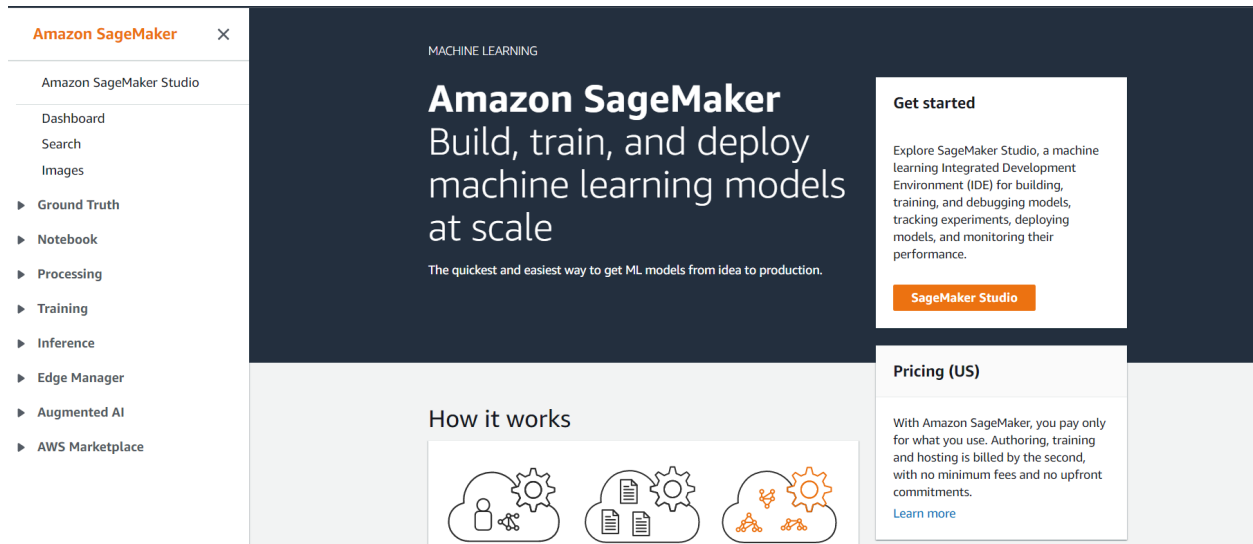


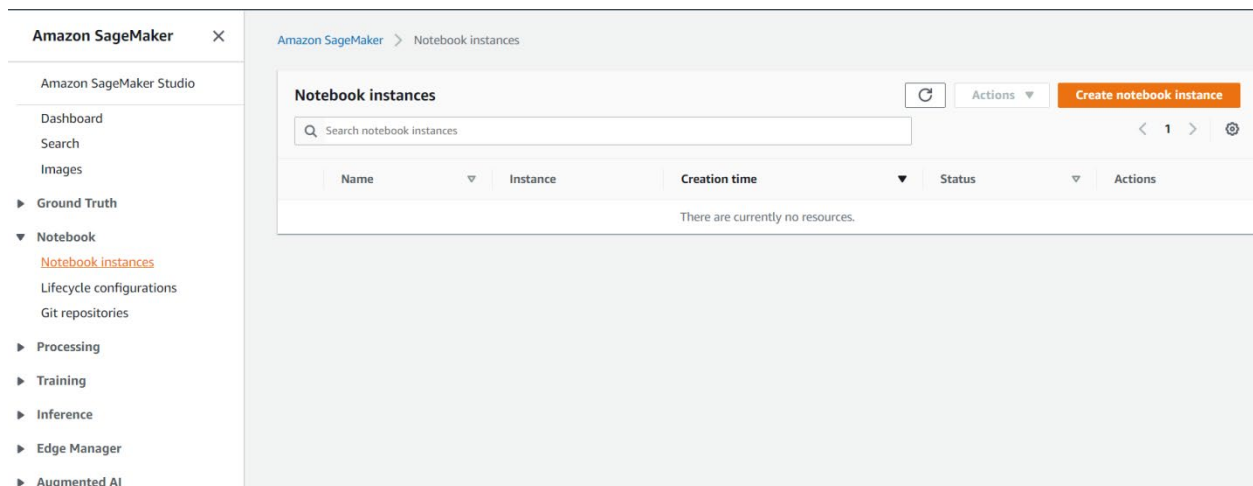
AWS SageMaker

Build, Train, and Deploy Machine Learning models at scale

1. Creating and Importing Data



AWS SageMaker Dashboard



NoteBook Instances Page

Amazon SageMaker > Notebook instances > Create notebook instance

Create notebook instance

Amazon SageMaker provides pre-built fully managed notebook instances that run Jupyter notebooks. The notebook instances include example code for common model training and hosting exercises. [Learn more](#)

Notebook instance settings

Notebook instance name

Mynotebook

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Notebook instance type

ml.m4.xlarge

Elastic Inference [Learn more](#)

none

Amazon SageMaker Notebook Instance is ending its standard support on Amazon Linux AMI (AL1). [Learn more](#)

Platform identifier [Learn more](#)

notebook-ml-v1

▶ Additional configuration

Creating Notebook Instance

Amazon SageMaker

Amazon SageMaker Studio

Dashboard

Search

Images

▶ Ground Truth

▶ Notebook

▶ Processing

▶ Training

▶ Inference

▶ Edge Manager

▶ Augmented AI

▶ AWS Marketplace

Amazon SageMaker > Notebook instances

Notebook instances

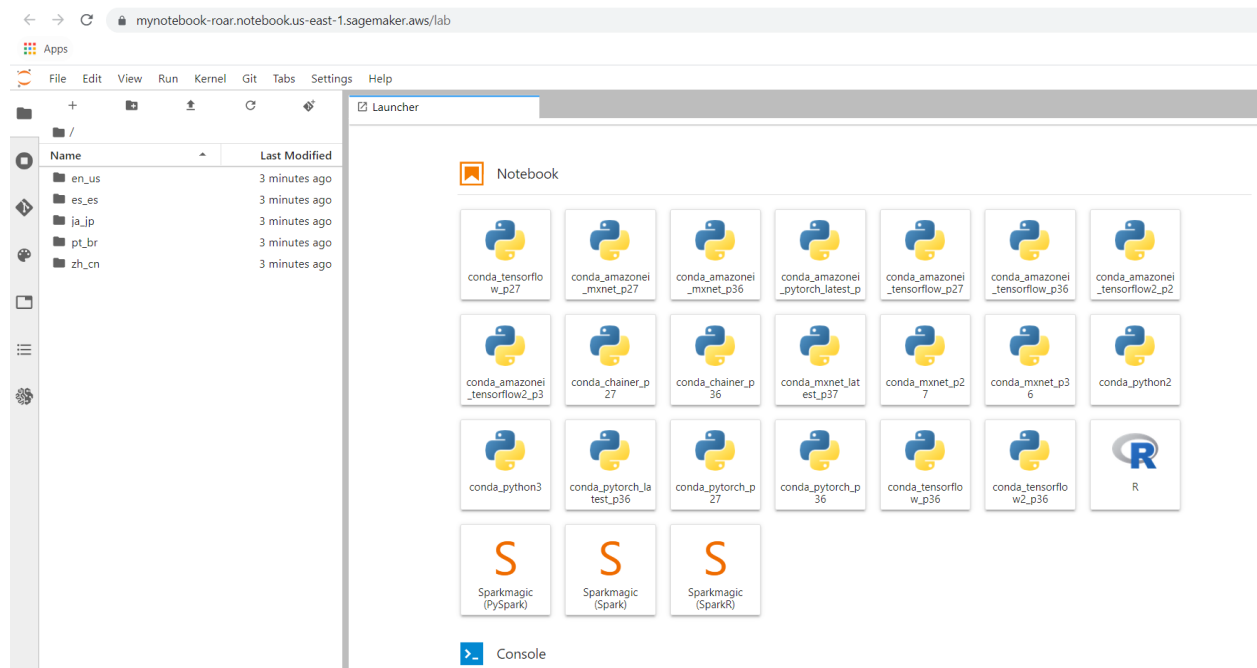
Search notebook instances

< 1 >

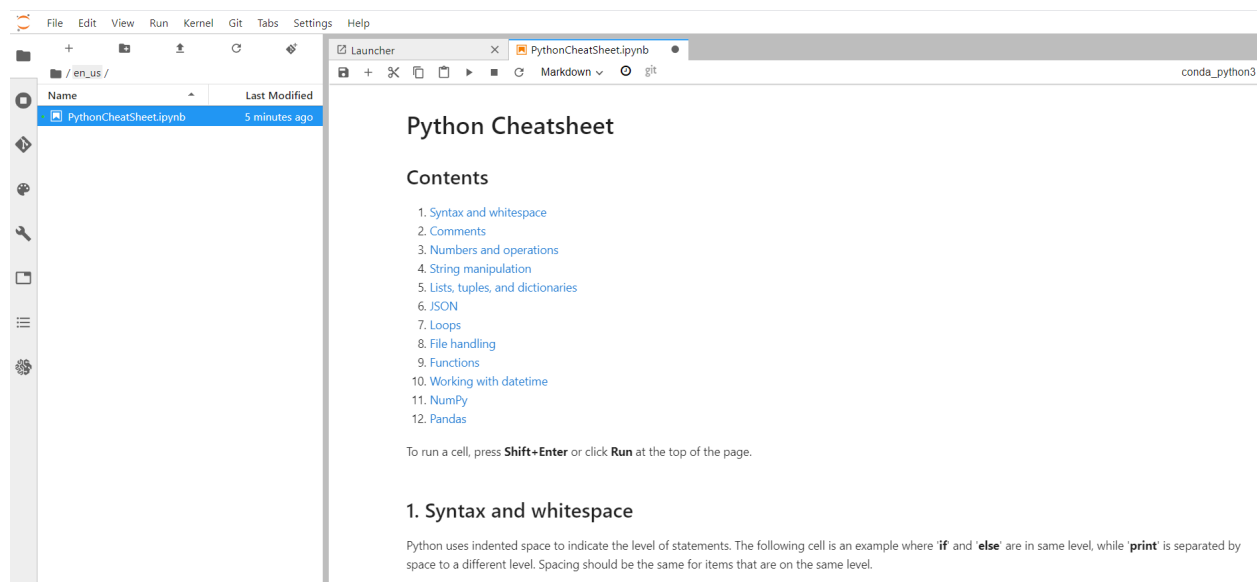
Create notebook instance

	Name	Instance	Creation time	Status	Actions
<input type="radio"/>	Mynotebook	ml.m4.xlarge	Sep 22, 2021 16:28 UTC	InService	Open Jupyter Open JupyterLab

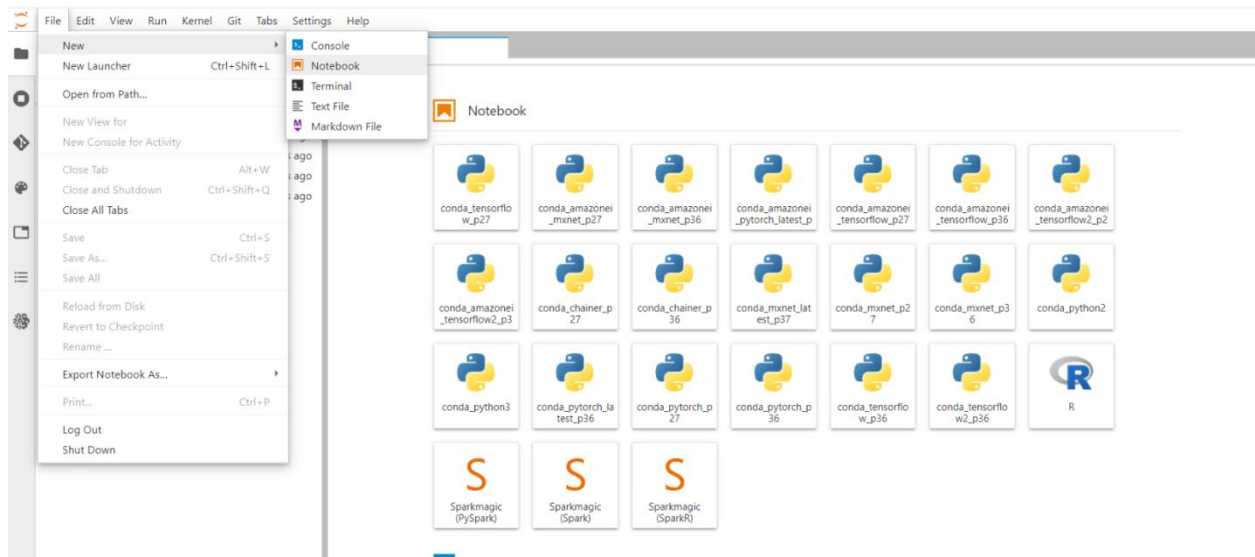
Notebook Successfully Created



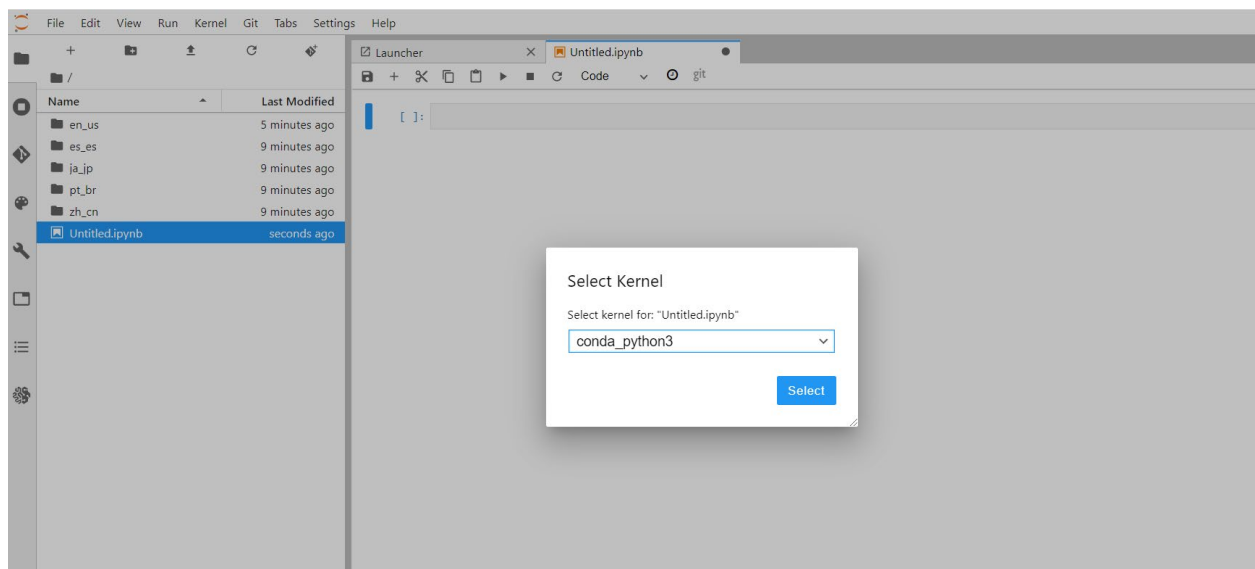
Jupyter Lab for the Notebook



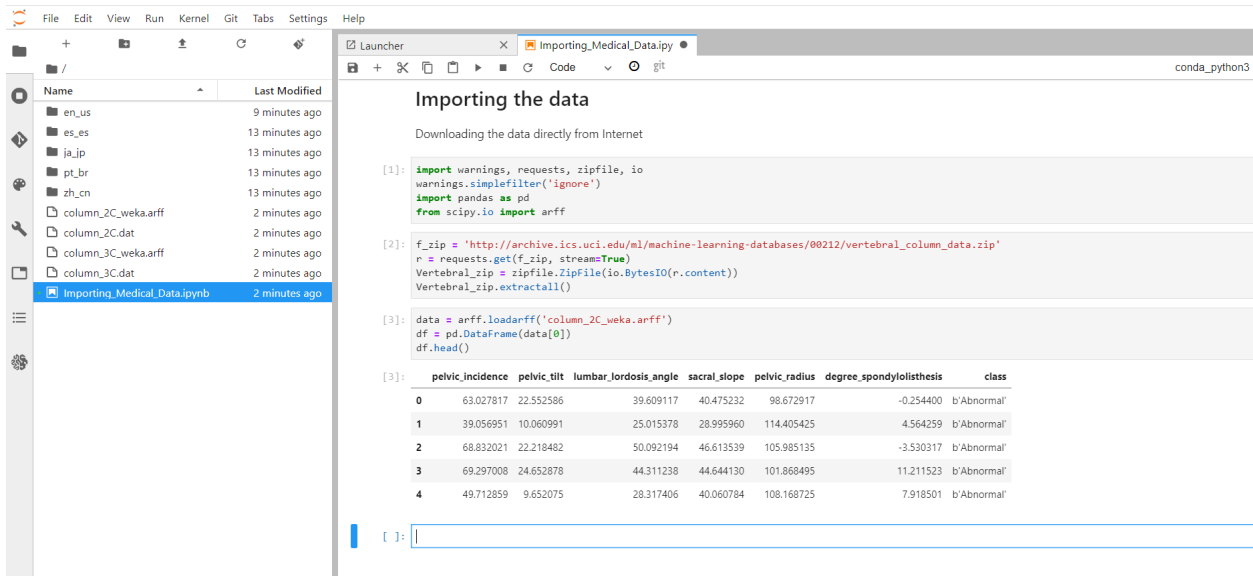
Ipynb files opens in a new window in Jupyter Lab



Creating New Notebook

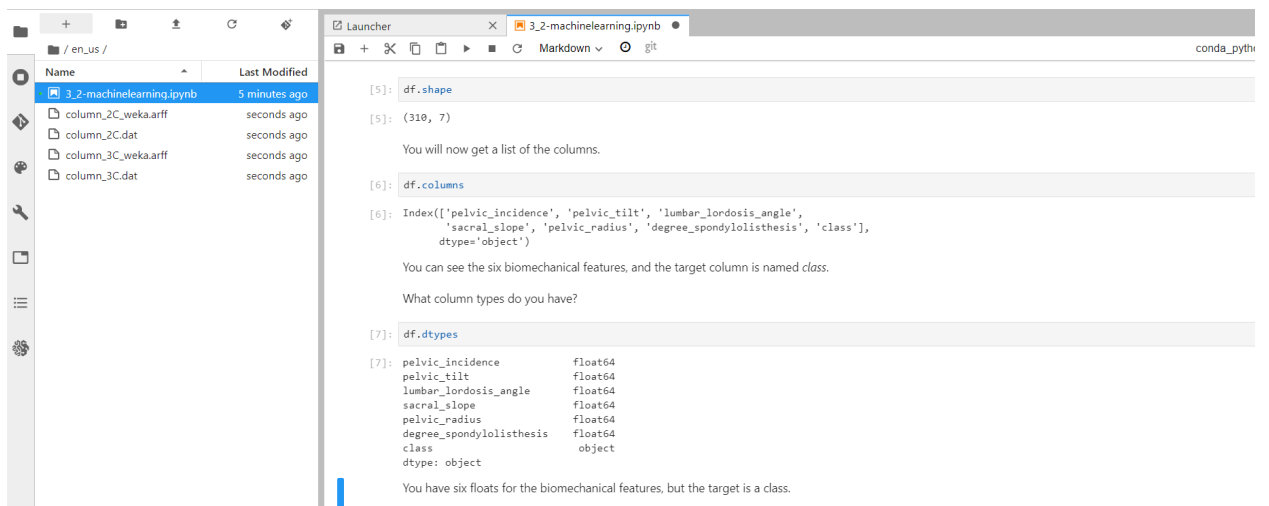


Selecting the Kernel for New Notebook



Working on New Notebook (Downloading and Importing Data)

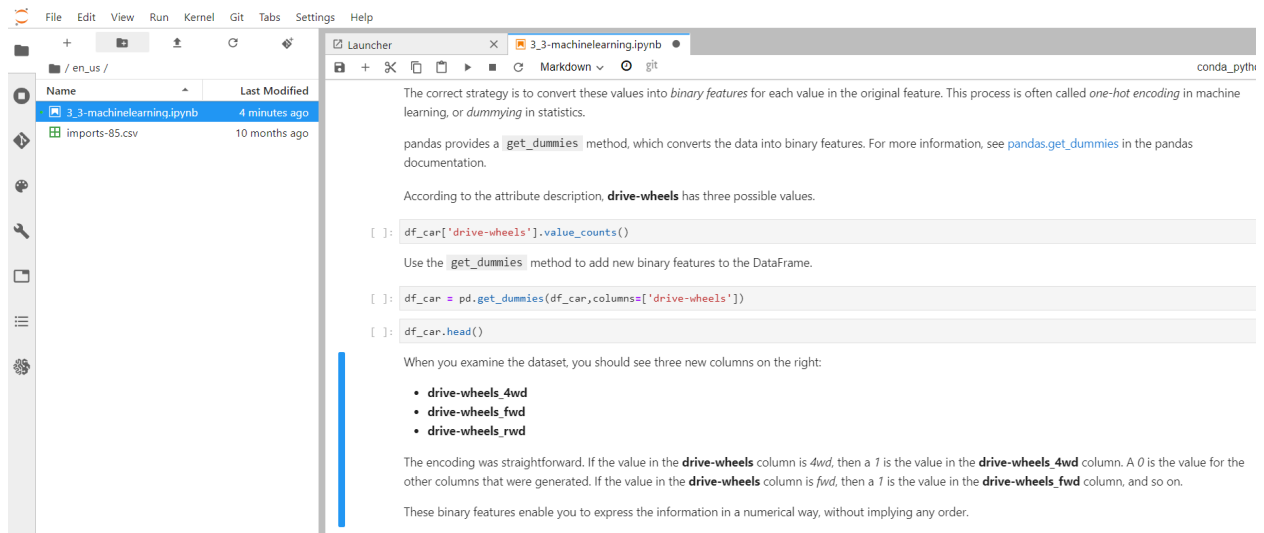
2. Exploring Data using Pandas, Matplotlib



Exploratory Data Analysis

3. Feature Engineering

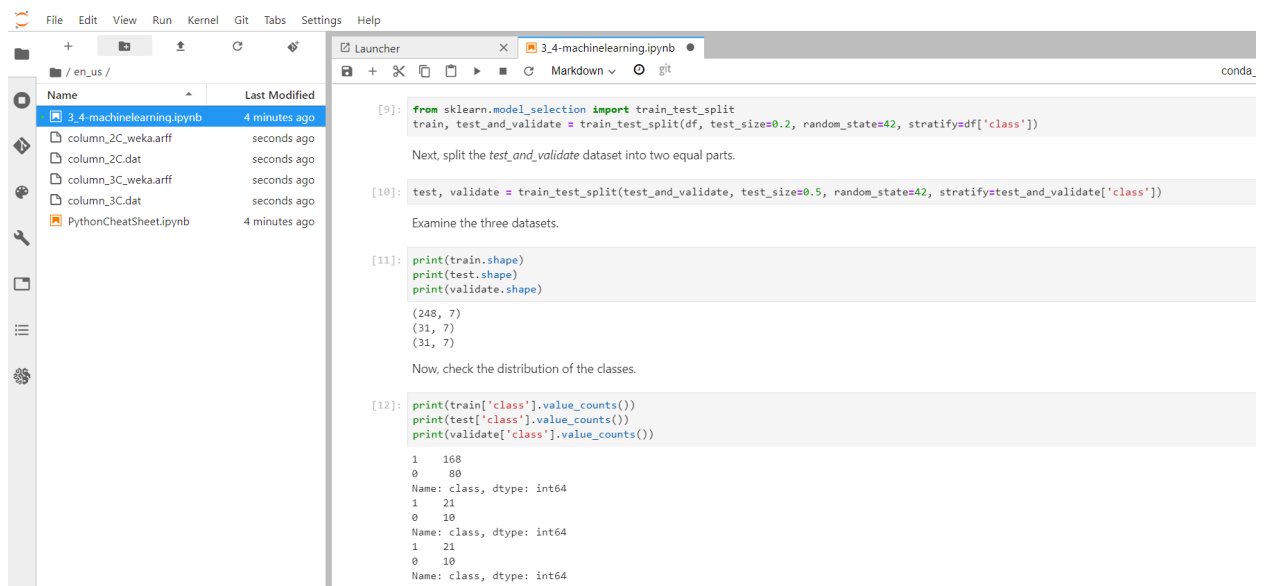
- Cleaning Data
- Dealing with Outliers and Selecting Features
- Encoding Categorical Data



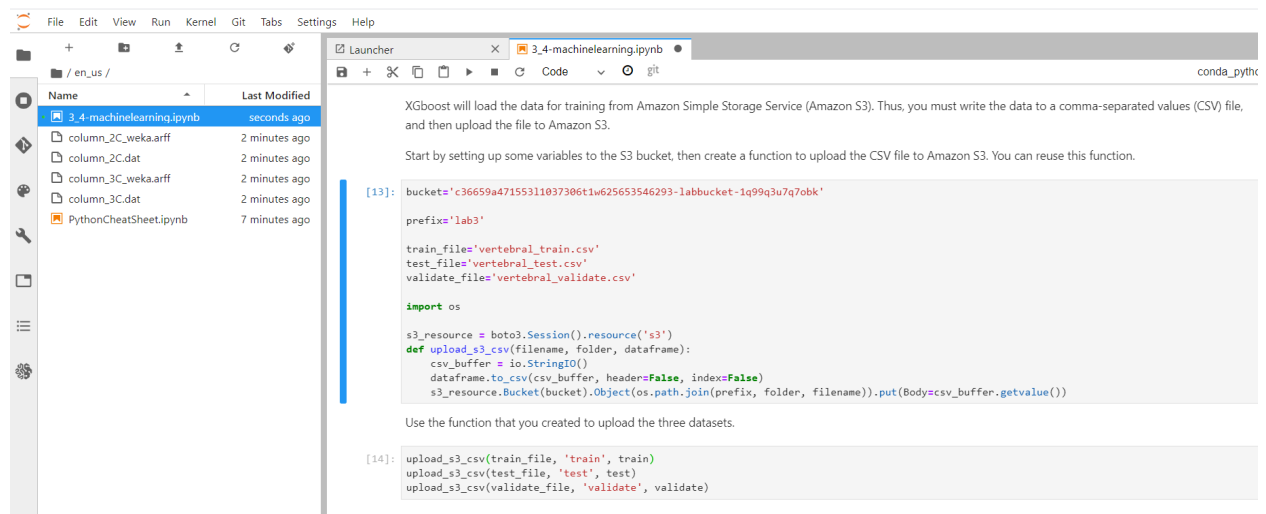
Encoding Categorical Data using Pandas (get_dummies) – AWS SageMaker Notebook

4. Training

- Train / Test / Validation Split



- Uploading Data to S3 Bucket for Model Building



The screenshot shows a Jupyter Notebook with a file explorer on the left and a code editor on the right. The file explorer shows a directory with files like 'column_2C_weka.arff', 'column_2C.dat', 'column_3C_weka.arff', 'column_3C.dat', and 'PythonCheatSheet.ipynb'. The code editor shows the following code:

```
[13]: bucket='c36659a47155311037306t1w625653546293-1abucket-1q99q3u7q7obk'

prefix='lab3'

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

import os

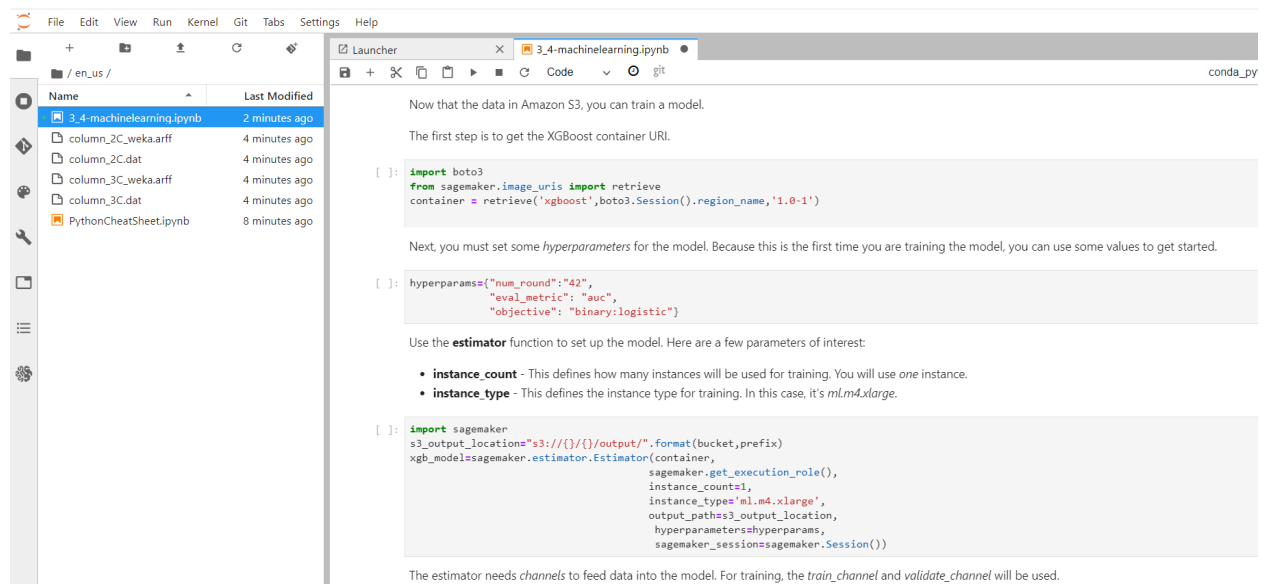
s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False)
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).put(Body=csv_buffer.getvalue())

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

[14]: upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)
```

XGBoost Model loads data from S3 bucket (uploading data to S3 bucket)

- Training the Model



The screenshot shows a Jupyter Notebook with a file explorer on the left and a code editor on the right. The file explorer shows a directory with files like 'column_2C_weka.arff', 'column_2C.dat', 'column_3C_weka.arff', 'column_3C.dat', and 'PythonCheatSheet.ipynb'. The code editor shows the following code:

```
Now that the data in Amazon S3, you can train a model.

The first step is to get the XGBoost container URI.

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

Next, you must set some hyperparameters for the model. Because this is the first time you are training the model, you can use some values to get started.

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

Use the estimator function to set up the model. Here are a few parameters of interest:

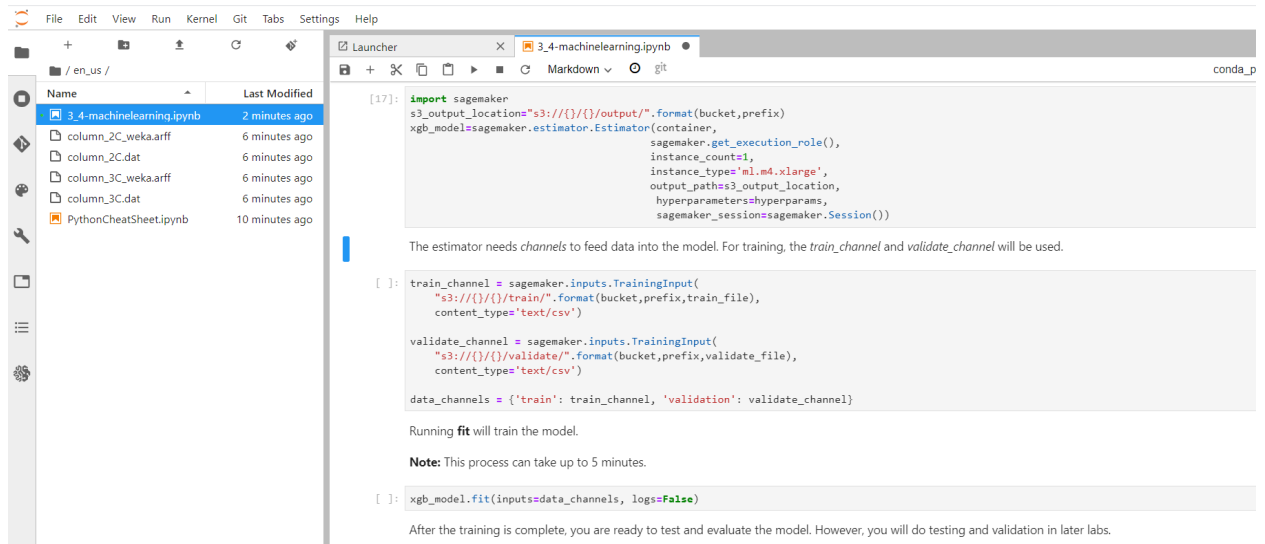
• instance_count - This defines how many instances will be used for training. You will use one instance.
• instance_type - This defines the instance type for training. In this case, it's ml.m4.xlarge.

[ ]: 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.
```

XGBoost Model Retrieved from Container and setting up

- Fitting the Model



```
[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.

[ ]: 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)

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

Creating Channels and Fitting the Model

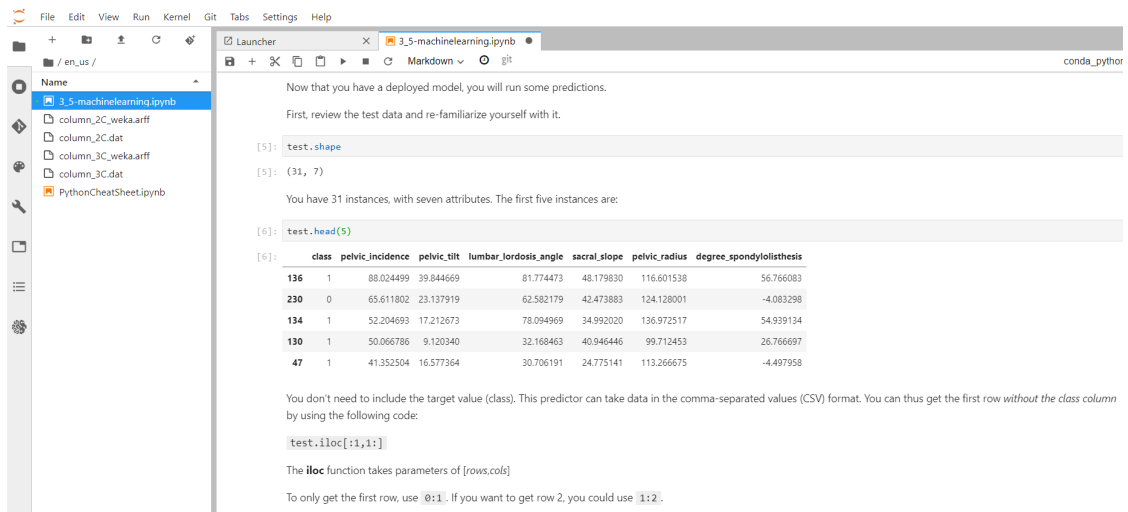
5. Hosting & Using the Model

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

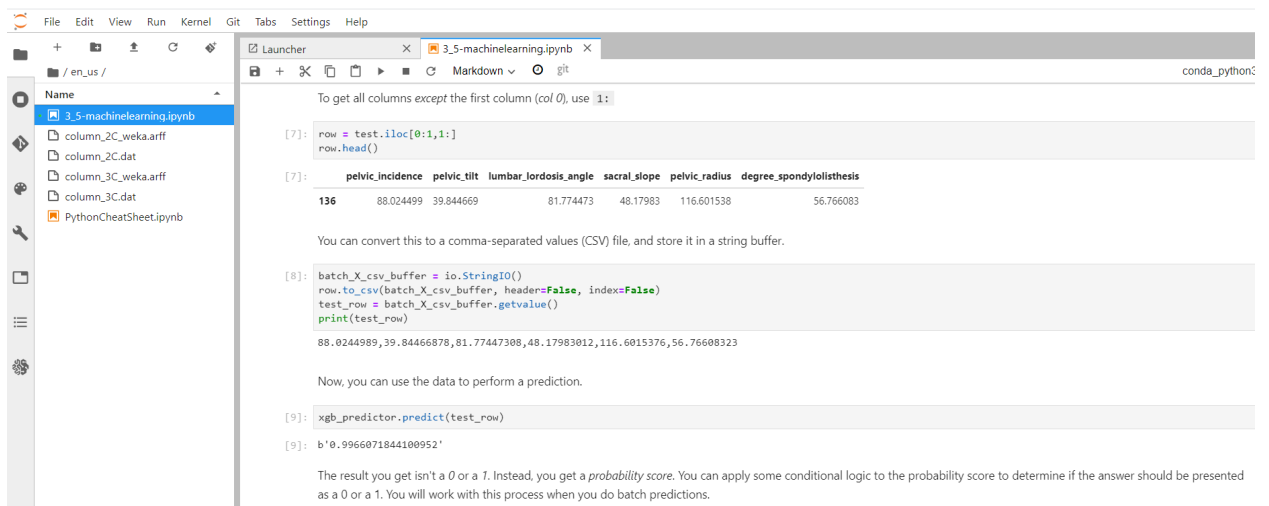
The first step is to deploy the model. Because you have a model object, *xgb_model*, you can use the **deploy** method. For this lab, you will use a single ml.m4.xlarge instance.

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

Deploying the Model



Test data (for Predictions)



Predictions on Test Data

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

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

After Prediction, Delete the endpoint Manually (No an efficient method)

- Using Batch Transformer

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()
```

```
[12]:
```

	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	78.094969	34.992020	136.972517	54.939134
130	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

Next, write your data to a CSV file.

Passing Data to Batch Transformer

```
[13]: batch_X_files='batch-in.csv'
      upload_s3_csv(batch_X_file, 'batch-in', batch_X)
```

Last, before you perform a transform, configure your transformer with the input file, output location, and instance type.

```
[14]: 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.m4.xlarge',
                                              strategy='MultiRecord',
                                              assemble_with='Line',
                                              output_path=batch_output)

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

      xgb_transformer.wait()
```

```
.....[2021-09-22:20:17:34:INFO] No GPUs detected (normal if no gpus installed)
[2021-09-22:20:17:34:INFO] No GPUs detected (normal if no gpus installed)
[2021-09-22:20:17:34: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 {
```

Batch Transformer Running

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, Keys=["/batch-out/{}".format(prefix, 'batch-in.csv.out')])
target_predicted = pd.read_csv(io.BytesIO(obj["Body"].read()), ',', names=['class'])
target_predicted.head(5)
```

```
[15]: 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)
```

Getting results from S3 bucket and storing it in Dataframe

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

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

```
[16]:
```

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

Predictions made and instance terminated automatically