Chapter 7. Big Data Modeling and AI

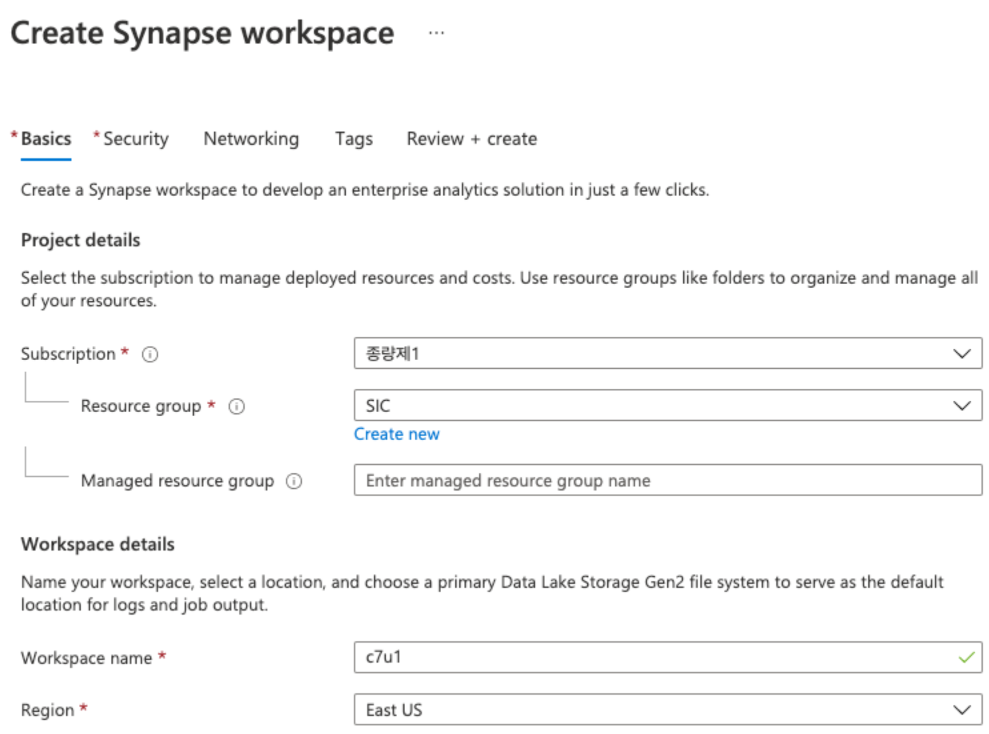
Exercise Workbook

Contents

Lab 1: Working with Azure Machine Learning

In this lab, we will use Azure Synapse and Machine Learning studio from azure portal to create machine learning. We create Synapse Analytics workspace and Azure Machine Learning workspace and link both them.

1. Create a Synapse Analytics Workspace
   1.1 First, log in to the azure portal site (https://portal.azure.com/). If need, please refer to the site contents here.
   1.2 Prerequisites - To use azure, you must have an account and know the basics of how to use it.
   1.3 If necessary, create a free account using the following link. https://azure.microsoft.com/en-us/free/. And If you need to learn about azure, use the link below to learn beforehand. https://azure.microsoft.com/en-us/free/
   1.4 Open the Azure portal, in the search bar enter Synapse without hitting enter.
      1.4.1 In the search results, under Services, select Azure Synapse Analytics.
      1.4.2 Select Create to create a workspace.
   1.5 Fill in the following fields.
      1.5.1 subscription: Select a subscription.
      1.5.2 Resource Group: Select a resource group
      1.5.3 Managed resource group: blank
      1.5.4 Workspace name: A globally unique name
      1.5.5 select data lake storage: Create an account name and filesystem name and set the name.
         (a) if you see error message “There was an error trying to validate storage account name. Please try again”, then register 'Microsoft.storage' in the resource providers menu in the Azure subscription. Select the item and click on 'register' on the top of the page.


Create Synapse workspace ...

*Basics *Security Networking Tags Review + create

Create a Synapse workspace to develop an enterprise analytics solution in just a few clicks.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all of your resources.

Subscription *	종량제1
Resource group *	SIC
	Create new
Managed resource group	Enter managed resource group name

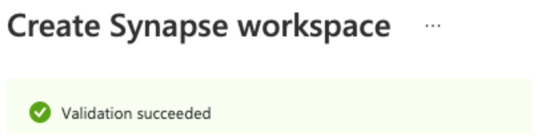
Workspace details

Name your workspace, select a location, and choose a primary Data Lake Storage Gen2 file system to serve as the default location for logs and job output.

Workspace name *	c7u1
Region *	East US

1.5.6 Click the Security tab and configure the SQL administrator credentials.

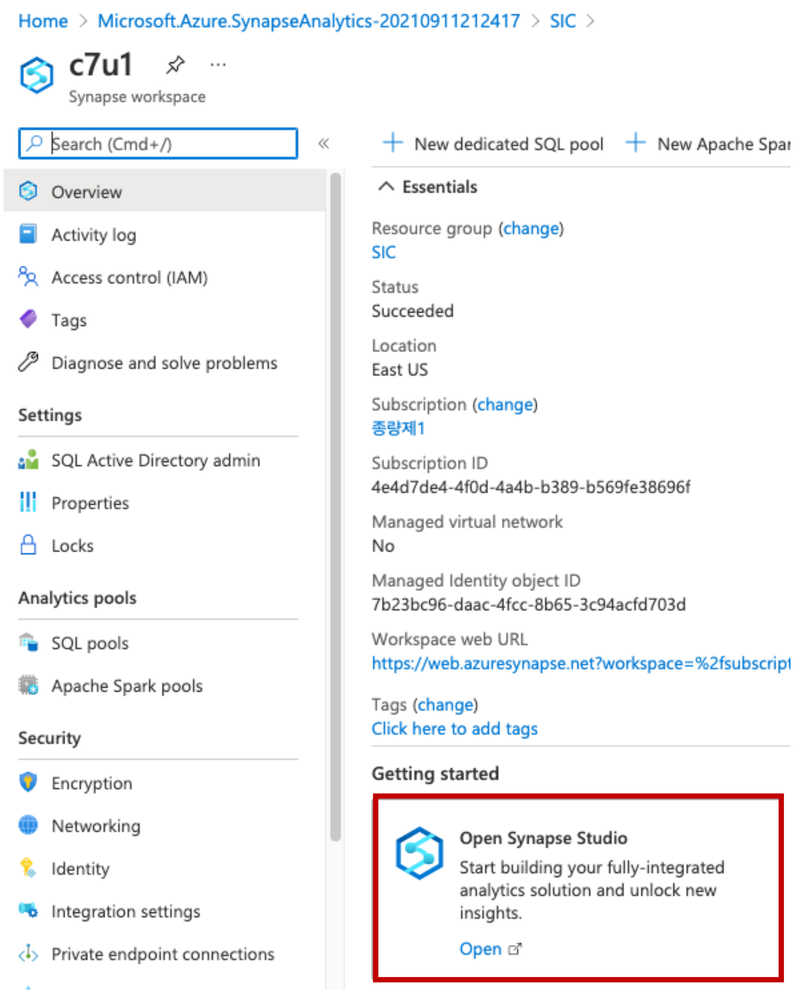
1.5.7 After entering SQL credentials, click "review+create" and if "Validation succeeded" appears, click "create".


Create Synapse workspace ...

Validation succeeded

1.5.8 Create spark pool in Azure Synapse Analytics workspace

1.5.9 Go to Synapse workspace (c7u1), and click the open Synapse Studio.


Home > Microsoft.Azure.SynapseAnalytics-20210911212417 > SIC >

c7u1

Synapse workspace

Resource group (change)
SIC

Status
Succeeded

Location
East US

Subscription (change)
종량제1

Subscription ID
4e4d7de4-4f0d-4a4b-b389-b569fe38696f

Managed virtual network
No

Managed Identity object ID
7b23bc96-daac-4fcc-8b65-3c94acfd703d

Workspace web URL
https://web.azuresynapse.net?workspace=%2fsubscript

Tags (change)
Click here to add tags

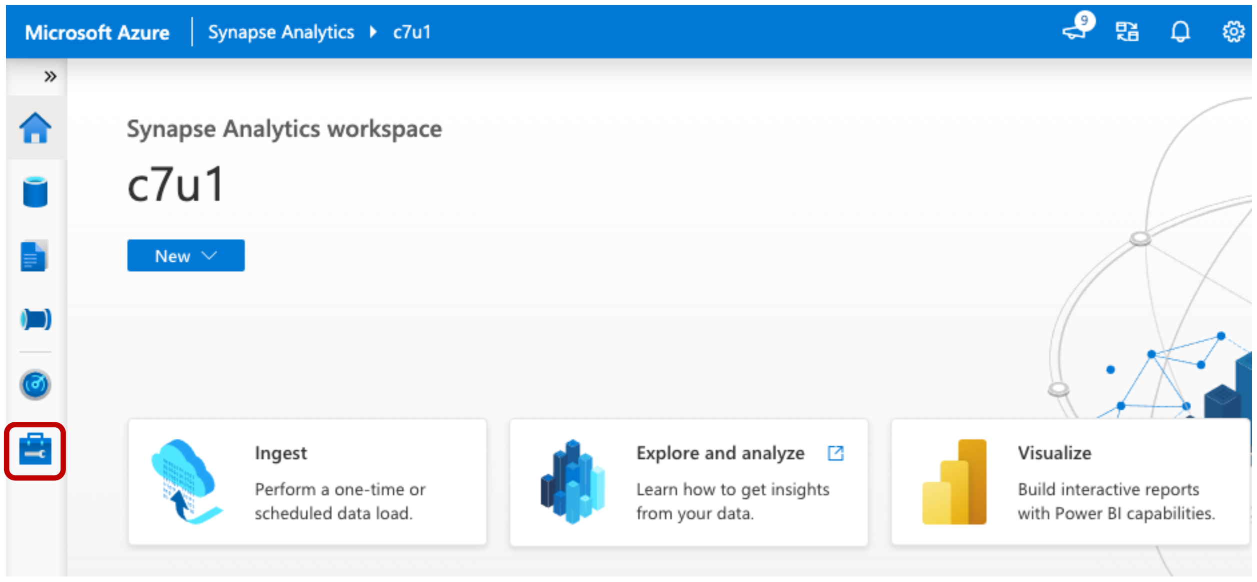
Getting started

Open Synapse Studio

Start building your fully-integrated analytics solution and unlock new insights.

Open

1.5.10 If you select the link above, you will be directed to the Synapse workspace in the same way.



Synapse Analytics workspace

c7u1

Ingest

Perform a one-time or scheduled data load.

Explore and analyze

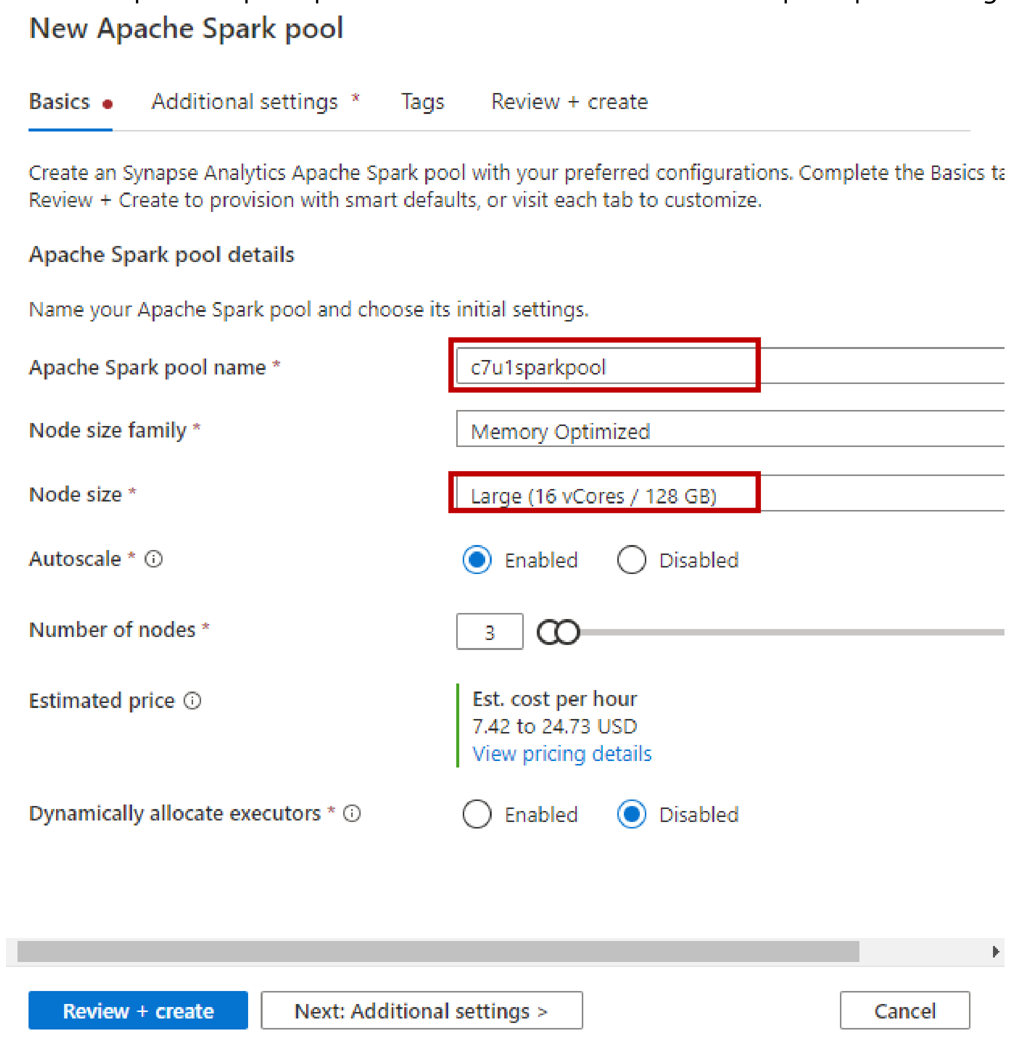
Learn how to get insights from your data.

Visualize

Build interactive reports with Power BI capabilities.

1.5.11 Click the left "manage" icon. (red square)

1.5.12 Select "Apache Spark pool" and select "new" to create spark pool using wizard.



New Apache Spark pool

Basics ● | Additional settings * | Tags | Review + create

Create an Synapse Analytics Apache Spark pool with your preferred configurations. Complete the Basics ta
Review + Create to provision with smart defaults, or visit each tab to customize.

Apache Spark pool details

Name your Apache Spark pool and choose its initial settings.

Apache Spark pool name *	c7u1sparkpool
Node size family *	Memory Optimized
Node size *	Large (16 vCores / 128 GB)
Autoscale * ⓘ	[✓ Enabled] Disabled
Number of nodes *	3
Estimated price ⓘ	Est. cost per hour 7.42 to 24.73 USD View pricing details
Dynamically allocate executors * ⓘ	Enabled [✓ Disabled]

2. Create Azure Machine Learning Workspace

2.1 Create a machine learning workspace

2.1.1 Workspace name: Enter a unique name to identify your workspace

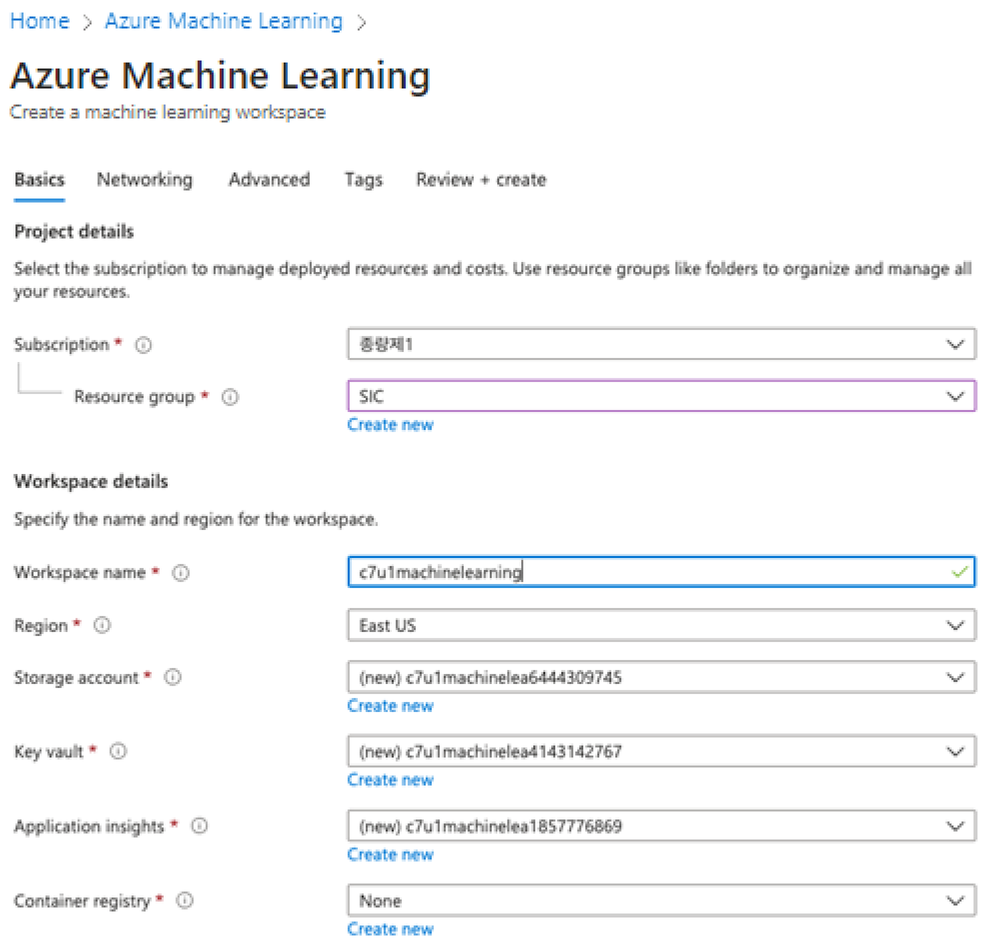
2.1.2 Subscription: Select subscription

2.1.3 Storage account: Create a new storage account for your workspace.

2.1.4 Key vault: Azure Key vault used by the workspace. create a new

2.1.5 Application Insights: An instance of Application Insights for a workspace, and create a new one.

2.1.6 Container Registry: A container registry for workspaces, by default no new workspaces are created in the directory. Created when building a Docker image during training or deployment.



Azure Machine Learning

Create a machine learning workspace

Project details

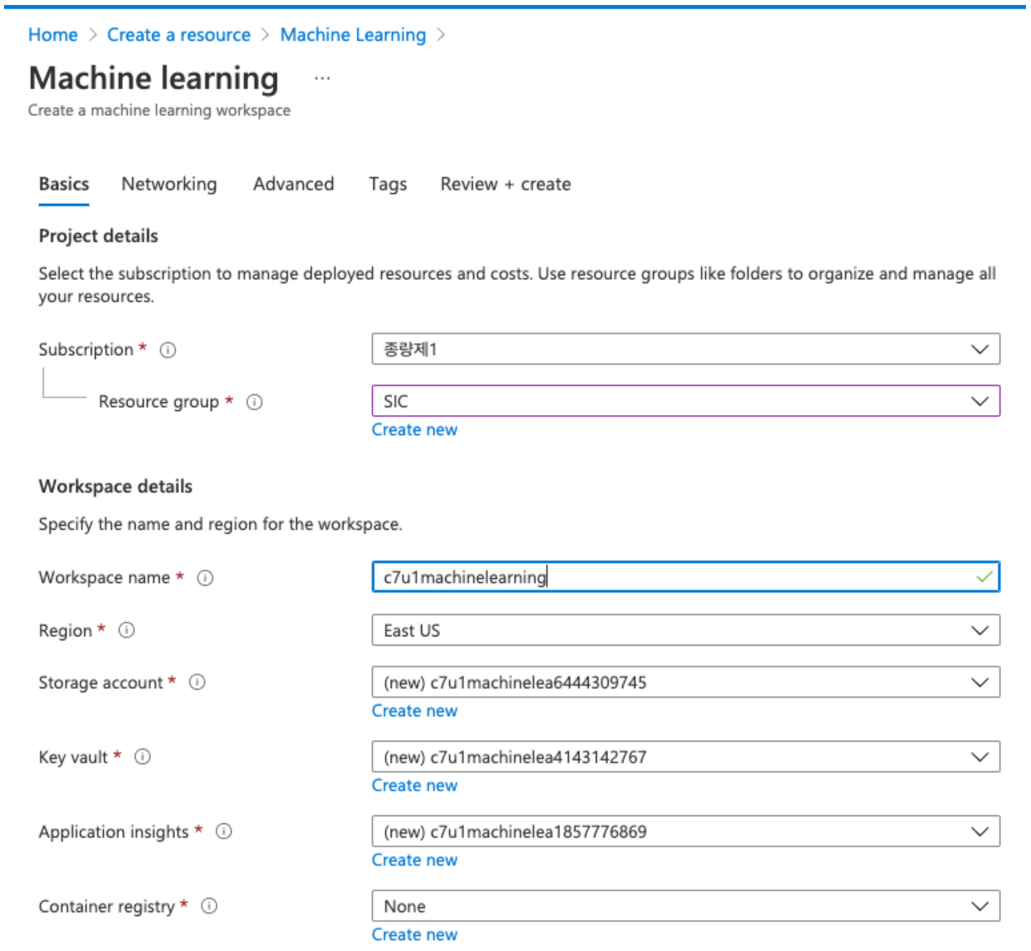
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	종량제1
Resource group *	SIC

Workspace details

Specify the name and region for the workspace.

Workspace name *	c7u1machinelearning
Region *	East US
Storage account *	(new) c7u1machinelea6444309745
Key vault *	(new) c7u1machinelea4143142767
Application insights *	(new) c7u1machinelea1857776869
Container registry *	None


Home > Create a resource > Machine Learning >

Machine learning

Create a machine learning workspace

Basics | Networking | Advanced | Tags | Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

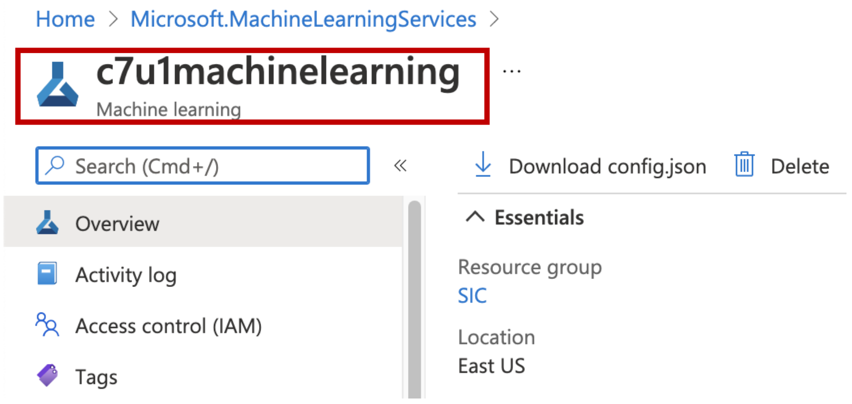
Subscription *	종량제1
Resource group *	SIC

Workspace details

Specify the name and region for the workspace.

Workspace name *	c7u1machinelearning
Region *	East US
Storage account *	(new) c7u1machinelea6444309745
Key vault *	(new) c7u1machinelea4143142767
Application insights *	(new) c7u1machinelea1857776869
Container registry *	None

2.1.7 Select "Review+create" and click "create" to create a workspace.


Home > Microsoft.MachineLearningServices >

c7u1machinelearning

Machine learning

Resource group: SIC

Location: East US

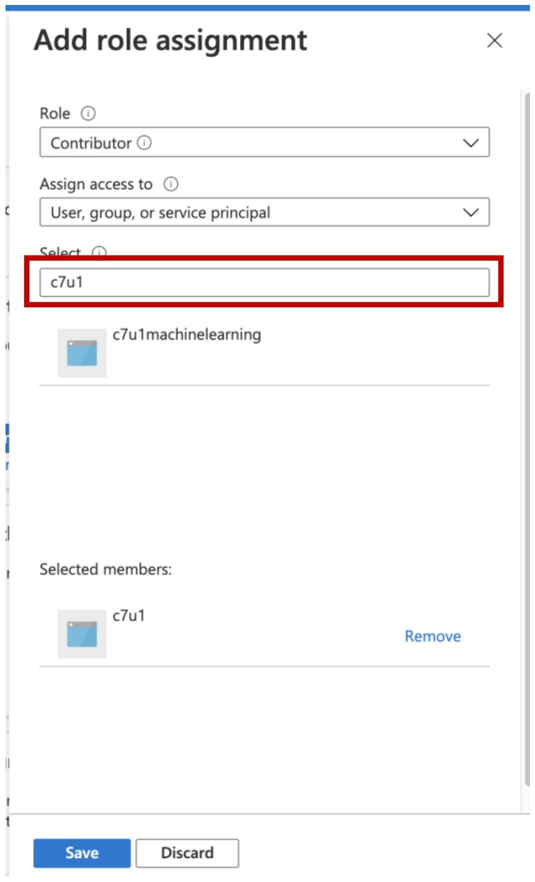
3. Create a linked service to Azure Machine Learning

Connect the workspace created in Synapse Analytics to the Azure Machine Learning workspace.

3.1 Create a Workspace Managed ID.

3.1.1 Go to Azure Machine Learning workspace resource (c7u1machinelearning)

3.1.2 In left side, select the "Access control (IAM)". And click the "Add" menu and select the "Add role Assignment". Then the following pop-up menu is created on the right.



Add role assignment

Role: Contributor

Assign access to: User, group, or service principal

Select: c7u1

c7u1machinelearning

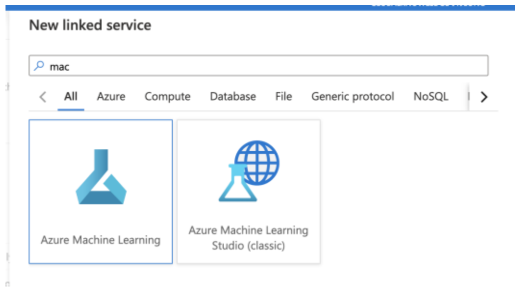
Selected members:

c7u1

3.1.3 Select the value of each field as shown in the figure, and enter the name of the synapse workspace (c7u1) in the select item. Then a list will appear at the bottom, choose the name and save it.

3.1.4 Create an Azure ML linked service to Synapse. Go to Synapse Analytics Studio page (refer this page in 1.4.10 section)

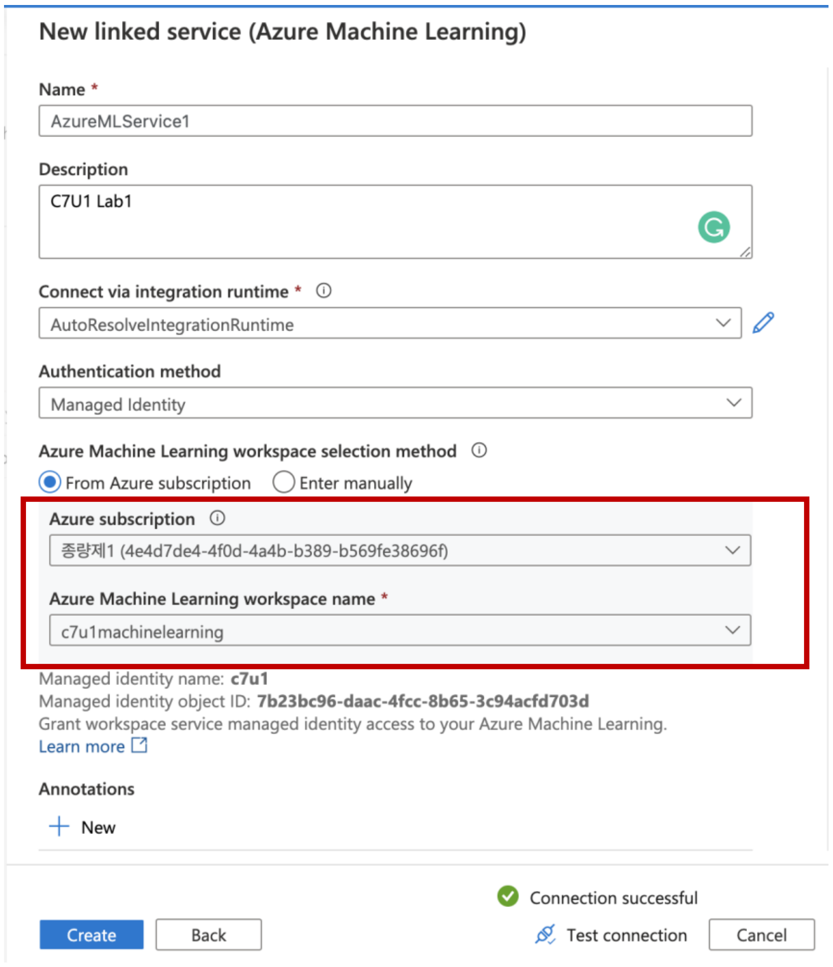
3.1.5 Click the "manage" and select the "linked service". Then click the "+New" for new linked service (Azure Machine Learning) pop-up.



New linked service

mac

3.1.6 Here, if you select Azure Machine Learning, enter the subscript name and other information as shown below. For other values, choose the default and select the name of the Machine Learning workspace you created in the red box.


New linked service (Azure Machine Learning)

Name *
AzureMLService1

Description
C7U1 Lab1

Connect via integration runtime *
AutoResolveIntegrationRuntime

Authentication method
Managed Identity

Azure Machine Learning workspace selection method
[✓ From Azure subscription] Enter manually

Azure subscription
종량제1 (4e4d7de4-4f0d-4a4b-b389-b569fe38696f)

Azure Machine Learning workspace name *
c7u1machinelearning

Managed identity name: **c7u1**
Managed identity object ID: **7b23bc96-daac-4fcc-8b65-3c94acfd703d**
Grant workspace service managed identity access to your Azure Machine Learning.
Learn more

Annotations

3.1.7 Check "test connection" before creation, and publish in linked services after creation.

4. Create spark table for Training Set

4.1 Import the NYC_taxidata.ipynb file using notebook in synapse studio. The NYC file can be imported from the student account Data folder in the virtualbox, or use the NYC example below.

4.1.1 NYC example

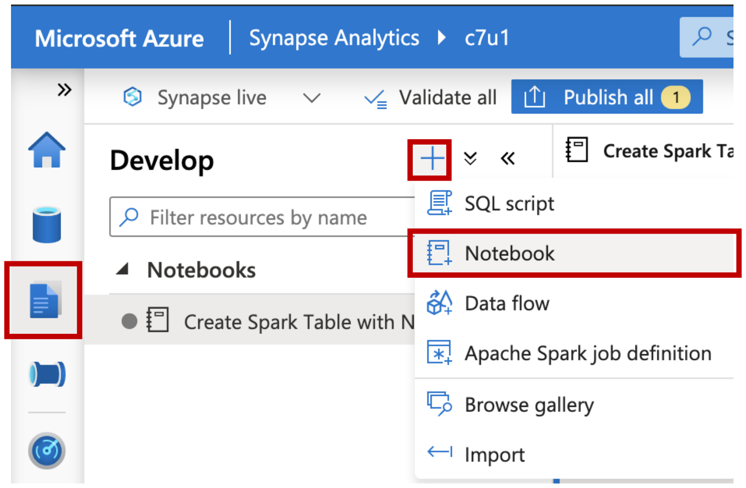
cell 1	`# This is a package in preview.`
	`from azureml.opendatasets import NycTlcYellow`
	
	`from datetime import datetime`
	`from dateutil import parser`

	`end_date = parser.parse('2018-06-06')`
	`start_date = parser.parse('2018-05-01')`
	`nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)`
	`nyc_tlc_df = nyc_tlc.to_spark_dataframe()`
	
	`# Display top 5 rows`
	`display(nyc_tlc_df.limit(5))`

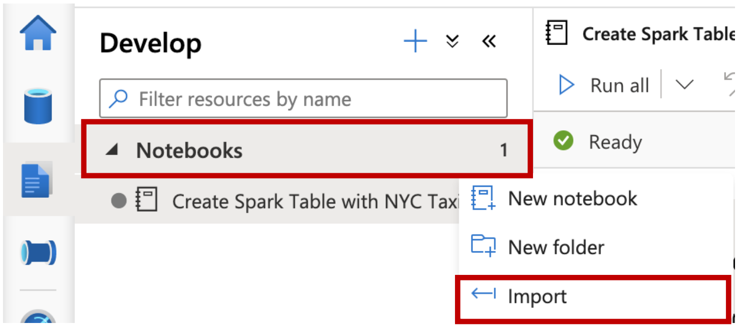
cell 2	`%%pyspark`
	`nyc_tlc_df.printSchema()`

cell 3	`%%pyspark`
	`spark.sql("CREATE DATABASE IF NOT EXISTS nyctaxi")`
	`nyc_tlc_df.write.mode("overwrite").saveAsTable("nyctaxi.trip")`

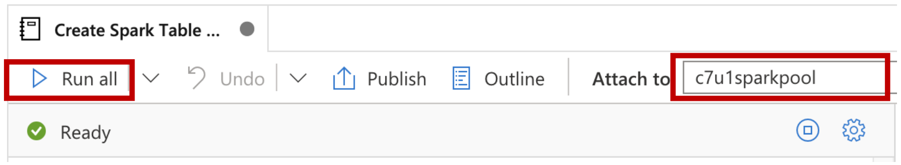
4.1.2 First, click the “Develop” and “+” and select the “Notebook”.



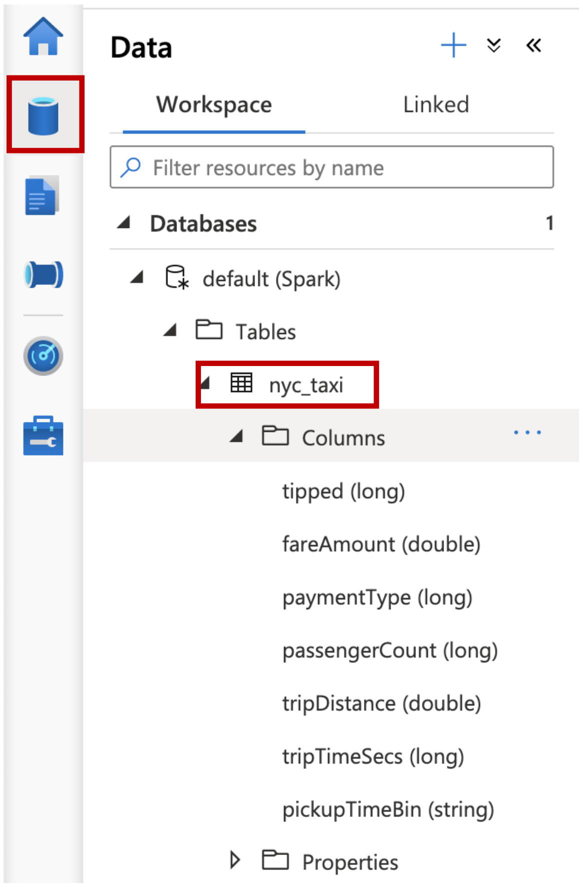
4.1.3 Import the NYC_taxidata.ipynb file from Notebooks menu in red box.



4.1.4 To run the imported code, select spark pool (c7u1sparkpool) and select "Run All".



4.1.5 When the notebook execution is completed, the basic Spark database is created and check that the nyc_taxi table is created through the "Data" in left menu.

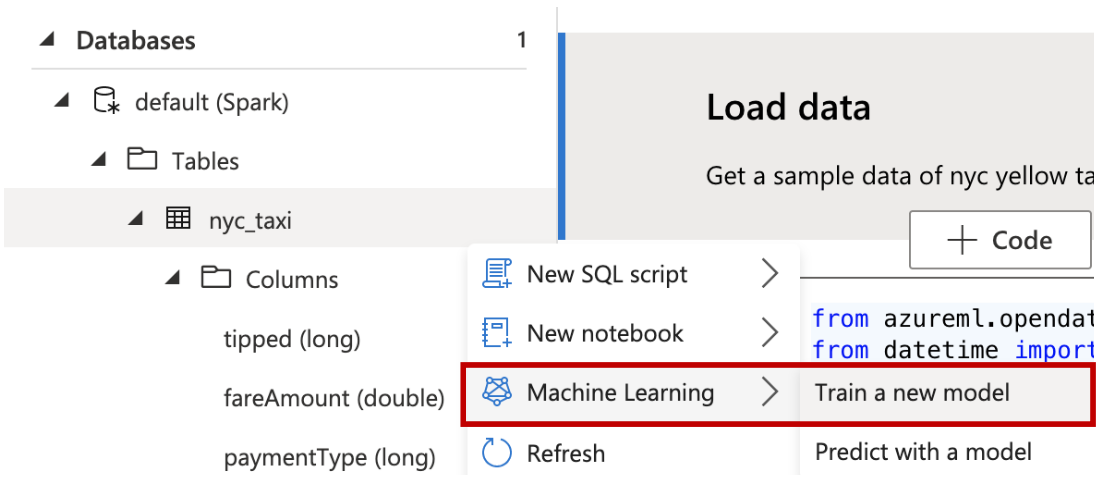


4.1.6 Verify the schema of the created table nyc_taxi.

5. Automated machine learning wizard

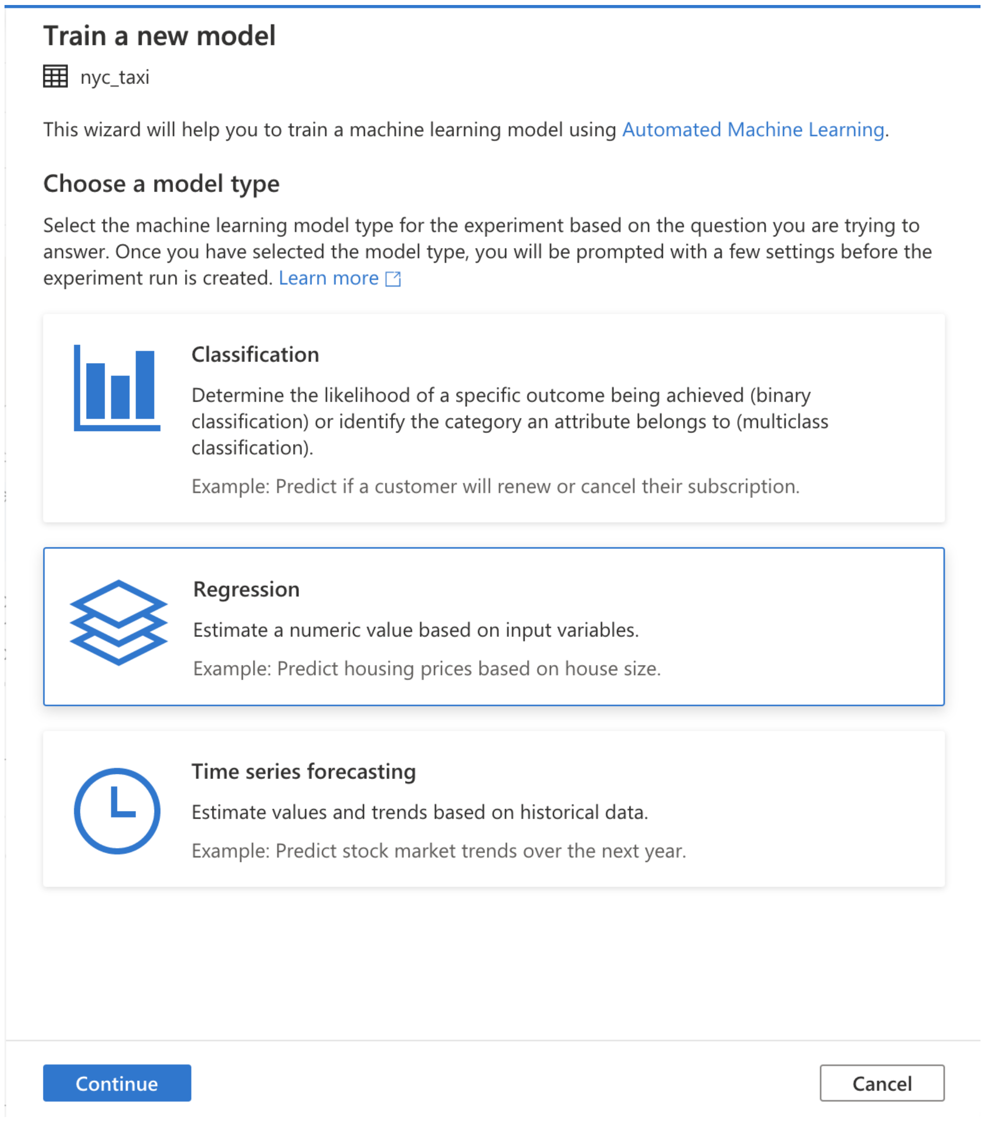
5.1 Right-click the spark table created in 4.1.4.

5.2 Select "Machine learning” and “Train a new model".



```
from azureml.opendat
from datetime import
```

5.3 Select the model to train. Here, select the regression model.



Train a new model

nyc_taxi

This wizard will help you to train a machine learning model using Automated Machine Learning.

Choose a model type

Select the machine learning model type for the experiment based on the question you are trying to answer. Once you have selected the model type, you will be prompted with a few settings before the experiment run is created. Learn more

Classification

Determine the likelihood of a specific outcome being achieved (binary classification) or identify the category an attribute belongs to (multiclass classification).

Example: Predict if a customer will renew or cancel their subscription.

Regression

Estimate a numeric value based on input variables.

Example: Predict housing prices based on house size.

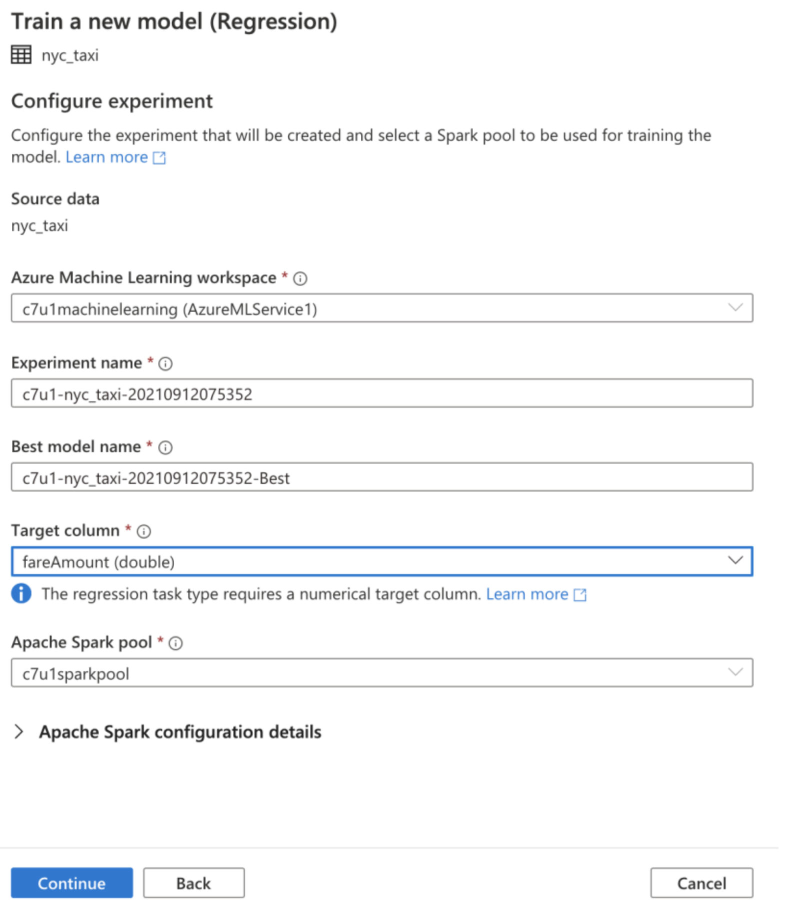
Time series forecasting

Estimate values and trends based on historical data.

Example: Predict stock market trends over the next year.

5.4 It provides configuration details for creating automated machine learning experiments that run on Azure Machine Learning.

5.4.1 Select "fareAmount" in the target column.


Train a new model (Regression)
nyc_taxi
Configure experiment
Configure the experiment that will be created and select a Spark pool to be used for training the model. Learn more
Apache Spark configuration details
Continue
Back
Cancel

Source data
nyc_taxi

Azure Machine Learning workspace *
c7u1machinelearning (AzureMLService1)

Experiment name *
c7u1-nyc_taxi-20210912075352

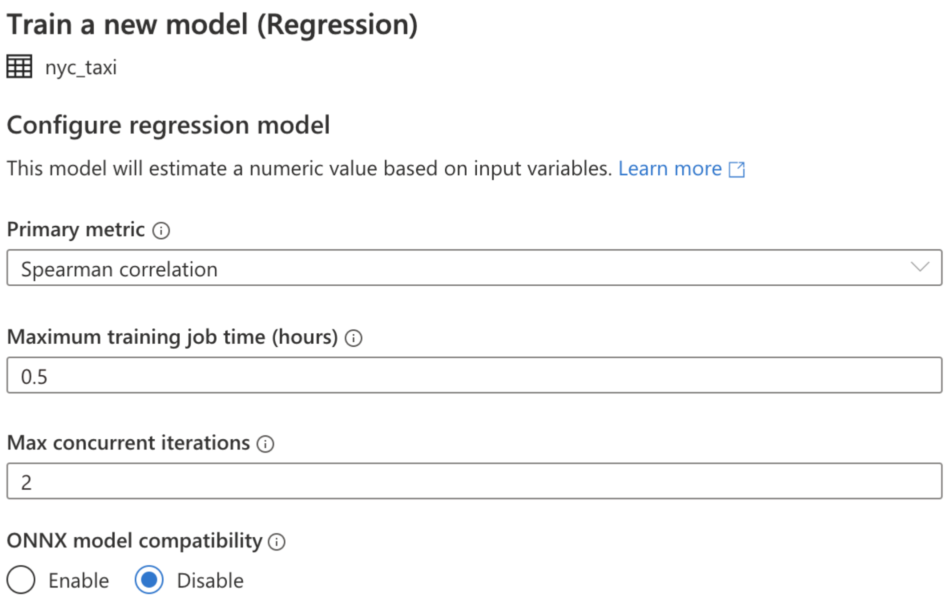
Best model name *
c7u1-nyc_taxi-20210912075352-Best

Target column *
fareAmount (double)

The regression task type requires a numerical target column. Learn more

Apache Spark pool *
c7u1sparkpool

5.4.2 Use the default values except maximum training job time (hours) for regression model, and click the “Create run” button.


Train a new model (Regression)

nyc_taxi

Configure regression model

This model will estimate a numeric value based on input variables. Learn more

Primary metric: Spearman correlation

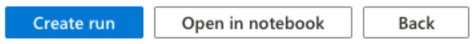
Maximum training job time (hours): 0.5

Max concurrent iterations: 2

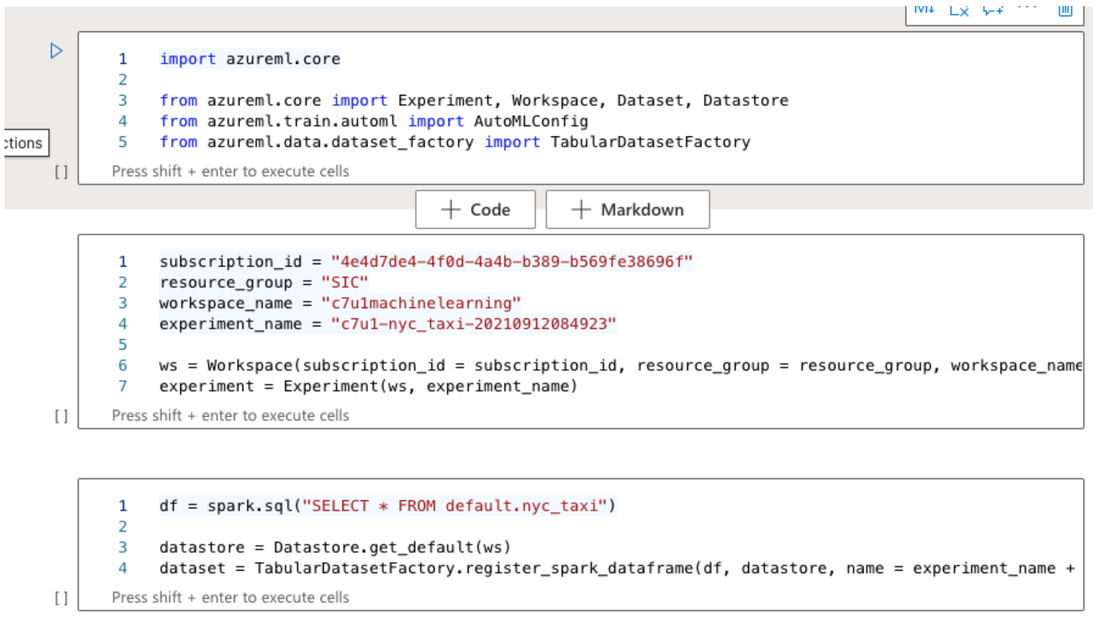
ONNX model compatibility: Enable [✓ Disable]

Note: Allocate a minimum amount of time within 30 minutes for testing.

5.4.3 After all the necessary configuration is done, you can start auto-run. You can choose to create a run that starts running directly without code. Also, if you're writing code, you can choose to open it in a notebook.



5.4.4 This option allows you to see the code that creates a run and then runs the notebook.



```
import azureml.core

from azureml.core import Experiment, Workspace, Dataset, Datastore
from azureml.train.automl import AutoMLConfig
from azureml.data.dataset_factory import TabularDatasetFactory
```

Press shift + enter to execute cells

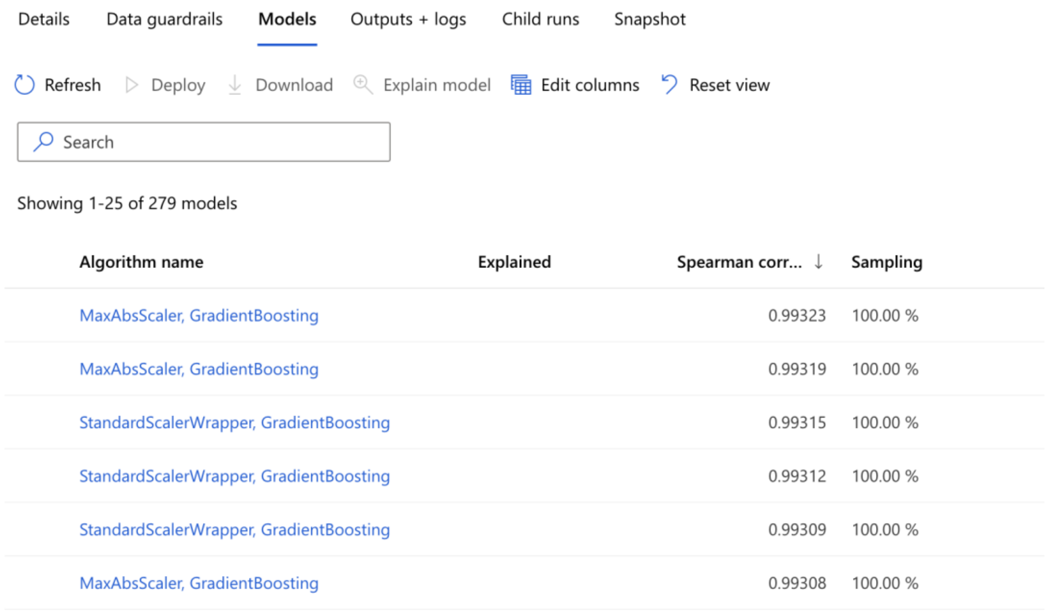
```
subscription_id = "4e4d7de4-4f0d-4a4b-b389-b569fe38696f"
resource_group = "SIC"
workspace_name = "c7u1machinelearning"
experiment_name = "c7u1-nyc_taxi-20210912084923"

ws = Workspace(subscription_id = subscription_id, resource_group = resource_group, workspace_name
experiment = Experiment(ws, experiment_name)
```

Press shift + enter to execute cells

```
df = spark.sql("SELECT * FROM default.nyc_taxi")

datastore = Datastore.get_default(ws)
dataset = TabularDatasetFactory.register_spark_dataframe(df, datastore, name = experiment_name +
```

Press shift + enter to execute cells

5.5 This trains performed several models.


Details Data guardrails Models Outputs + logs Child runs Snapshot

Refresh Deploy Download Explain model Edit columns Reset view

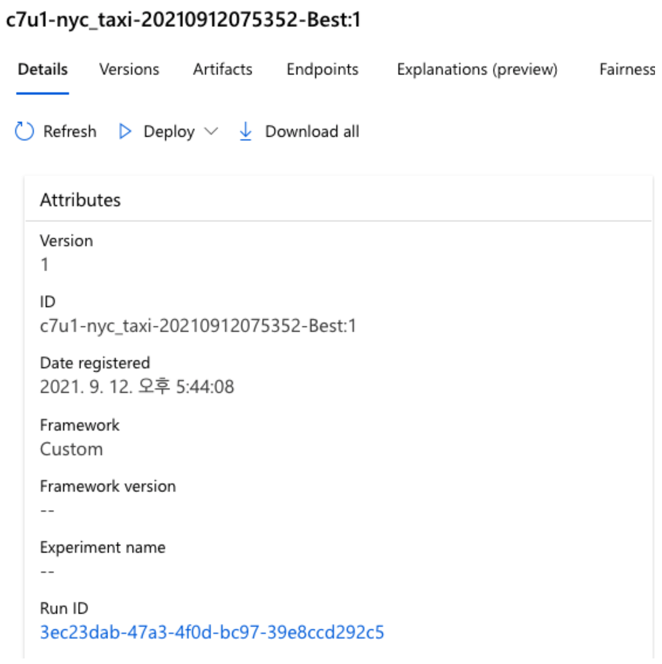
Search

Showing 1-25 of 279 models

Algorithm name	Explained	Spearman corr... ↓	Sampling
MaxAbsScaler, GradientBoosting		0.99323	100.00 %
MaxAbsScaler, GradientBoosting		0.99319	100.00 %
StandardScalerWrapper, GradientBoosting		0.99315	100.00 %
StandardScalerWrapper, GradientBoosting		0.99312	100.00 %
StandardScalerWrapper, GradientBoosting		0.99309	100.00 %
MaxAbsScaler, GradientBoosting		0.99308	100.00 %

Note: Several algorithms that have been trained are listed.

5.5.1 The best model for successful execution registers with the Azure Machine Learning model registry.


c7u1-nyc_taxi-20210912075352-Best:1

Details Versions Artifacts Endpoints Explanations (preview) Fairness

Refresh Deploy Download all

Attributes

Version
1

ID
c7u1-nyc_taxi-20210912075352-Best:1

Date registered
2021. 9. 12. 오후 5:44:08

Framework
Custom

Framework version
--

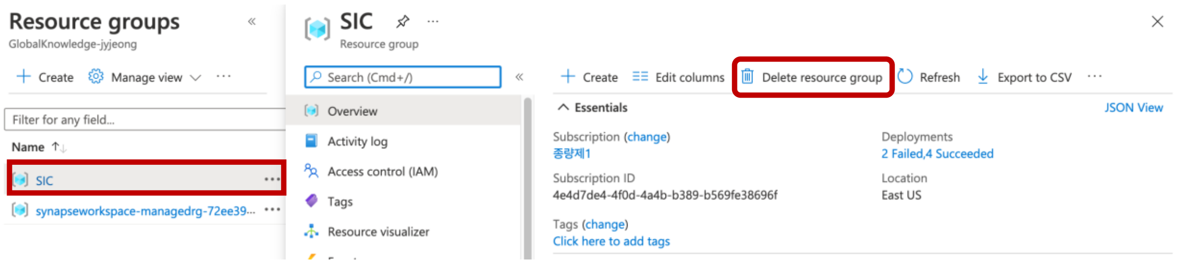
Experiment name
--

Run ID
3ec23dab-47a3-4f0d-bc97-39e8ccd292c5

6. Clean up

6.1 In this step, we will delete the resources we used in the lab.

6.1.1 Check the resource group list and delete the SIC resource group.



Subscription (change)
종량제1

Subscription ID
4e4d7de4-4f0d-4a4b-b389-b569fe38696f

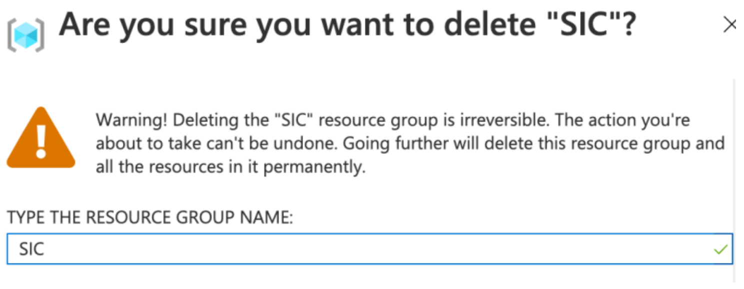
Tags (change)
Click here to add tags

Deployments
2 Failed,4 Succeeded

Location
East US

6.2 Confirm to delete the SIC resource. Type the name of resource group for confirmation.

6.2.1 Delete all other resource groups as well.



Are you sure you want to delete "SIC"?

Warning! Deleting the "SIC" resource group is irreversible. The action you're about to take can't be undone. Going further will delete this resource group and all the resources in it permanently.

TYPE THE RESOURCE GROUP NAME:
SIC

Lab 2: Working with SageMaker in AWS

In this lab, you will learn how to build, train, and deploy machine learning (ML) models using the XGBoost ML algorithm on Amazon SageMaker.

1. Create a SageMaker notebook instance

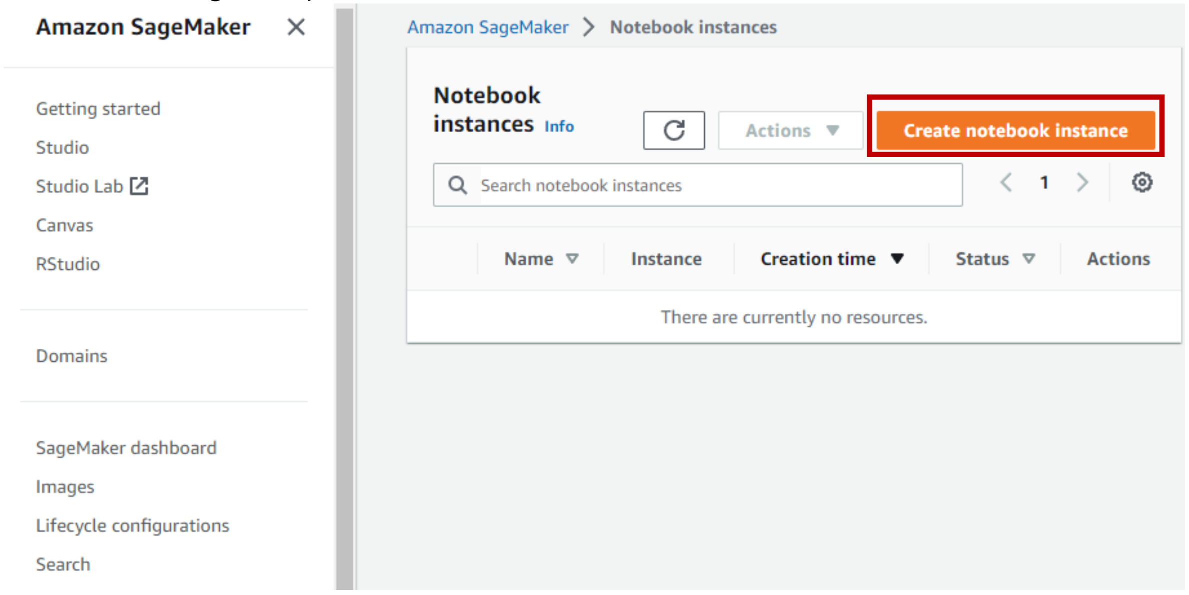
In this step, you will create a notebook instance that will be used to download and process data. As part of the creation process, you also create an Identity and Access Management (IAM) role that allows Amazon SageMaker to access data in Amazon S3.

1.1 First, log in to the AWS site (https://aws.amazon.com/). If need, please refer to the site contents here.

1.2 Prerequisites - To use AWS, you must have an account and know the basics of how to use it.

1.3 Move to https://console.aws.amazon.com/sagemaker/ for SageMake usage

1.4 In the left navigation pane, choose the Notebook, and then create Notebook Instance.



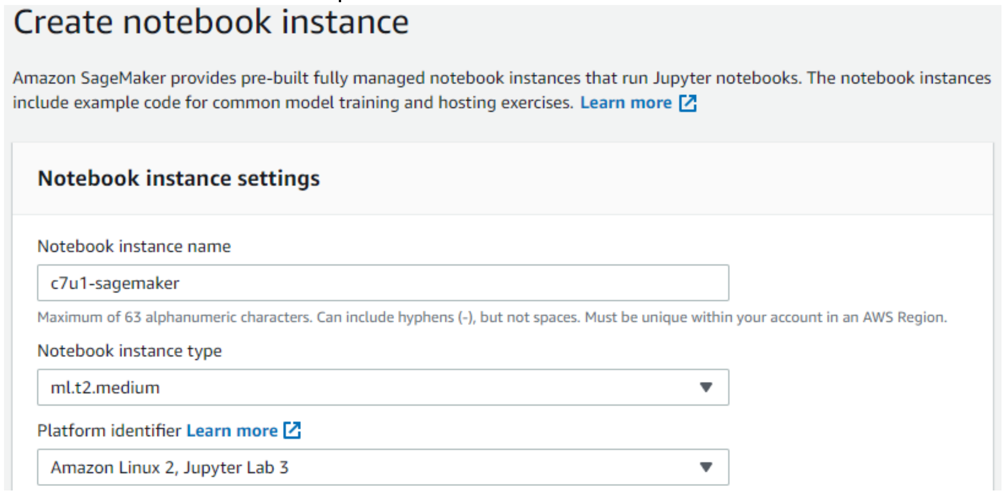
There are currently no resources.

1.5 On the Create Notebook Instance page, fill in the following fields in the Notebook Instance Settings box.

1.5.1 Notebook instance name: Enter c7u1-sagemaker.

1.5.2 Notebook Instance Type: Choose ml.t2.medium.

1.5.3 Platform identifier: Keep the default selection.



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

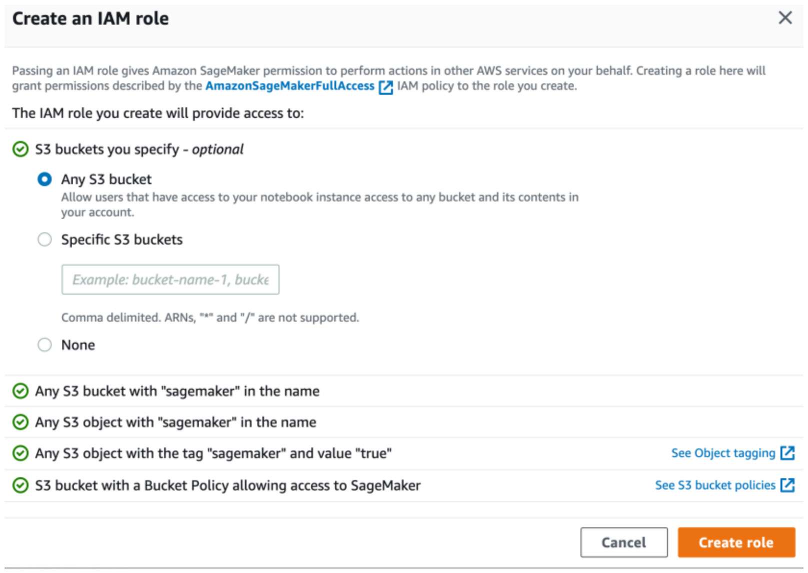
Notebook instance name: c7u1-sagemaker

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.t2.medium

Platform identifier Learn more: Amazon Linux 2, Jupyter Lab 3

1.6 In the Permissions and Encryption section, choose Create new role for IAM role, choose any S3 buckets in the Create IAM role dialog box, and then choose Create role.



Create an IAM role

Passing an IAM role gives Amazon SageMaker permission to perform actions in other AWS services on your behalf. Creating a role here will grant permissions described by the AmazonSageMakerFullAccess IAM policy to the role you create.

The IAM role you create will provide access to:

✓ S3 buckets you specify - *optional*

[✓ Any S3 bucket]
Allow users that have access to your notebook instance access to any bucket and its contents in your account.

Specific S3 buckets

Example: bucket-name-1, bucke

Comma delimited. ARNs, "*" and "/" are not supported.

None

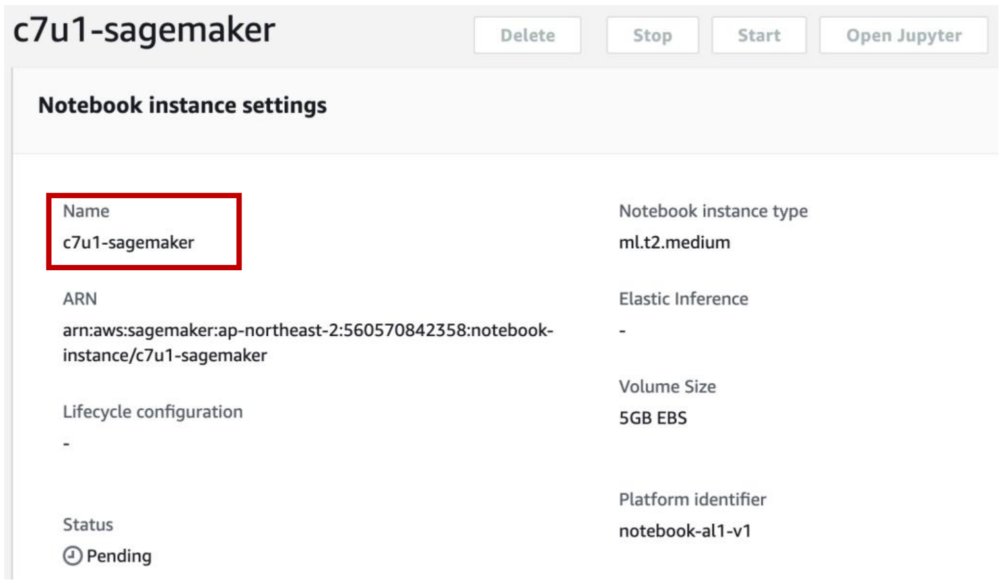
✓ Any S3 bucket with "sagemaker" in the name

✓ Any S3 object with "sagemaker" in the name

✓ Any S3 object with the tag "sagemaker" and value "true" — See Object tagging

✓ S3 bucket with a Bucket Policy allowing access to SageMaker — See S3 bucket policies

1.7 Keep the default settings for the rest of the options and choose Create Notebook Instance.


c7u1-sagemaker

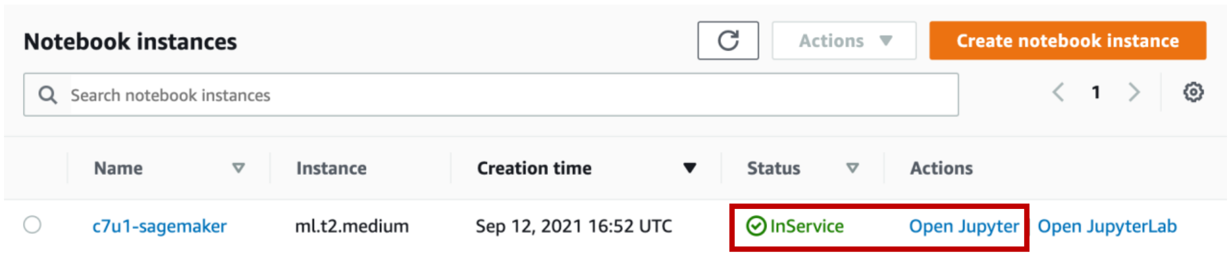
Notebook instance settings

Name: c7u1-sagemaker	Notebook instance type: ml.t2.medium
ARN: arn:aws:sagemaker:ap-northeast-2:560570842358:notebook-instance/c7u1-sagemaker	Elastic Inference: -
Lifecycle configuration: -	Volume Size: 5GB EBS
Status: Pending	Platform identifier: notebook-al1-v1

2. Prepare the data

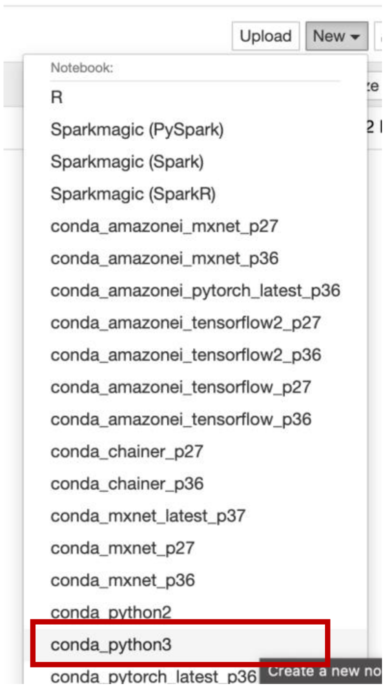
Use an Amazon SageMaker notebook instance to pre-process the data needed to train a machine learning model, and then upload the data to Amazon S3.

2.1 When the SageMaker-Tutorial notebook instance state changes to InService, choose Open Jupyter.


Notebook instances

Name	Instance	Creation time	Status	Actions
c7u1-sagemaker	ml.t2.medium	Sep 12, 2021 16:52 UTC	InService	Open Jupyter Open JupyterLab

2.2 In Jupyter, choose New, then choose conda_python3.



2.3 Copy and paste the following code into a new code cell in your Jupyter notebook and select Run

```
# import libraries
import boto3, re, sys, math, json, os, sagemaker, urllib.request
from sagemaker import get_execution_role
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Image
from IPython.display import display
from time import gmtime, strftime
# from sagemaker.predictor import csv_serializer

# Define IAM role
role = get_execution_role()
prefix = 'sagemaker/DEMO-xgboost-dm'
my_region = boto3.session.Session().region_name # set the region of the instance
```

```
# this line automatically looks for the XGBoost image URI and builds an XGBoost
container.
xgboost_container = sagemaker.image_uris.retrieve("xgboost", my_region, "latest")


print("Success - the MySageMakerInstance is in the " + my_region + " region. You will
use the " + xgboost_container + " container for your SageMaker endpoint.")
```

2.4 Copy and paste the following code into a cell and select Run.

```
bucket_name = “c7u1-sagemaker”
s3 = boto3.resource('s3')
try:
  if  my_region == 'us-east-1':
    s3.create_bucket(Bucket=bucket_name)
  else:
    s3.create_bucket(Bucket=bucket_name,
CreateBucketConfiguration={ 'LocationConstraint': my_region })
  print('S3 bucket created successfully')
except Exception as e:
  print('S3 error: ',e)
```

Note: Replace bucket_name your-s3-bucket-name.(c7u1-sagemaker) If you do not receive a success message after running the code, change the bucket name and try again.

2.5 Downloads data to a SageMaker instance and loads data into a dataframe.

```
try:
 urllib.request.urlretrieve ("https://d1.awsstatic.com/tmt/build-train-deploy-machine-
learning-model-
sagemaker/bank_clean.27f01fbbdf43271788427f3682996ae29ceca05d.csv",
"bank_clean.csv")
 print('Success: downloaded bank_clean.csv.')
except Exception as e:
 print('Data load error: ',e)
try:
 model_data = pd.read_csv('./bank_clean.csv',index_col=0)
 print('Success: Data loaded into dataframe.')
  except Exception as e:
   print('Data load error: ',e)
```

2.6 Shuffle and split the data into training and test data. Copy and paste the following code into the next code cell and select Run.

```
train_data, test_data = np.split(model_data.sample(frac=1, random_state=1729),
[int(0.7 * len(model_data))])
print(train_data.shape, test_data.shape)
```

3. Train the model to learn from the data

In this step, you train a machine learning model using the training dataset.

3.1 The following reformats the header and first column of the training data and then loads the data from the S3 bucket. This step is required to use the Amazon SageMaker pre-built XGBoost algorithm.

```
pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)],
axis=1).to_csv('train.csv', index=False, header=False)
boto3.Session().resource('s3').Bucket(bucket_name).Object(os.path.join(prefix,
'train/train.csv')).upload_file('train.csv')
s3_input_train =
sagemaker.inputs.TrainingInput(s3_data='s3://{}/{}/train'.format(bucket_name,
prefix), content_type='csv')
```

3.2 Set up an Amazon SageMaker session, create an instance of the XGBoost model (estimator), and define the model's hyperparameters.

```
sess = sagemaker.Session()
```

```
xgb = sagemaker.estimator.Estimator(xgboost_container,role, instance_count=1,
instance_type='ml.m4.xlarge',output_path='s3://{}/{}/output'.format(bucket_name,
prefix),sagemaker_session=sess)

xgb.set_hyperparameters(max_depth=5,eta=0.2,gamma=4,min_child_weight=6,subsam
ple=0.8,silent=0,objective='binary:logistic',num_round=100)
```

3.3 Start training work. This code trains a model using gradient optimization on a ml.m4.xlarge instance. After a few minutes, you should see the training log being created in your Jupyter notebook.

```
xgb.fit({'train': s3_input_train})
```

4. Deploy the model

In this step, you deploy the trained model to the endpoint, reformat and load the CSV data, and then run the model to generate predictions.

4.1 This code deploys the model to the server and creates a SageMaker endpoint that you can access. This step may take a few minutes to complete.

```
xgb_predictor = xgb.deploy(initial_instance_count=1,instance_type='ml.m4.xlarge')
```

4.2 To predict whether a customer in your test data is registered with a bank product, copy the following code into a cell and select Run.

```
from sagemaker.serializers import CSVSerializer


test_data_array = test_data.drop(['y_no', 'y_yes'], axis=1).values #load the data into an
array

xgb_predictor.serializer = CSVSerializer() # set the serializer type

predictions = xgb_predictor.predict(test_data_array).decode('utf-8') # predict!

predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into
an array

print(predictions_array.shape)
```

5. Evaluate your ML model's performance

In this step, you evaluate the performance and accuracy of your machine learning model.

5.1 The next code compares the actual and predicted values in a table called a confusion matrix.

```
cm = pd.crosstab(index=test_data['y_yes'], columns=np.round(predictions_array),
rownames=['Observed'], colnames=['Predicted'])

tn = cm.iloc[0,0]; fn = cm.iloc[1,0]; tp = cm.iloc[1,1]; fp = cm.iloc[0,1]; p =
(tp+tn)/(tp+tn+fp+fn)*100

print("\n{0:<20}{1:<4.1f}%\n".format("Overall Classification Rate: ", p))

print("{0:<15}{1:<15}{2:>8}".format("Predicted", "No Purchase", "Purchase"))

print("Observed")

print("{0:<15}{1:<2.0f}% ({2:<}){3:>6.0f}% ({4:<})".format("No Purchase",
tn/(tn+fn)*100,tn, fp/(tp+fp)*100, fp))

print("{0:<16}{1:<1.0f}% ({2:<}){3:>7.0f}% ({4:<}) \n".format("Purchase",
fn/(tn+fn)*100,fn, tp/(tp+fp)*100, tp))
```

6. Clean up the resources

This step delete the resources used in this lab.

6.1 Delete endpoint: Copy and paste the following code from your Jupyter notebook and select Run.

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

6.2 Delete training artifacts and S3 bucket : Copy and paste the following code from your Jupyter notebook and choose Run.

```
bucket_to_delete = boto3.resource('s3').Bucket(bucket_name)

bucket_to_delete.objects.all().delete()
```

6.3 Delete SageMaker Notebook: Stops and deletes SageMaker Notebook.

6.3.1 Open the SageMaker console.

6.3.2 In Notebooks, select the notebook instance.

6.3.3 Select the notebook instance you created for this lab, then choose Actions, Stop.

6.3.4 It may take up to several minutes for the notebook instance to stop. When the status changes to Stopped, go to the next step.

6.3.5 Choose an action, then choose Delete.

6.3.6 Choose Delete.

Lab 3: Build a pipeline Machine Learning application

In this lab, you will learn how to build a machine learning algorithm using Spark MLlib.

You will use the Jupyter notebook to build and test the application. The application uses the data file that it is located in local Data folder. (It uses apache spark site (http://spark.apache.org) data and samples)

1. Start the pyspark in terminal.
   1.1 To run jupyter notebook, type pyspark in the terminal.

```
$ pyspark
```

   1.2 To begin the lab, import the types required for Machine Learning pipeline and LogisticRegression.

   1.3 Prepare training documents form a list of (id, text, label) tuples as snippet code.

      1.3.1 To do so, paste the following code snippet in a cell and press SHIFT + ENTER. Pressing SHIFT + ENTER executes the entry in the current cell and moves the cursor to the next cell.

```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

spark = SparkSession.builder.getOrCreate()
training = spark.createDataFrame([
   (0, "This is a testing for spark", 1.0),
   (1, "kudu ozone", 0.0),
   (2, "spark is in-memory engine", 1.0),
   (3, "hive is data warehouse", 0.0),
   (4, "hadoop is mapreduce for batch", 0.0)
], ["id", "text", "label"])
```

2. Configure the Spark machine learning pipeline.

   2.1 This Spark machine learning pipeline consists of three stages.

      2.1.1 The tokenizer Transformer stage takes in raw text and converts them to words DataFrame.

      2.1.2 The hashingTF Transformer stage takes those words and creates a feature vector DataFrame.

      2.1.3 Finally, the logistic regression Estimator takes in the feature vectors and fits them to create a new model – which is a Transformer.

```
# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

3. Fit the pipeline to the training document and verify the training data.

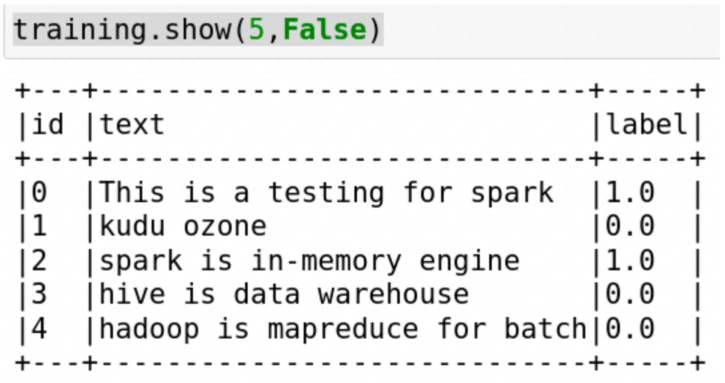
   3.1 Call the pipeline fit method to begin fitting the training document over the pipeline that was set up in the previous step.

```
model = pipeline.fit(training)
```

   3.2 Verify the training document to checkpoint the application progress.

```
training.show(5,False)
```

   3.3 This should give output similar to the following:



```
training.show(5,False)

+---+----------------------------+-----+
|id |text                        |label|
+---+----------------------------+-----+
|0  |This is a testing for spark |1.0  |
|1  |kudu ozone                  |0.0  |
|2  |spark is in-memory engine   |1.0  |
|3  |hive is data warehouse      |0.0  |
|4  |hadoop is mapreduce for batch|0.0  |
+---+----------------------------+-----+
```

3.4 Compare this output against the list of tuples that was prepared in an earlier step. In our training, the lable for id 0 and 3 should be 1.0, and the rest is set to 0.0. Because the id 0 and 3 contain spark.

4. Prepare data set to run against trained model.

   4.1 The model accepts a id and text as input.

   4.2 The model predicts whether a sentence with ID and text contains spark (1.0) or not (0.0).

```
Document = Row("id", "text")
test = sc.parallelize([(5, "K O 1"),
  (6, "spark hadoop spark impala"),
  (7, "apache spark open-source"),
  (8, "spark is a platform"),
  (9, "Hadoop is for Big Data")]).map(lambda x: Document(*x)).toDF()
```

5. Make prediction on test data.

   5.1 model is the new Transformer that resulted from executing the pipeline.

   5.2 We will now use the transform method of the newly created predictor model against the test data.

```
# Make predictions on test documents and print columns of interest
prediction = model.transform(test)
selected = prediction.select("id", "text", "probability", "prediction")
for row in selected.collect():
  rid, text, prob, prediction = row
  print(
    "(%d, %s) --> prob=%s, prediction=%f" % (
      rid, text, str(prob), prediction
    )
  )
```

5.3 You can see output similar to the following:

```
(5, K O 1) --> prob=[0.760510559377351,0.23948944062264899], prediction=0.000000
(6, spark hadoop spark impala) --> prob=[0.09933562532409944,0.9006643746759005], prediction=1.000000
(7, apache spark open-source) --> prob=[0.19380944053253446,0.8061905594674655], prediction=1.000000
(8, spark is a platform) --> prob=[0.03587540471389246,0.9641245952861075], prediction=1.000000
(9, Hadoop is for Big Data) --> prob=[0.9907289093358328,0.009271090664167203], prediction=0.000000
```

Lab 4: Build Logistic Regression Model

In this lab, you will learn how to build a Logistic regression model on a Spark cluster. You will use the Jupyter notebook to build and test the application.

1. Import the types required for ML for Vectors and LogisticRegression.

   1.1 Prepare the training data from list of tuples.

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

#training data with label and features
training = spark.createDataFrame([
   (1.0, Vectors.dense([0.0, 1.1, 0.1])),
   (0.0, Vectors.dense([2.0, 1.0, -1.0])),
   (0.0, Vectors.dense([2.0, 1.3, 1.0])),
   (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])
```

2. Create a model and verify the training data.

   2.1 Create a instance and LogisticRegreesion model with the parameters stored in lr.

```
lr = LogisticRegression(maxIter=10, regParam=0.01)
Model1=lr.fit(training)
```

Note: This instance is an Estimator.

   2.2 Verify the training document to checkpoint the application progress.

```
training.show(10,False)
```

2.3 This should give output similar to the following:

```
+-----+--------------+
|label|features      |
+-----+--------------+
|1.0  |[0.0,1.1,0.1] |
|0.0  |[2.0,1.0,-1.0]|
|0.0  |[2.0,1.3,1.0] |
|1.0  |[0.0,1.2,-0.5]|
+-----+--------------+
```

2.4 The model1 is a transformer produced by an Estimator.

3. Make a new model.

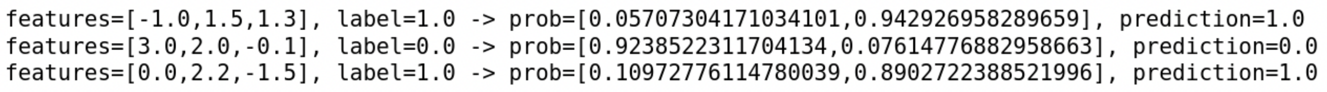
3.1 A new model using the paramMapCombined parameters. The paramMap will be used a Python dictionary as parameters.

```
paramMap = {lr.maxIter: 20}
paramMap[lr.maxIter] = 30  # Specify 1 Param, overwriting the original maxIter.
# Specify multiple Params.
paramMap.update({lr.regParam: 0.1, lr.threshold: 0.55})
paramMap2 = {lr.probabilityCol: "myProbability"}  # type: ignore
paramMapCombined = paramMap.copy()
paramMapCombined.update(paramMap2)  # type: ignore

# paramMapCombined overrides all parameters set earlier via lr.set* methods.
model2 = lr.fit(training, paramMapCombined)
```

4. Prepare data set to run against trained model. And make a prediction on test data.

```
 test = spark.createDataFrame([
    (1.0, Vectors.dense([-1.0, 1.5, 1.3])),
    (0.0, Vectors.dense([3.0, 2.0, -0.1])),
    (1.0, Vectors.dense([0.0, 2.2, -1.5]))], ["label", "features"])
# Make predictions on test data using the Transformer.transform() method.
# LogisticRegression.transform will only use the 'features' column.
# Note that model2.transform() outputs a "myProbability" column instead of the usual
# 'probability' column since we renamed the lr.probabilityCol parameter previously.
prediction = model2.transform(test)
result = prediction.select("features", "label", "myProbability", "prediction") \
    .collect()

for row in result:
    print("features=%s, label=%s -> prob=%s, prediction=%s"
          % (row.features, row.label, row.myProbability, row.prediction))
```

4.1 You can see output similar to the following:



```
features=[-1.0,1.5,1.3], label=1.0 -> prob=[0.05707304171034101,0.942926958289659], prediction=1.0
features=[3.0,2.0,-0.1], label=0.0 -> prob=[0.9238522311704134,0.07614776882958663], prediction=0.0
features=[0.0,2.2,-1.5], label=1.0 -> prob=[0.10972776114780039,0.8902722388521996], prediction=1.0
```

Lab 5: Build a Clustering Application

In this lab, You will learn how to build a KMeans Clustering algorithm on a Spark cluster.

1. import the types required for this lab.

   1.1 To do so, paste the following code snippet in a cell and press SHIFT + ENTER. Pressing SHIFT + ENTER executes the entry in the current cell and moves the cursor to the next cell.

```
from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator
```

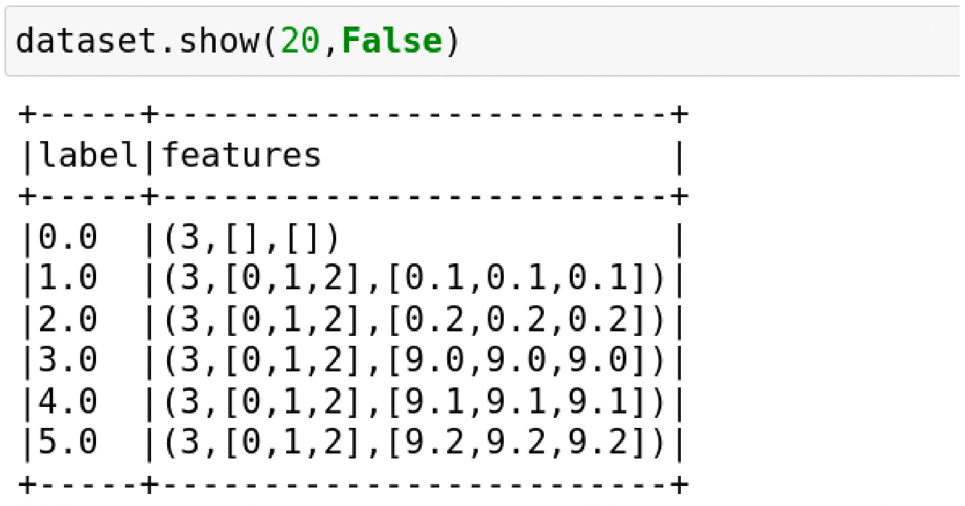
2. Load the Input data example data set and cache it to memory.

```
#load the data.

dataset =
spark.read.format("libsvm").load("file:///home/student/Data/sample_kmeans_data.txt")
```

   2.1 Verify the training document to checkpoint the application progress.

```
dataset.show(10,False)
```

   2.2 This should give output similar to the following:



```
dataset.show(20,False)

+-----+-------------------------+
|label|features                 |
+-----+-------------------------+
|0.0  |(3,[],[])                |
|1.0  |(3,[0,1,2],[0.1,0.1,0.1])|
|2.0  |(3,[0,1,2],[0.2,0.2,0.2])|
|3.0  |(3,[0,1,2],[9.0,9.0,9.0])|
|4.0  |(3,[0,1,2],[9.1,9.1,9.1])|
|5.0  |(3,[0,1,2],[9.2,9.2,9.2])|
+-----+-------------------------+
```

3. Make the KMeans model with the features and prediction column

```
# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)
```

4. Make a prediction on dataset.
   4.1 We will use the transform method of the newly created predictor model against dataset.

```
# Make predictions
predictions = model.transform(dataset)
```

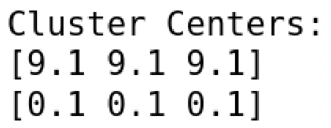
5. Run the test data set
   5.1 Based on the input data modified, we develop a model for the clustering algorithm.
   5.2 However, you train and test the model with the same data set. Test a dataset actually should be another dataset in order to check out the accuracy of the model.
   5.3 Evaluate clustering by computing Silhouette score

```
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

   5.4 Show the result.

```
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
  print(center)
```

5.4.1 You can see output similar to the following:



```
Cluster Centers:
[9.1 9.1 9.1]
[0.1 0.1 0.1]
```

Lab 6: Build Linear Regression Model

In this lab, you will learn how to build a linear regression model on a Spark cluster. You will use the Jupyter notebook to build and test the application.

The Boston Housing Dataset is a popular machine learning dataset on Kaggle. The dataset is derived from information collected by the U.S. Census Service concerning housing in the Boston, MA area. The dataset contains the following columns:

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

The goal is to predict MEDV, the median value of owner-occupied homes.

1. Construct Source DataFrame

   1.1 Copy housing.csv from /home/student/Data to your local HDFS directory

   1.2 Use spark.read() to create a dataframe from housing.csv. The CSV file has a header row. Make sure to set the option for the header. Also, set the option to infer the schema. Inspect the schema of the new dataframe. Does it conform to the following?

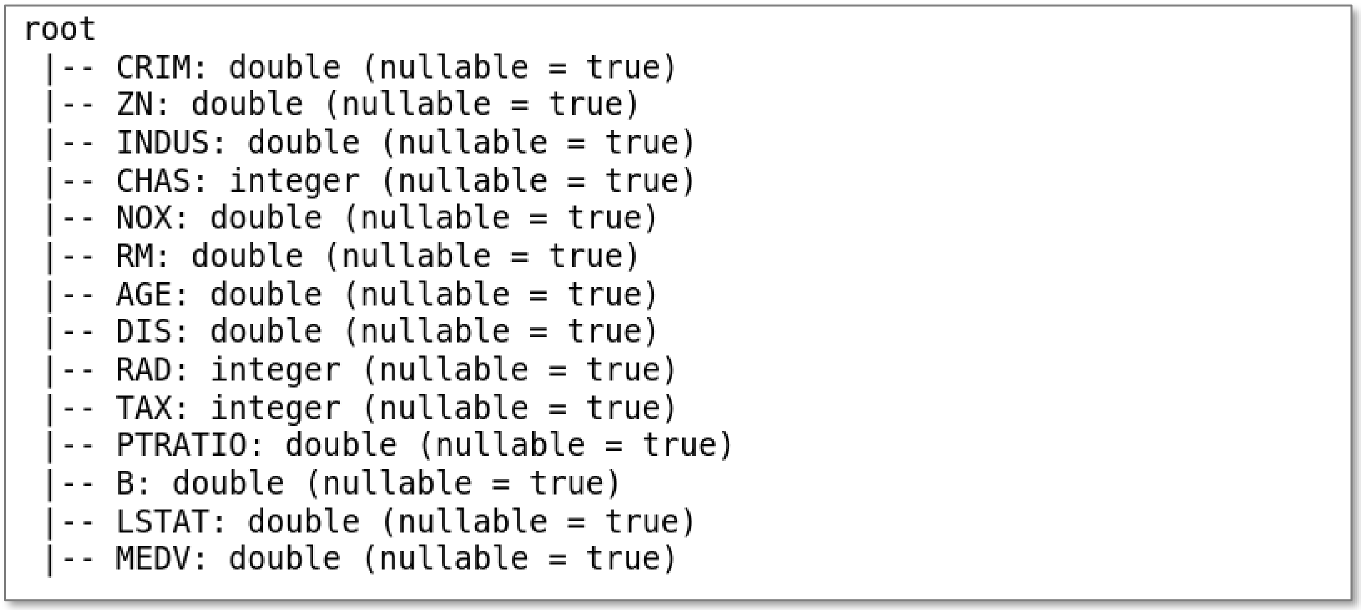
- CRIM - double
- ZN - double
- INDUS - double
- CHAS - integer
- NOX - double

- RM - double
- AGE - double
- DIS - double
- RAD - integer
- TAX - integer
- PTRATIO - double
- B - double
- LSTAT - double
- MEDV - double

1.3 Unfortunately, Spark did not infer the schema correctly. We will have to manually create a schema.

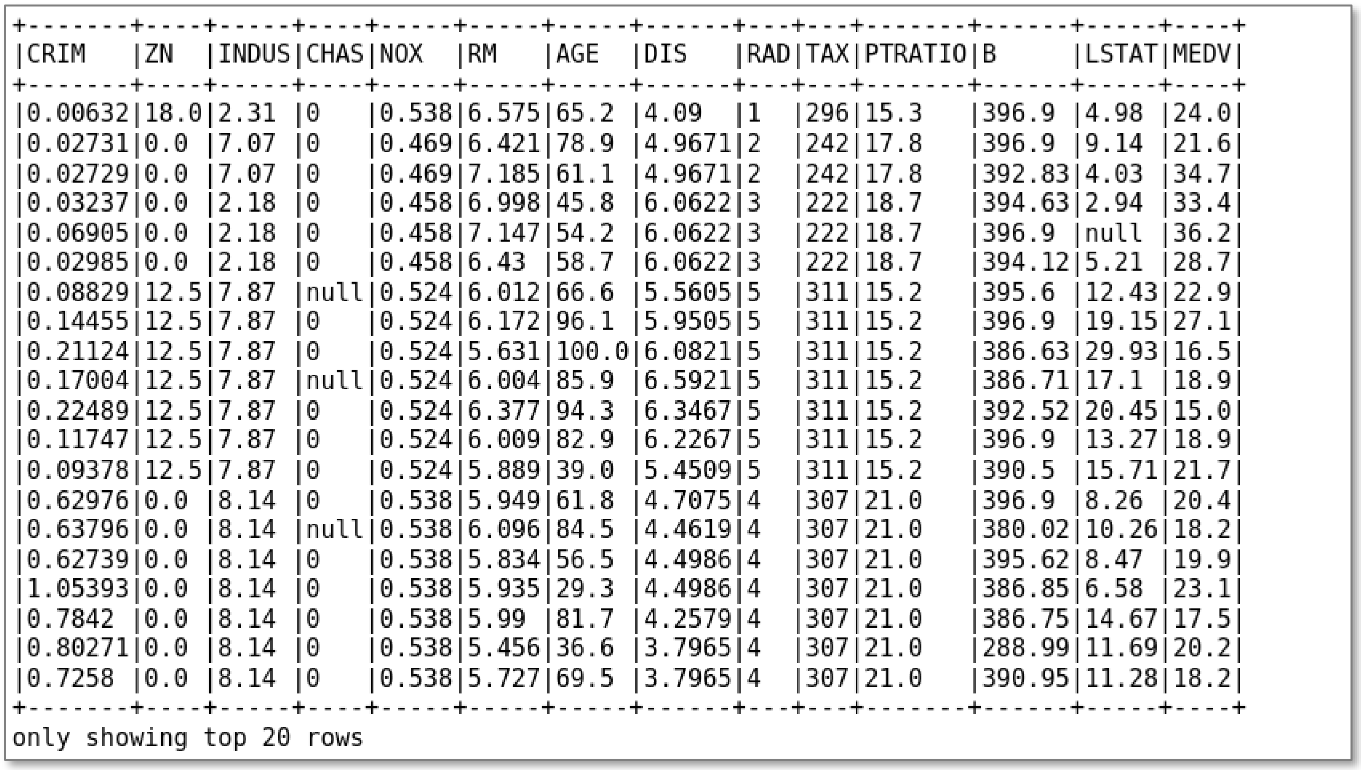
1.3.1 Create the above schema using SructType() and StructField().. Don't forget to import the necessary SQL types from pyspark.sql.types

1.3.2 Inspect the dataframe using printSchema(). Make sure the schema is now correct. Your output should be similar to below.



```
root
 |-- CRIM: double (nullable = true)
 |-- ZN: double (nullable = true)
 |-- INDUS: double (nullable = true)
 |-- CHAS: integer (nullable = true)
 |-- NOX: double (nullable = true)
 |-- RM: double (nullable = true)
 |-- AGE: double (nullable = true)
 |-- DIS: double (nullable = true)
 |-- RAD: integer (nullable = true)
 |-- TAX: integer (nullable = true)
 |-- PTRATIO: double (nullable = true)
 |-- B: double (nullable = true)
 |-- LSTAT: double (nullable = true)
 |-- MEDV: double (nullable = true)
```

1.4 Print out 20 lines and make sure to disable truncation. Your output should look similar to below:



```
+-------+----+-----+----+-----+-----+-----+------+---+---+-------+------+-----+----+
|CRIM   |ZN  |INDUS|CHAS|NOX  |RM   |AGE  |DIS   |RAD|TAX|PTRATIO|B     |LSTAT|MEDV|
+-------+----+-----+----+-----+-----+-----+------+---+---+-------+------+-----+----+
|0.00632|18.0|2.31 |0   |0.538|6.575|65.2 |4.09  |1  |296|15.3   |396.9 |4.98 |24.0|
|0.02731|0.0 |7.07 |0   |0.469|6.421|78.9 |4.9671|2  |242|17.8   |396.9 |9.14 |21.6|
|0.02729|0.0 |7.07 |0   |0.469|7.185|61.1 |4.9671|2  |242|17.8   |392.83|4.03 |34.7|
|0.03237|0.0 |2.18 |0   |0.458|6.998|45.8 |6.0622|3  |222|18.7   |394.63|2.94 |33.4|
|0.06905|0.0 |2.18 |0   |0.458|7.147|54.2 |6.0622|3  |222|18.7   |396.9 |null |36.2|
|0.02985|0.0 |2.18 |0   |0.458|6.43 |58.7 |6.0622|3  |222|18.7   |394.12|5.21 |28.7|
|0.08829|12.5|7.87 |null|0.524|6.012|66.6 |5.5605|5  |311|15.2   |395.6 |12.43|22.9|
|0.14455|12.5|7.87 |0   |0.524|6.172|96.1 |5.9505|5  |311|15.2   |396.9 |19.15|27.1|
|0.21124|12.5|7.87 |0   |0.524|5.631|100.0|6.0821|5  |311|15.2   |386.63|29.93|16.5|
|0.17004|12.5|7.87 |null|0.524|6.004|85.9 |6.5921|5  |311|15.2   |386.71|17.1 |18.9|
|0.22489|12.5|7.87 |0   |0.524|6.377|94.3 |6.3467|5  |311|15.2   |392.52|20.45|15.0|
|0.11747|12.5|7.87 |0   |0.524|6.009|82.9 |6.2267|5  |311|15.2   |396.9 |13.27|18.9|
|0.09378|12.5|7.87 |0   |0.524|5.889|39.0 |5.4509|5  |311|15.2   |390.5 |15.71|21.7|
|0.62976|0.0 |8.14 |0   |0.538|5.949|61.8 |4.7075|4  |307|21.0   |396.9 |8.26 |20.4|
|0.63796|0.0 |8.14 |null|0.538|6.096|84.5 |4.4619|4  |307|21.0   |380.02|10.26|18.2|
|0.62739|0.0 |8.14 |0   |0.538|5.834|56.5 |4.4986|4  |307|21.0   |395.62|8.47 |19.9|
|1.05393|0.0 |8.14 |0   |0.538|5.935|29.3 |4.4986|4  |307|21.0   |386.85|6.58 |23.1|
|0.7842 |0.0 |8.14 |0   |0.538|5.99 |81.7 |4.2579|4  |307|21.0   |386.75|14.67|17.5|
|0.80271|0.0 |8.14 |0   |0.538|5.456|36.6 |3.7965|4  |307|21.0   |288.99|11.69|20.2|
|0.7258 |0.0 |8.14 |0   |0.538|5.727|69.5 |3.7965|4  |307|21.0   |390.95|11.28|18.2|
+-------+----+-----+----+-----+-----+-----+------+---+---+-------+------+-----+----+
only showing top 20 rows
```

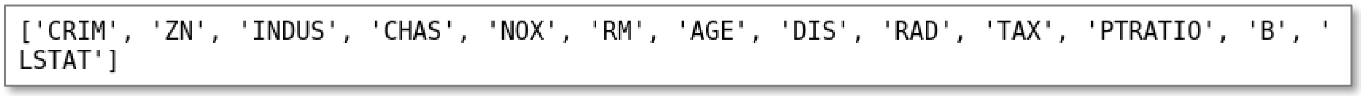
2. Data preprocessing

   2.1 Notice from the printout above that several rows contain null values for the CHAS column. This column is 1 if the property borders Charles river and 0 otherwise. There doesn't seem to be a good way to fill in this information from other rows. In this case, we will have to drop all rows that contain a null value. The Spark command to do so is the following:

```
dataframe.na.drop()
```

   2.2 Create a list containing the column names of all the columns except MEDV. MEDV is our label column.

      2.2.1 You can use whatever method you choose including manually creating the list. However, keep in mind that dataframe.columns() API returns a list of all columns. If you print the list, the output should be similar to below:



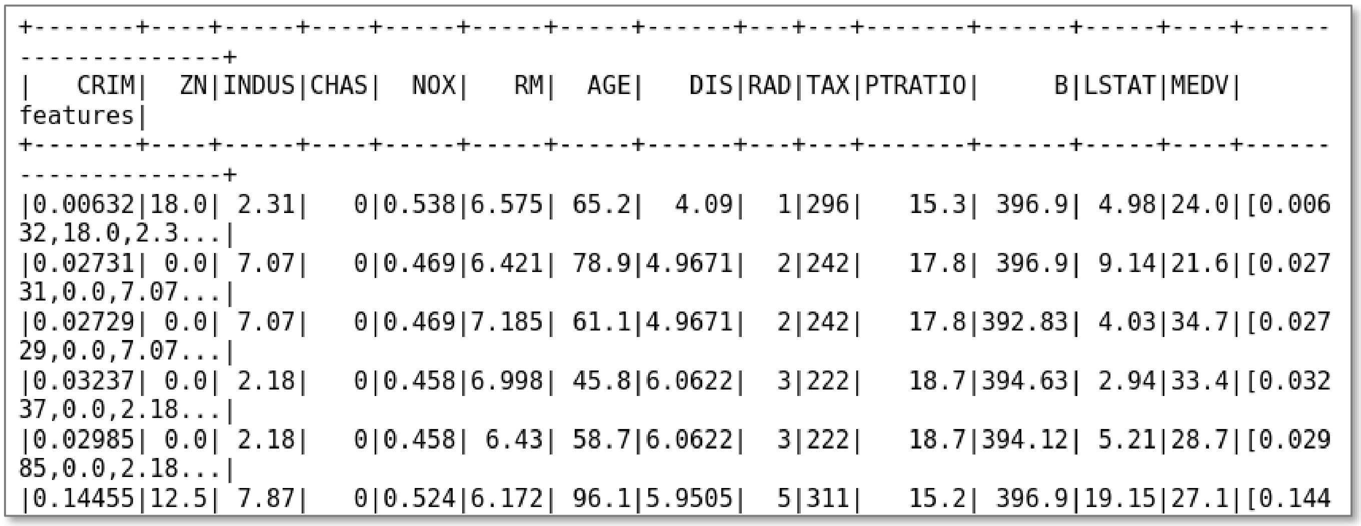
```
['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', '
LSTAT']
```

   2.3 Create a Transformer to assemble the feature columns as a Vector Column. Use the pyspark.ml.feature.VectorAssembler() API to accomplish this.

https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.ml.feature.VectorAssembler.html

```
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssemble(inputCols=<list of feature column names>,
                                   outputCol=<name of new vector column>)
```

2.4 Now use the assembler's transform(<dataframe>) method to transform the dataframe from above and create a new dataframe. The new dataframe will have an additional column with the name you specified above as the outputCol. Print out the newly transformed dataframe. It should look similar to below:



```
+-------+----+-----+----+-----+-----+-----+------+---+---+-------+------+-----+----+------
--------------+
|   CRIM|  ZN|INDUS|CHAS|  NOX|   RM|  AGE|   DIS|RAD|TAX|PTRATIO|     B|LSTAT|MEDV|
features|
+-------+----+-----+----+-----+-----+-----+------+---+---+-------+------+-----+----+------
--------------+
|0.00632|18.0| 2.31|   0|0.538|6.575| 65.2|  4.09|  1|296|   15.3| 396.9| 4.98|24.0|[0.006
32,18.0,2.3...|
|0.02731| 0.0| 7.07|   0|0.469|6.421| 78.9|4.9671|  2|242|   17.8| 396.9| 9.14|21.6|[0.027
31,0.0,7.07...|
|0.02729| 0.0| 7.07|   0|0.469|7.185| 61.1|4.9671|  2|242|   17.8|392.83| 4.03|34.7|[0.027
29,0.0,7.07...|
|0.03237| 0.0| 2.18|   0|0.458|6.998| 45.8|6.0622|  3|222|   18.7|394.63| 2.94|33.4|[0.032
37,0.0,2.18...|
|0.02985| 0.0| 2.18|   0|0.458| 6.43| 58.7|6.0622|  3|222|   18.7|394.12| 5.21|28.7|[0.029
85,0.0,2.18...|
|0.14455|12.5| 7.87|   0|0.524|6.172| 96.1|5.9505|  5|311|   15.2| 396.9|19.15|27.1|[0.144
```

2.5 Split the data into the training and test set.

2.5.1 Spark MLlib has a nice method that randomly splits the dataset. The syntax is dataframe.randomSplit(<list of doubles as weigh>). Use a weight of 0.7 and 0.3 to split the data into 70% training and 30% test data.

```
weights = [0.7, 0.3]
train, test = df.randomSplit(weights)
```

2.5.2 Print out a few lines of train and test. They are both dataframes. Also count the number of elements in each.

3. Create the linear regression model

3.1 Use the LinearRegression class from pyspark.ml.regression to create a new linear regression Estimator. Refer to the syntax below.

https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.ml.regression.LinearRegression.html?highlight=linearregression#pyspark.ml.regression.LinearRegression

In our lab, the featuresCol will be the name of the outputCol you created in step 2.3 above. It is the "features" column that was added by the VectorAssembler Transformer.

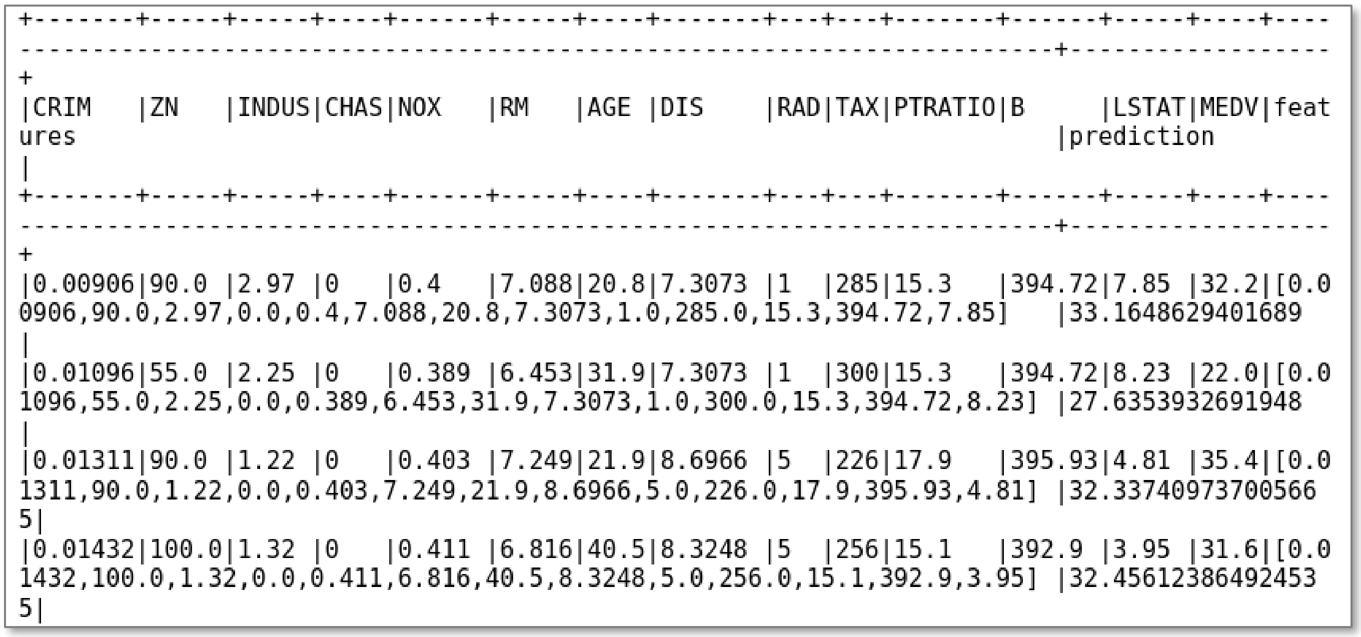
The labelCol is "MEDV" This is the median value of owner-occupied homes that we are trying to predict.

3.2 Use the fit(<training set dataframe>) method of the linear regression Estimator your created in above step. This will create a new linear regression model based on the training data set.

4. Test the linear regression model.

4.1 Use the transform(<test dataframe>) method of the model that you created in above step to create predictions. The test dataframe was created in step 2.5 above.

4.2 Print out a few lines of the predictions dataframe created. Your output should look similar to below:



```
+-------+-----+-----+----+------+-----+----+-------+---+---+-------+------+-----+----+----
-----------------------------------------------------------------------+------------------
+
|CRIM   |ZN   |INDUS|CHAS|NOX   |RM   |AGE |DIS    |RAD|TAX|PTRATIO|B     |LSTAT|MEDV|feat
ures                                                                   |prediction
|
+-------+-----+-----+----+------+-----+----+-------+---+---+-------+------+-----+----+----
-----------------------------------------------------------------------+------------------
+
|0.00906|90.0 |2.97 |0   |0.4   |7.088|20.8|7.3073 |1  |285|15.3   |394.72|7.85 |32.2|[0.0
0906,90.0,2.97,0.0,0.4,7.088,20.8,7.3073,1.0,285.0,15.3,394.72,7.85]   |33.1648629401689
|
|0.01096|55.0 |2.25 |0   |0.389 |6.453|31.9|7.3073 |1  |300|15.3   |394.72|8.23 |22.0|[0.0
1096,55.0,2.25,0.0,0.389,6.453,31.9,7.3073,1.0,300.0,15.3,394.72,8.23] |27.6353932691948
|
|0.01311|90.0 |1.22 |0   |0.403 |7.249|21.9|8.6966 |5  |226|17.9   |395.93|4.81 |35.4|[0.0
1311,90.0,1.22,0.0,0.403,7.249,21.9,8.6966,5.0,226.0,17.9,395.93,4.81] |32.33740973700566
5|
|0.01432|100.0|1.32 |0   |0.411 |6.816|40.5|8.3248 |5  |256|15.1   |392.9 |3.95 |31.6|[0.0
1432,100.0,1.32,0.0,0.411,6.816,40.5,8.3248,5.0,256.0,15.1,392.9,3.95] |32.45612386492453
5|
```

END OF LAB