9.757092
1	29.454016
2	36.668076
3	40.201233
4	-20.331472

Next, plot the P10 predictions.

```
[64]: # Plot
prediction_df_p10.plot()
```

A line plot titled "Value" showing three distinct peaks at different time points. The x-axis represents time, and the y-axis represents the value, ranging from approximately 40 to 60.

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.

The screenshot shows a JupyterLab environment with two tabs open: 'forecast-lab.ipynb' and 'JupyterLab'. The left sidebar lists files: 'forecast-autonm.ipynb', 'forecast-lab.ipynb' (selected), and 'online\_retail\_1.xlsx'. The right pane contains code cells:

```
[65]: prediction_df_p50 = pd.DataFrame.from_dict(forecast_response['Forecast']['Predictions'][1]['p50'])
prediction_df_p90 = pd.DataFrame.from_dict(forecast_response['Forecast']['Predictions'][1]['p90'])

Comparing the prediction to actual results

After you obtain the DataFrames, the next task is to plot them together to determine the best fit.

[66]: # Start by creating a DataFrame to house the content. Here, Source will be which DataFrame it came from.
results_df = pd.DataFrame(columns=['timestamp','value','source'])

results_df.head()

[67]: timestamp value Source

Import the observed values into the DataFrame:

[68]: import datetutil.parser
for index, row in actual_df.iterrows():
    clean_timestamp = datetutil.parser.parse(index)
    results_df = results_df.append({'timestamp': index , 'value' : row['Quantity'], 'Source': 'Actual'} , ignore_index=True)

# To show the new DataFrame
results_df.head()

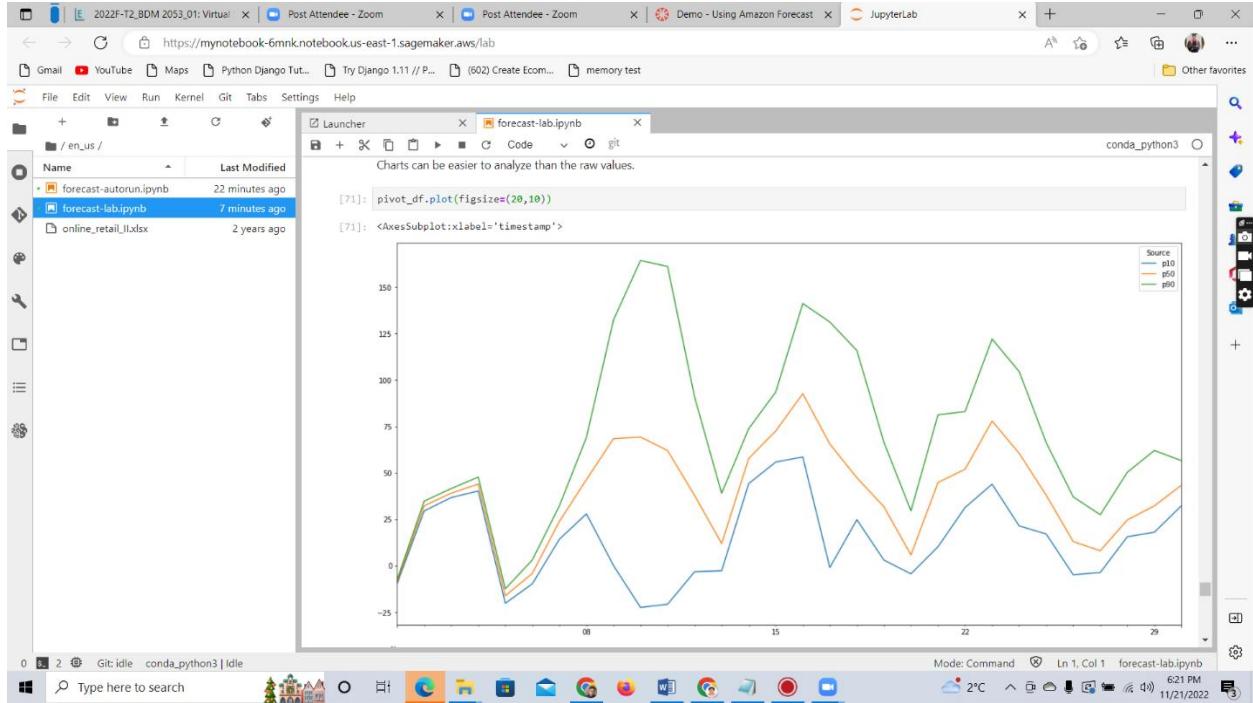
[69]: timestamp value Source

# Now add the P10, P50, and P90 Values
for index, row in prediction_df_p10.iterrows():
    clean_timestamp = datetutil.parser.parse(row['Timestamp'])
    results_df = results_df.append({'timestamp' : clean_timestamp , 'value' : row['Value'], 'Source': 'p10'} , ignore_index=True)
for index, row in prediction_df_p50.iterrows():

[70]: pivot_df = results_df.pivot(columns='Source', values='value', index="timestamp")
pivot_df
```

Source	p10	p50	p90
timestamp			
2010-11-01	-9.757092	-8.605609	-7.506508
2010-11-02	29.454016	32.205383	34.749817
2010-11-03	36.668076	39.031940	41.509037
2010-11-04	40.201233	43.933784	47.714184
2010-11-05	-20.331472	-16.284260	-12.509626
2010-11-06	-9.820400	-4.214762	3.061040
2010-11-07	14.188220	23.852175	32.203808
2010-11-08	27.934444	46.236472	69.068732
2010-11-09	0.050453	68.426559	132.382172
2010-11-10	-22.562981	69.280502	164.464813
2010-11-11	-20.847816	62.014250	161.215790
2010-11-12	-3.324810	37.775764	90.025630
2010-11-13	-2.811493	11.860881	38.911682
2010-11-14	44.176720	57.724674	73.787865
2010-11-15	55.809765	72.565063	93.530701
2010-11-16	58.539242	92.650169	141.247345
2010-11-17	-1.036302	65.498503	131.273514

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

The screenshot shows two browser windows side-by-side.

**Top Window (AWS Academy):**

- Title: Lab 5 - Guided Lab: Facial Recog
- URL: awsacademy.instructure.com/courses/29029/modules/items/2412042
- Content: A modal dialog titled "Start Lab" displays lab details:
  - Region: us-east-1
  - Lab ID: arn:aws:cloudformation:us-east-1:561975534266:stack/c65030a1253e1713205083t1w561975534266/ee4fe280-6f46-11ed-balb-0e5f7b6f9193
  - Creation Time: 2022-11-28T10:03:15-0800
- Below the modal, a list of tasks is visible:
  - Start session at: 2022-11-28T10:03:16-0800
  - Remaining session time: 02:00:00 (120 minutes)
  - Lab status: ready
  - Facial learning
  - Note this is
  - Create how
  - detect
  - The a
  - Create a collection
  - Upload an image of a face
  - Add the image to the collection
  - View the bounding box that was created for the image
  - List the faces in the collection

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

```
[1]: from skimage import io
from skimage.transform import rescale
from matplotlib import pyplot as plt
import boto3
import numpy as np
from PIL import Image, ImageDraw, ImageColor, ImageOps
```

**Task 1: Creating a collection**

In this task, you create a collection in Amazon Rekognition.

You only need to run this step once.

```
[2]: client = boto3.client('rekognition')
collection_id = "Collection"
response = client.create_collection(CollectionId=collection_id)
print("Collection ARN: " + response['CollectionArn'])
print("Status Code: " + str(response['StatusCode']))
print("Done...")
```

Collection ARN: awsrekognition:us-east-1:561975534266:collection/Collection  
Status Code:200  
Done...

**Task 2: Uploading an image to search**

Use the provided sample image, which is named *mum.jpg*, and upload it to this notebook.

Then, look at the image by running the cell.

```
[3]: filename = "mum.jpg"
faceimage = io.imread(filename)
```

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.

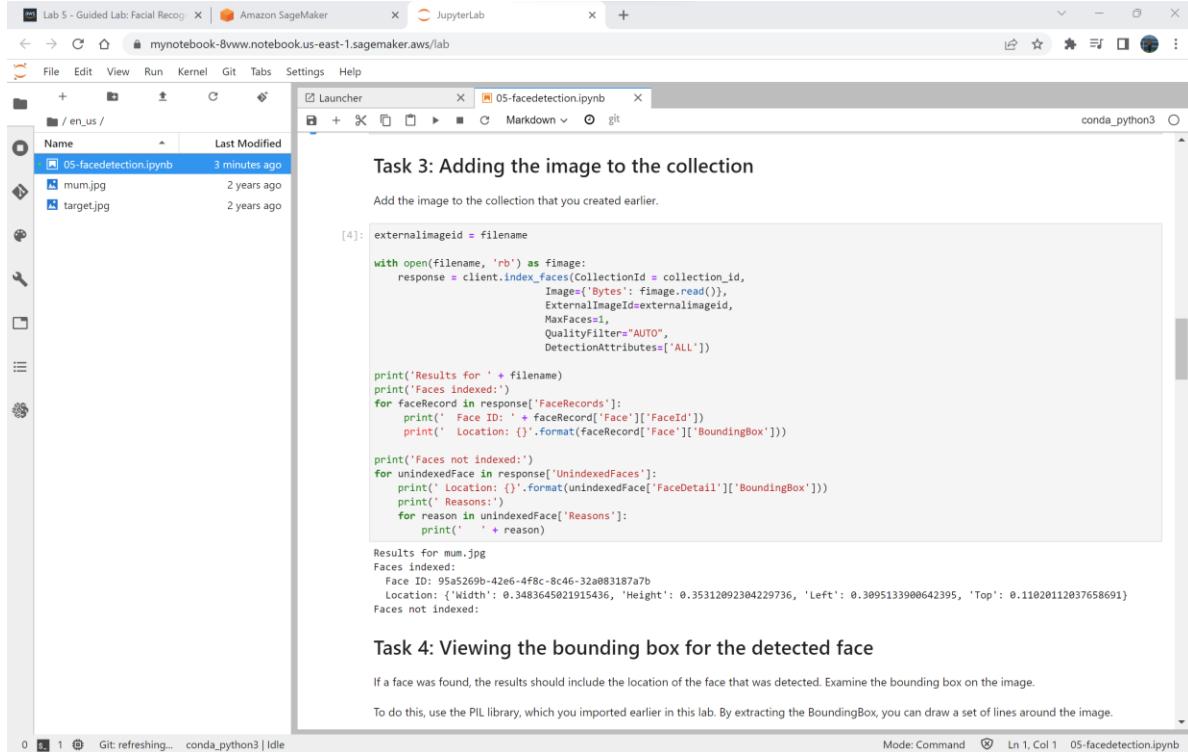
```
plt.imshow(faceimage)
```

**Task 3: Adding the image to the collection**

Add the image to the collection that you created earlier.

```
[4]: externalimageid = filename
```

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.



```
[4]: externalimageid = filename

with open(filename, 'rb') as fimage:
    response = client.index_faces(CollectionId = collection_id,
                                    Image={'Bytes': fimage.read()},
                                    ExternalImageId=externalimageid,
                                    MaxFaces=1,
                                    QualityFilter="AUTO",
                                    DetectionAttributes=['ALL'])

print('Results for ' + filename)
print('Faces indexed:')
for faceRecord in response['FaceRecords']:
    print(' Face ID: ' + faceRecord['Face']['FaceId'])
    print(' Location: {}'.format(faceRecord['Face']['BoundingBox']))

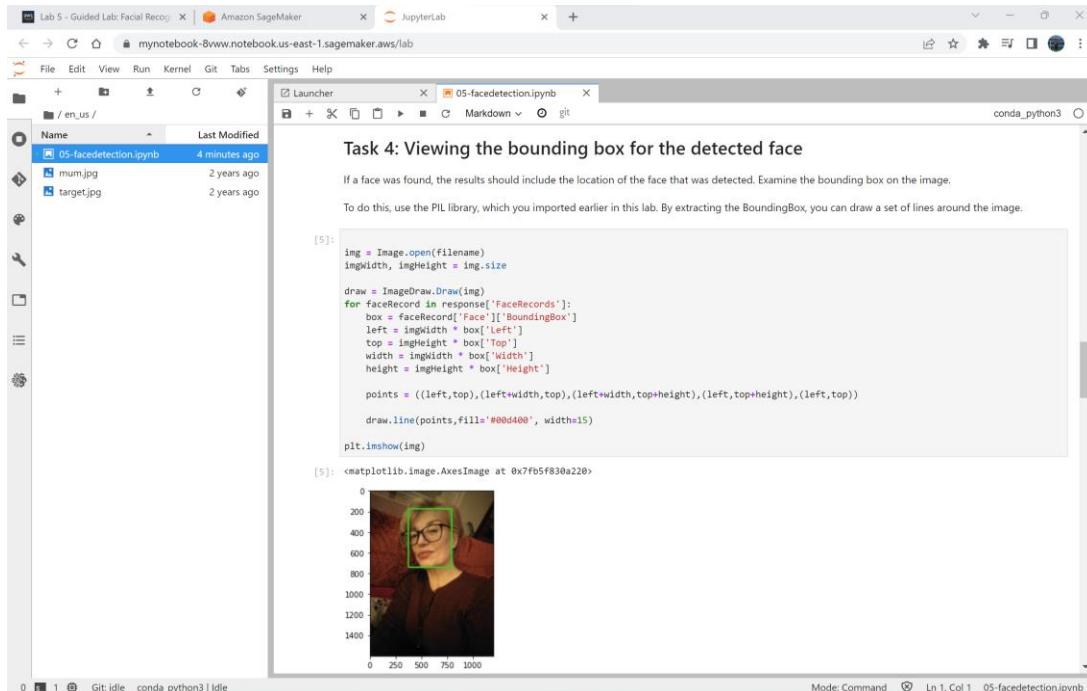
print('Faces not indexed:')
for unindexedFace in response['UnindexedFaces']:
    print(' Location: {}'.format(unindexedFace['FaceDetail']['BoundingBox']))
    print(' Reasons: ')
    for reason in unindexedFace['Reasons']:
        print(' ' + reason)

Results for mum.jpg
Faces indexed:
 Face ID: 95a5269b-42e6-4f8c-8c46-32a083187a7b
Location: {'Width': 0.3483645021915436, 'Height': 0.35312092304229736, 'Left': 0.3095133900642395, 'Top': 0.11020112037658691}
Faces not indexed:
```

**Task 3: Adding the image to the collection**

Add the image to the collection that you created earlier.

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.



```
[5]: img = Image.open(filename)
imgWidth, imgHeight = img.size

draw = ImageDraw.Draw(img)
for faceRecord in response['FaceRecords']:
    box = faceRecord['Face']['BoundingBox']
    left = imgWidth * box['Left']
    top = imgHeight * box['Top']
    width = imgWidth * box['Width']
    height = imgHeight * box['Height']

    points = ((left,top),(left+width,top+height),(left+width,top),(left,top))
    draw.line(points,fill="#00d400", width=15)

plt.imshow(img)
```

**Task 4: Viewing the bounding box for the detected face**

If a face was found, the results should include the location of the face that was detected. Examine the bounding box on the image.

To do this, use the PIL library, which you imported earlier in this lab. By extracting the BoundingBox, you can draw a set of lines around the image.

[5]:



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.

```
[6]: maxResults=2
faces_count=0
tokens=True

response=client.list_faces(CollectionId=collection_id,
                           MaxResults=maxResults)

print('Faces in collection ' + collection_id)

while tokens:

    faces=response['Faces']

    for face in faces:
        print (face)
        faces_count+=1
    if 'NextToken' in response:
        nextToken=response['NextToken']
        response=client.list_faces(CollectionId=collection_id,
                                   NextToken=nextToken,MaxResults=maxResults)
    else:
        tokens=False

Faces in collection Collection
{'FaceId': '95a5269b-42e5-4f8c-8c46-32a083187a7b', 'BoundingBox': {'Width': 0.34836500883102417, 'Height': 0.3531210124492645, 'Left': 0.3095130026344846, 'Top': 0.11028100116729736}, 'ImageId': 'SF3c4b46-11b8-3e4e-9912-37abd7e69e0', 'ExternalImageId': 'mum.jpg', 'Confidence': 99.99430084228516, 'IndexFacesModelVersion': '6.0'}
```

**Task 6: Finding a face by using the collection**

In this step, you will use the collection to detect a face in an image.

Use the provided sample image that's named `target.jpg` and upload it to this notebook.

```
[7]: targetfilename = "target.jpg"
```

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

```
[7]: targetfilename = "target.jpg"
targetimage = Image.open(targetfilename)
plt.imshow(targetimage)
```

[7]: <matplotlib.image.AxesImage at 0x7fb5f827b490>

Next, call the `search_faces_by_image` operation and see if you get a match.

```
[8]: threshold = .70
maxfaces=2

with open(targetfilename, 'rb') as timage:
    response2=client.search_faces_by_image(CollectionId=collection_id,
                                           Image={'Bytes': timage.read()},
                                           FaceMatchThreshold=threshold,
                                           MaxFaces=maxFaces)

faceMatches=response2['FaceMatches']
```

Task 6 : we try to find the face by using our collection in that particular image. We use the image called target.jpg . Then we use search\_faces\_by\_image to operate and see if we get a match.

```

print('Matching faces')
for match in FaceMatches:
    print('FaceId: ' + match['Face']['FaceId'])
    print('Similarity: ' + {:.2f}.format(match['Similarity']) + "%")
    print('ExternalImageId: ' + match['Face']['ExternalImageId'])
    print()

Matching faces
FaceId: 95a5269b-42e6-4f8c-8c46-32a083187a7b
Similarity: 99.97%
ExternalImageId: mum.jpg

```

**Task 7: Drawing a bounding box around the discovered face**

Draw a bounding box around the discovered face.

```

[9]: imgWidth, imgHeight = targetImage.size
draw = ImageDraw.Draw(targetImage)

box = response2['SearchedFaceBoundingBox']
left = imgWidth * box['Left']
top = imgHeight * box['Top']
width = imgWidth * box['Width']
height = imgHeight * box['Height']

points = ((left,top),(left+width,top),(left+width,top+height),(left,top+height),(left,top))
draw.line(points,fill="#00d400", width=15)

plt.imshow(targetImage)

```

[9]: <matplotlib.image.AxesImage at 0x7fb5f81f73d0>

0  
200  
400  
600  
800  
1000

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.

```

print('Attempting to delete collection ' + collection_id)
status_code=0
try:
    responseclient.delete_collection(CollectionId=collection_id)
    status_code=response['StatusCode']
    print('All done!')
except ClientError as e:
    if e.response['Error']['Code'] == 'ResourceNotFoundException':
        print ('The collection ' + collection_id + ' was not found.')
    else:
        print ('Error other than Not Found occurred: ' + e.response['Error']['Message'])
        status_code=response['ResponseMetadata']['HTTPStatusCode']

```

Attempting to delete collection Collection  
All done!  
200

**Congratulations!**

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

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.

The screenshot shows a web browser window for 'awsacademy.instructure.com' with the URL 'https://awsacademy.instructure.com/courses/29029/modules/items/2412042'. The page title is 'Lab 5 - Guided Lab: Facial Recog...'. The left sidebar has sections for Account, Dashboard, Courses, Calendar, Inbox, History, and Help. The main content area shows a 'Modules' section with 'Module 5 ...' and 'Lab 5 - Guided Lab: Facial Recognition'. A central modal dialog box is open, titled 'End Lab'. It displays the following information:  
Region: us-east-1  
Lab ID: arn:aws:cloudformation:us-east-1:561975534266:stack/c65030a125361713205083t1w561975534266/eed4fe280-6f46-11ed-balb-0e5f7b6f9193  
Creation Time: 2022-11-28T10:03:15-0800  
You may close this message box now. Lab resources are terminating ...  
Below the modal, there is a note: 'If you want to keep your lab environment running, go back to the lab page, choose End Lab, and then choose Yes.' At the bottom of the page are navigation buttons: '< Previous' and 'Next >'.

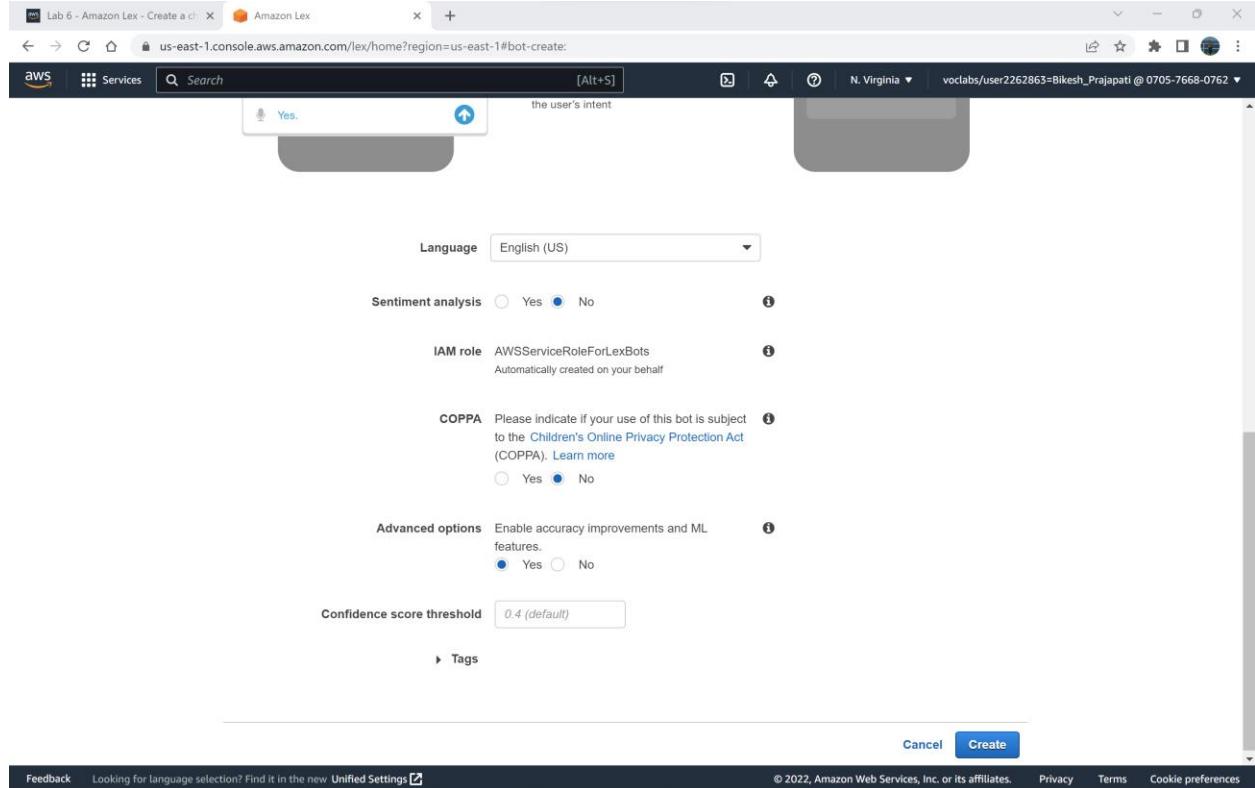
## LAB 6 Guided Lab: Natural Language Processing – Create chatbot

The screenshot shows a browser window for 'Lab 6 - Amazon Lex - Create a chatbot'. The URL is [awsacademy.instructure.com/courses/29029/modules/items/2412052](https://awsacademy.instructure.com/courses/29029/modules/items/2412052). The page title is 'AMLFv1E... > Modules > Module 6 ... > Lab 6 - Amazon Lex - Create a chatbot'. On the left, there's a sidebar with icons for Account, Dashboard, Courses, Calendar, Inbox, History, and Help. The main content area has a 'Start Lab' dialog box. Inside the dialog, it says 'Start session at: 2022-11-28T11:17:36-0800' and 'Remaining session time: 02:58:00(178 minutes)'. Below that, 'Lab status: ready'. At the bottom of the dialog, there are two numbered steps: '9. On the Create your bot page, choose the ScheduleAppointment blueprint.' and '10. For the Intent section, select'. Below the dialog are 'Previous' and 'Next' buttons.

The screenshot shows a browser window for 'Lab 6 - Amazon Lex - Create a chatbot' on the AWS Lambda console. The URL is [us-east-1.console.aws.amazon.com/lexv2/home?region=us-east-1#bots](https://us-east-1.console.aws.amazon.com/lexv2/home?region=us-east-1#bots). The page title is 'Lex Console'. The sidebar on the left shows 'Amazon Lex' with 'Bots' selected, 'Related resources', and a link to 'Return to the V1 console'. The main content area has a banner about 'Announcing Visual Conversation Builder!' with a 'Learn More' button. Below the banner is a table titled 'Bots (0) Info' with columns: Name, Description, Status, Latest Version, and Last updated. A message 'No bots found' is displayed. At the bottom of this section is a 'Create bot' button. Below this is another section titled 'Import/export history (0) Info' with a similar table structure, also showing 'No import/export history found'. The footer of the page includes links for 'Welcome', 'Privacy', 'Terms', and 'Cookie preferences', along with the copyright notice '© 2022, Amazon Web Services, Inc. or its affiliates.'

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.

ScheduleAppointment Latest

**Editor** **Settings** **Channels** **Monitoring**

**Intents** **+ MakeAppointment**

**Slot types** **+ AppointmentTypeValue**

**Error Handling**

**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**

**Context**

**Slots**

Priority	Required Name	Slot type	Version	Prompt	Settings
	e.g. Location	e.g. AMA...		e.g. What city?	
1. ▾	<input checked="" type="checkbox"/> <b>AppointmentType</b>	Appointm...	1 ▾	What type of appointn	⚙️ ⚙️
2. ▾	<input checked="" type="checkbox"/> <b>Date</b>	AMAZO...	Built-In	When should I sched	⚙️ ⚙️
3. ▾	<input checked="" type="checkbox"/> <b>Time</b>	AMAZO...	Built-In	At what time should I	⚙️ ⚙️

**Confirmation prompt**

**Feedback** Looking for language selection? Find it in the new [Unified Settings](#) **Build** **Publish** **?** © 2022, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

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.

ScheduleAppointment Latest

**Editor** **Settings** **Channels** **Monitoring**

**Intents** **+ MakeAppointment**

**Slot types** **+ AppointmentTypeValue**

**Error Handling**

**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**

**Context**

**Slots**

Priority	Required Name	Slot type	Version	Prompt	Settings
	e.g. Location	e.g. AMA...		e.g. What city?	
1. ▾	<input checked="" type="checkbox"/> <b>AppointmentType</b>	Appointm...	1 ▾	What type of appointn	⚙️ ⚙️
2. ▾	<input checked="" type="checkbox"/> <b>Date</b>	AMAZO...	Built-In	When should I sched	⚙️ ⚙️
3. ▾	<input checked="" type="checkbox"/> <b>Time</b>	AMAZO...	Built-In	At what time should I	⚙️ ⚙️

**Confirmation prompt**

**Test bot (Latest)** **Build** **Publish** **?** © 2022, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

I would like to make an appointment

What type of appointment would you like to schedule?

**Inspect response** **Dialog State: ElicitSlot**

Summary  Detail

**Intent:** MakeAppointment

**Slots** (0/3)

AppointmentType null

Date null

Time null

Intents

**MakeAppointment**

Slot types

AppointmentTypeValue

Error Handling

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

Context

Slots

Priority	Required Name	Slot type	Version	Prompt	Settings
1.	AppointmentType	AppointmentType	1	e.g. What type of appointment?	
2.	Date	AMAZON_DATE	Built-In	When should I schedule?	
3.	Time	AMAZON_TIME	Built-In	At what time should I	

Confirmation prompt

Feedback Looking for language selection? Find it in the new Unified Settings

Build Publish ?

Test bot (Latest) Ready. Build complete.

A root canal

When should I schedule your root canal?

5/1/2020

At what time on 2020-05-01?

Clear chat history

Chat with your bot...

Inspect response

Dialog State: ElicitSlot

Intent: MakeAppointment

Slots (2/3)

AppointmentType root canal

Date 2020-05-01

Time null

© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Intents

**MakeAppointment**

Slot types

AppointmentTypeValue

Error Handling

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

Context

Slots

Priority	Required Name	Slot type	Version	Prompt	Settings
1.	AppointmentType	AppointmentType	1	e.g. What type of appointment?	
2.	Date	AMAZON_DATE	Built-In	When should I schedule?	
3.	Time	AMAZON_TIME	Built-In	At what time should I	

Confirmation prompt

Feedback Looking for language selection? Find it in the new Unified Settings

Build Publish ?

Test bot (Latest) Ready. Build complete.

16:00 is available, should I go ahead and book your appointment?

Yes

Intent MakeAppointment is ReadyForFulfillment: AppointmentType:root canal Date:2020-05-01 Time:16:00

Clear chat history

Chat with your bot...

Inspect response

Dialog State: ReadyForFulfillment

Intent: MakeAppointment

Slots (3/3)

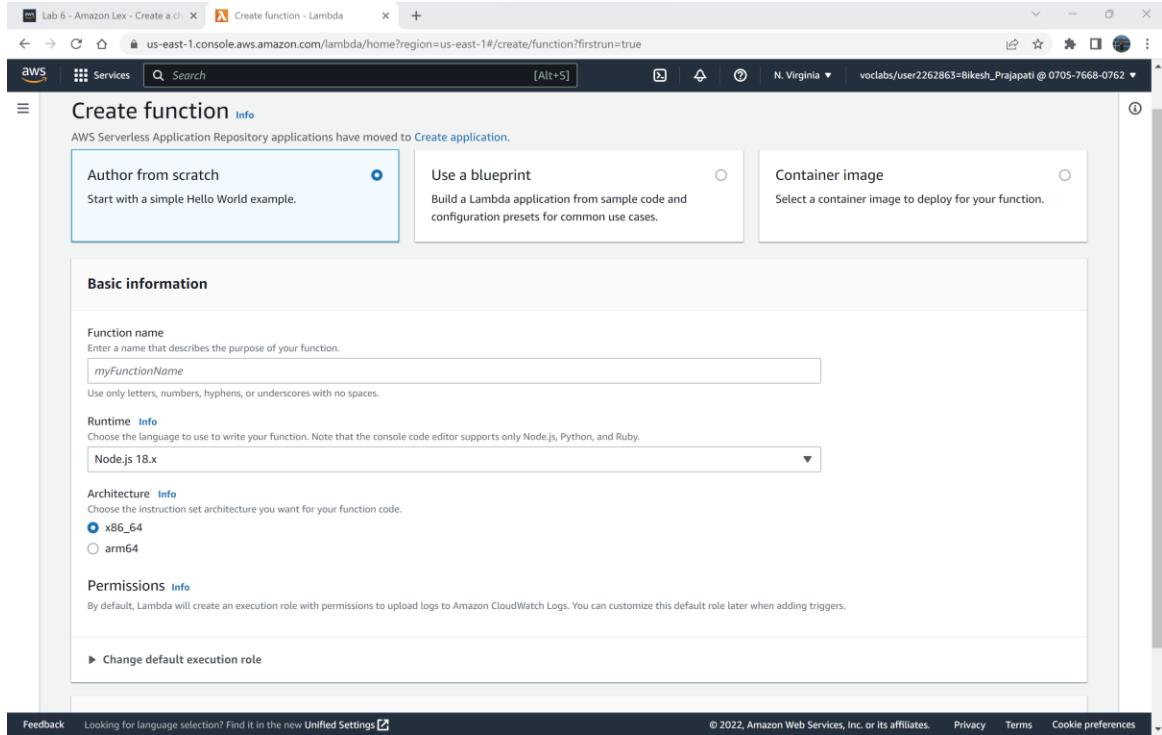
AppointmentType root canal

Date 2020-05-01

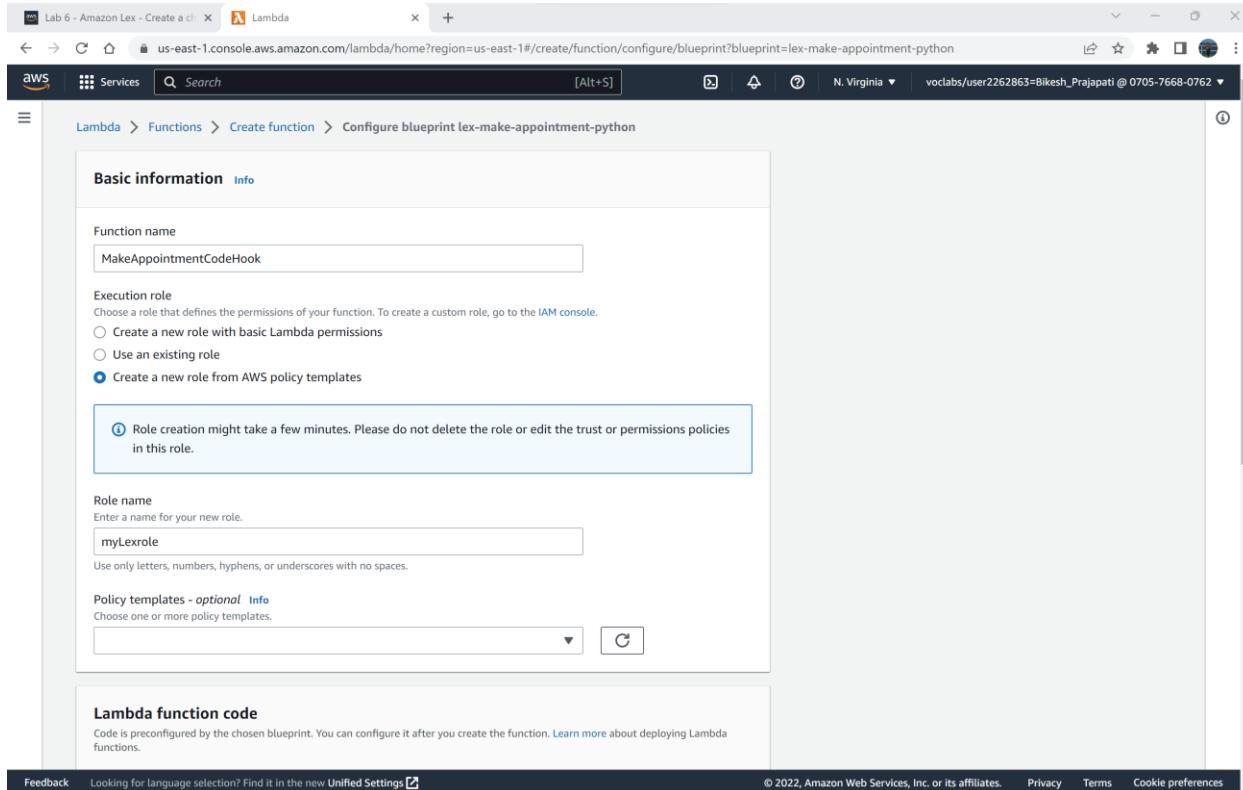
Time 16:00

© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

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 and click create.

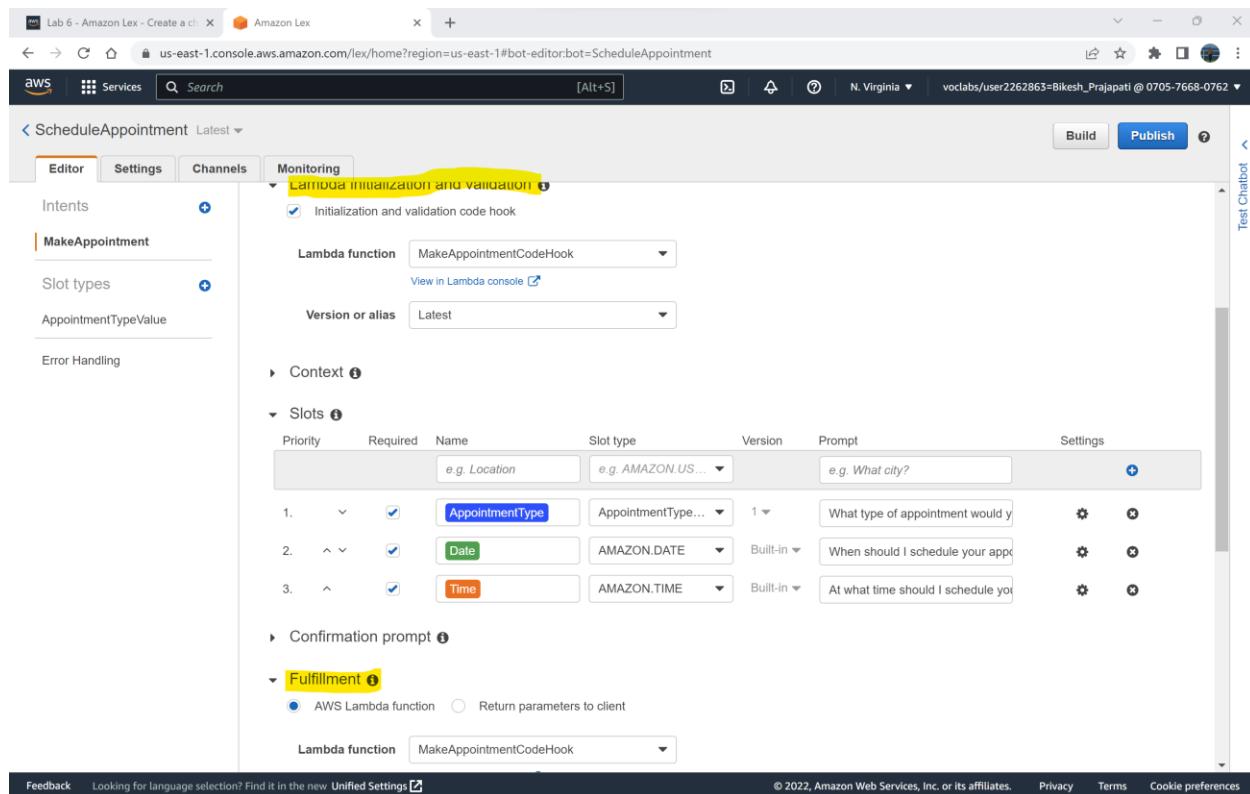
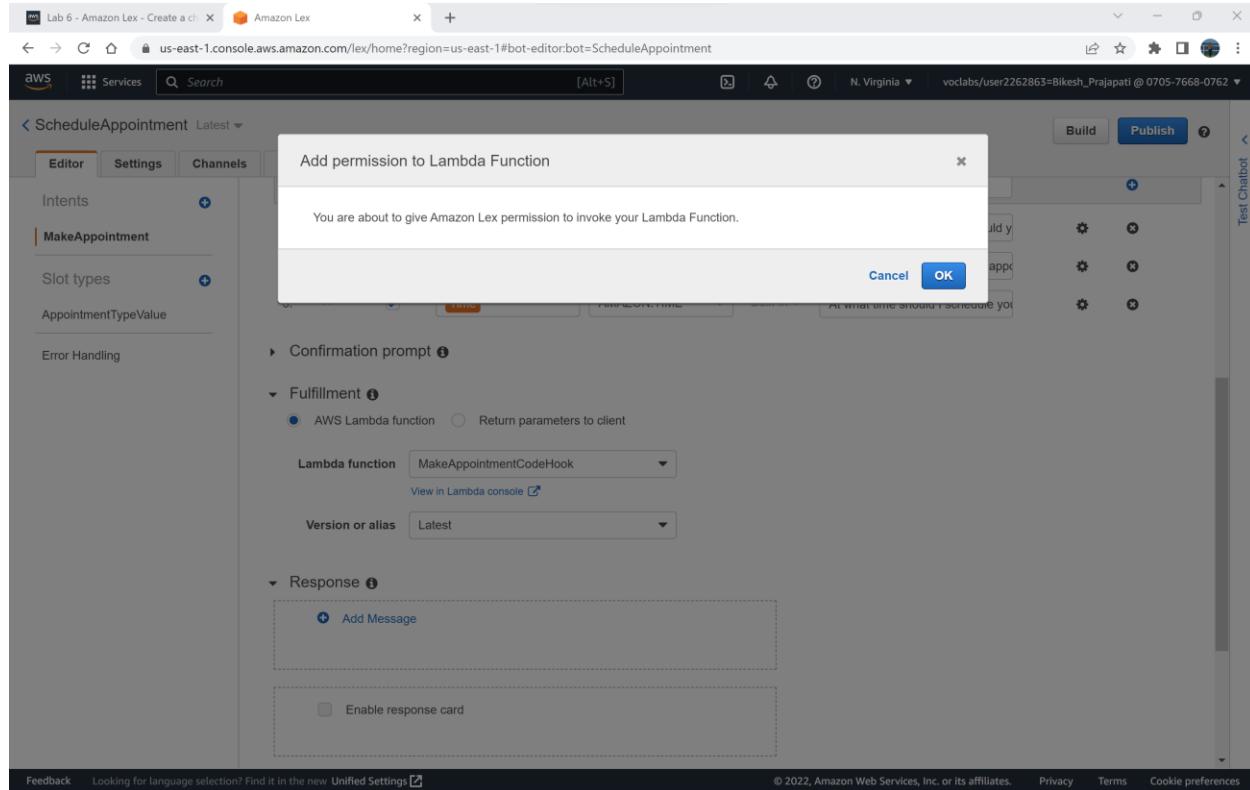


The screenshot shows the AWS Lambda console interface. At the top, there are two tabs: 'Lab 6 - Amazon Lex - Create a cl...' and 'MakeAppointmentCodeHook - L...'. The URL in the address bar is [us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/MakeAppointmentCodeHook?newFunction=true&tab=code](https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/MakeAppointmentCodeHook?newFunction=true&tab=code). The browser title bar also displays 'Lab 6 - Amazon Lex - Create a cl...' and 'MakeAppointmentCodeHook - L...'. The main content area is titled 'MakeAppointmentCodeHook' under 'Lambda > Functions > MakeAppointmentCodeHook'. A green banner at the top says 'Successfully created the function MakeAppointmentCodeHook. You can now change its code and configuration. To invoke your function with a test event, choose "Test".'. Below this, there's a 'Function overview' section with tabs for 'Info' (selected), 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code' tab is currently active. On the left, there's a code editor window with the title 'MakeAppointmentCodeHook' and a 'Layers' section showing '(0)'. On the right, there's a 'Description' section with the text 'Schedule a dentist appointment, using Amazon Lex to perform natural language understanding', 'Last modified' (43 seconds ago), 'Function ARN' (arn:aws:lambda:us-east-1:070576680762:function:MakeAppointmentCodeHook), and a 'Function URL' link. At the bottom of the page, there are links for 'Feedback', 'Unified Settings', 'Privacy', 'Terms', and 'Cookie preferences'.

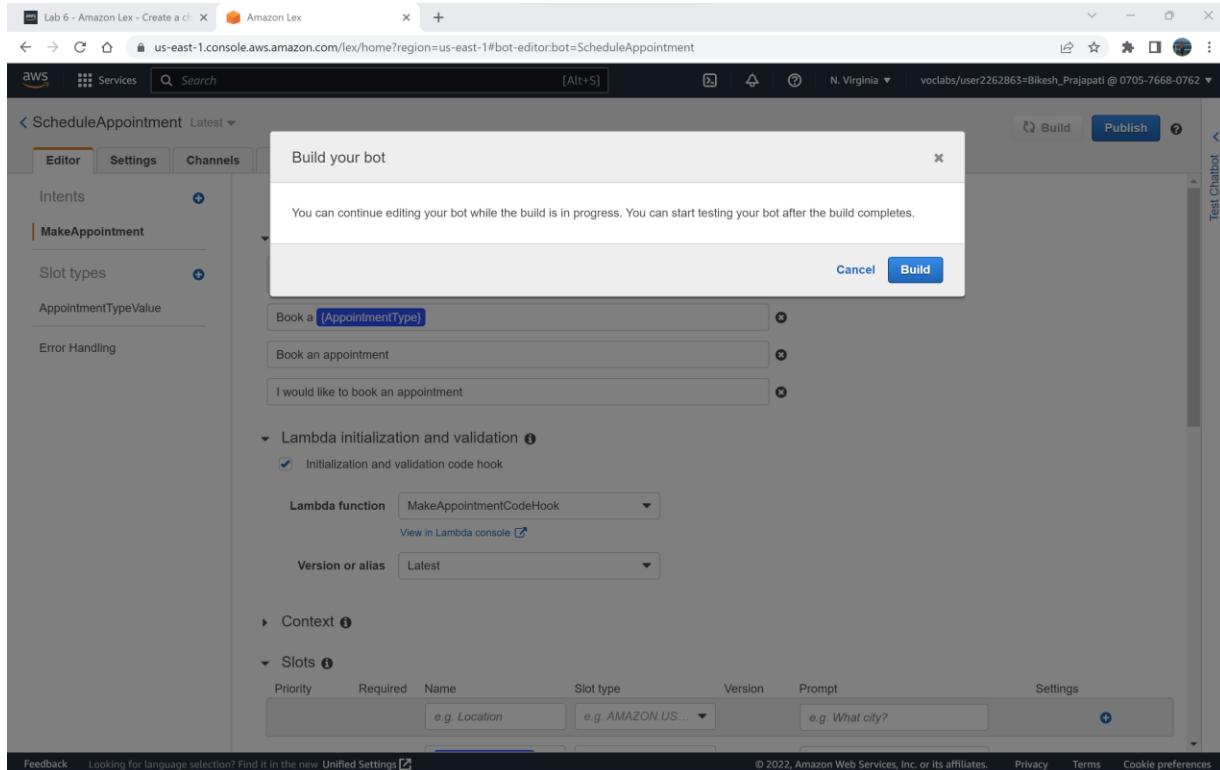
Here is the Lambda Designer window which is ready to use.

This screenshot is identical to the one above, showing the AWS Lambda Function Overview page for 'MakeAppointmentCodeHook'. The main difference is a message box at the bottom left that says 'Execution result: succeeded (logs)' with a 'Details' link. The rest of the interface, including the tabs, code editor, and right-hand details panel, remains the same.

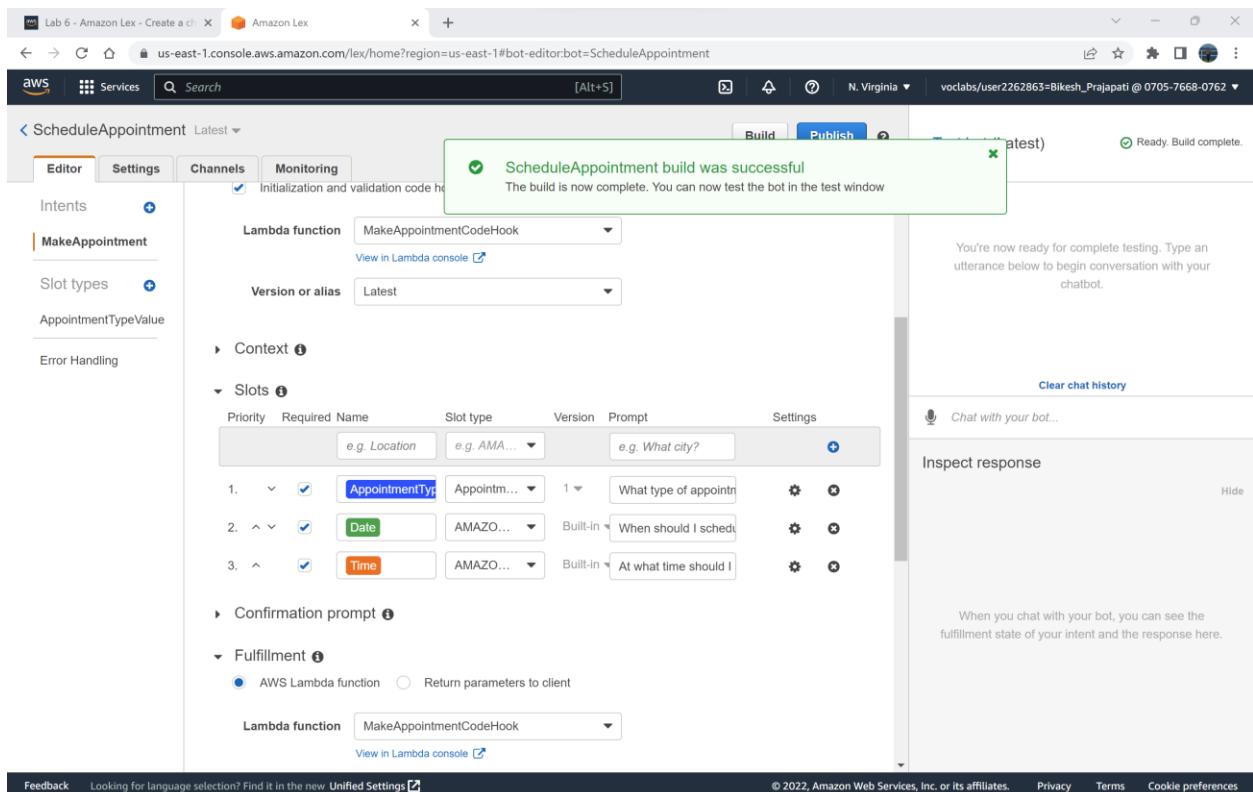
On the designer page, we choose test and provide the event name that we have create. The click test of confirm. we got succeeded message.



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 bot** | **Amazon Lex**

us-east-1.console.aws.amazon.com/lex/home?region=us-east-1#bot-editor:bot=ScheduleAppointment

**ScheduleAppointment** Latest

**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?	<b>+</b>

**Feedback** Looking for language selection? Find it in the new [Unified Settings](#)

© 2022, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

**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

**Inspect response**

**Dialog State:** Fulfilled [Hide](#)

Summary  Detail

**Intent:** MakeAppointment

**Slots** (3/3)  
AppointmentType root canal  
Date 2022-12-02  
Time 16:00

**Lab 6 - Amazon Lex - Create a bot** | **Amazon Lex**

us-east-1.console.aws.amazon.com/lex/home?region=us-east-1#bot-editor:bot=ScheduleAppointment

**ScheduleAppointment** Latest

**Editor** **Settings** **Channels**

**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?	<b>+</b>

**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**

**Feedback** Looking for language selection? Find it in the new [Unified Settings](#)

© 2022, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

**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

**Inspect response**

**Dialog State:** Fulfilled [Hide](#)

Summary  Detail

**Intent:** MakeAppointment

**Slots** (3/3)  
AppointmentType root canal  
Date 2022-12-02  
Time 16:00

**Role Summary**

**Role Description** Your authenticated identities would like access to Cognito.

**IAM Role** Create a new IAM Role

**Role Name** Auth\_Role

**View Policy Document**

**Introducing the new IAM roles experience**  
We've redesigned the IAM roles experience to make it easier to use. [Let us know what you think.](#)

**Attach policy to Auth\_Role**

**Current permissions policies (0)**

**Other permissions policies (Selected 2/793)**

Policy name	Type	Description
<input checked="" type="checkbox"/> AmazonLexReadOnly	AWS managed	Provides read-only access
<input type="checkbox"/> AmazonLexFullAccess	AWS managed	Provides full access to Am
<input checked="" type="checkbox"/> AmazonLexRunBotsOnly	AWS managed	Provides access to Amazc

**Attach policies**

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

**Identity and Access Management (IAM)**

**Unauth\_Role**

**Summary**

Creation date: November 28, 2022, 14:58 (UTC-05:00) ARN: arn:aws:iam::070576680762:role/Unauth\_Role

Last activity: None Maximum session duration: 1 hour

**Permissions** | Trust relationships | Tags | Access Advisor | Revoke sessions

**Permissions policies (3) Info**  
You can attach up to 10 managed policies.

Policy name	Type	Description
AmazonLexReadOnly	AWS managed	Provides read-only access to Amazon Lex
AmazonLexRunBotsOnly	AWS managed	Provides access to Amazon Lex conv
oneClick_Cognito_myidentitypoolUnauth_Role_1669665460007	Customer inline	

We do that permission policies setting for both auth\_role and unauth\_role.

**Amazon S3**

**Buckets**

**Buckets (1) Info**  
Successfully created bucket "lexlab6bikesh". To upload files and folders, or to configure additional bucket settings choose View details.

**Replicate data within and between AWS Regions using Amazon S3 Replication.**

**Account snapshot**

**Buckets (1) Info**  
Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
lexlab6bikesh	US East (N. Virginia) us-east-1	Bucket and objects not public	November 28, 2022, 15:12:45 (UTC-05:00)

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. [Learn more](#)

**Files and folders** (2 Total, 17.0 KB)

Name	Type	Size
error.html	text/html	3.0 KB
index.html	text/html	13.9 KB

**Destination**

Destination  
s3://lexlab6bikesh

**Destination details**

Bucket settings that impact new objects stored in the specified destination.

Feedback Looking for language selection? Find it in the new Unified Settings [Learn more](#)

© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

lab6.zip

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

**Static website hosting**

Use this bucket to host a website or redirect requests. [Learn more](#)

**Static website hosting**

Disable

Enable

**Hosting type**

Host a static website

Use the bucket endpoint as the web address. [Learn more](#)

Redirect requests for an object

Redirect requests to another bucket or domain. [Learn more](#)

For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see [Using Amazon S3 Block Public Access](#)

**Index document**

Specify the home or default page of the website.

index.html

**Error document - optional**

This is returned when an error occurs.

error.html

**Redirection rules - optional**

Redirection rules written in JSON. Alternatively, configure a Lambda function to handle content redirection. [Learn more](#)

Feedback Looking for language selection? Find it in the new Unified Settings [Learn more](#)

© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

lab6.zip

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

The screenshot shows the 'Edit bucket policy' page in the AWS S3 console. The policy is defined as follows:

```

1 - {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "PublicReadGetObject",
6       "Effect": "Allow",
7       "Principal": "*",
8       "Action": [
9         "s3:GetObject"
10      ],
11      "Resource": [
12        "arn:aws:s3:::lexlab6bikesh/*"
13      ]
14    }
15  ]
16 }

```

A modal window titled 'Edit statement PublicReadGetObject' is open, showing the selected action 's3:GetObject'. The sidebar lists other available actions like 's3:PutObject' and services like '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.

The screenshot shows a conversation with the 'Amazon Lex - Appointment BOT'. The history is as follows:

- User: book an appointment
- Bot: Specify Appointment Type
- User: root canal (60 min)
- Bot: Specify Date
- User: 12-2 (Fri)
- Bot: Confirm Appointment
- User: yes
- Bot: Okay, I have booked your appointment. We will see you at 4:00 p.m. on 2022-12-02

A text input field at the bottom says 'What do you want to do?'.

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.

