Table of Contents

Lesson 5 – Applications of Machine Learning

Chapter 1: Lesson Overview

In this lesson, we will first look at **deep learning**. You will learn about:

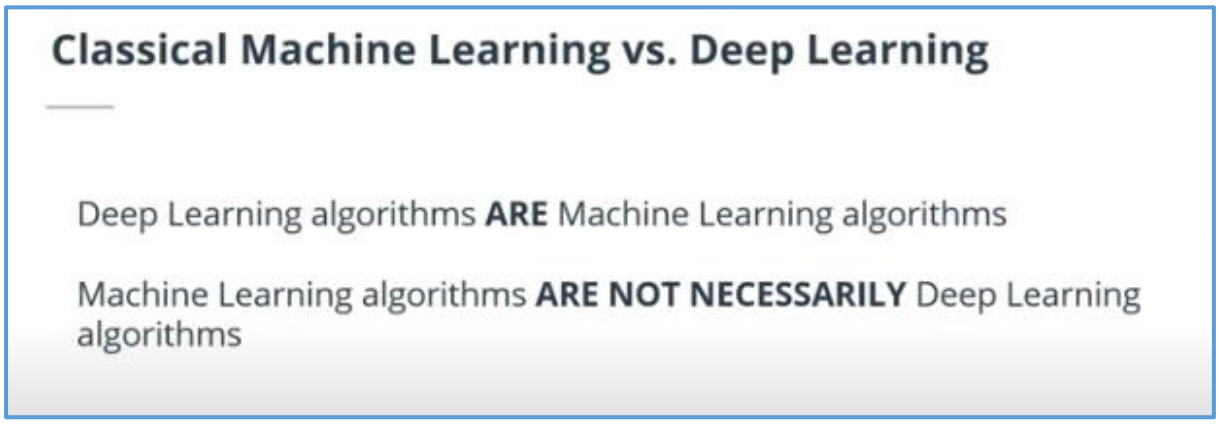
- The differences between classical machine learning and deep learning
- The benefits and applications of Deep Learning
- How to train your first neural network model

Next, you will learn about some of the most important **specialized cases of model training**, including:

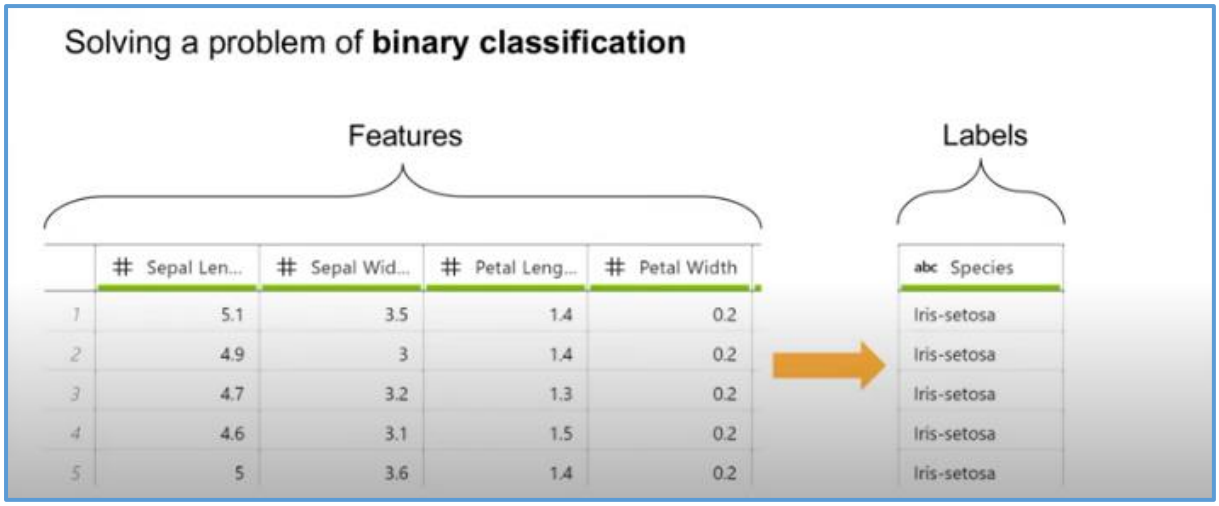
- *Similarity learning* and the basic features of a recommendation engine
- *Text classification* and the fundamentals of processing text in machine learning
- *Feature learning*, an essential task in feature engineering
- Anomaly detection
- Time-series forecasting.

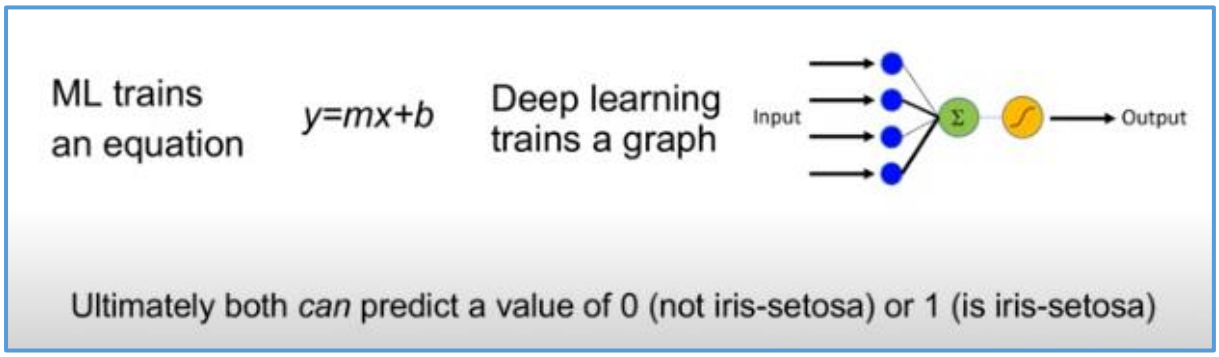
Along the way, you will get practice with several hands-on labs, in which you will train a simple neural network, train a recommendation engine, train a text classifier, and get some experience with forecasting

Chapter 2: Classical Machine Learning vs. Deep Learning

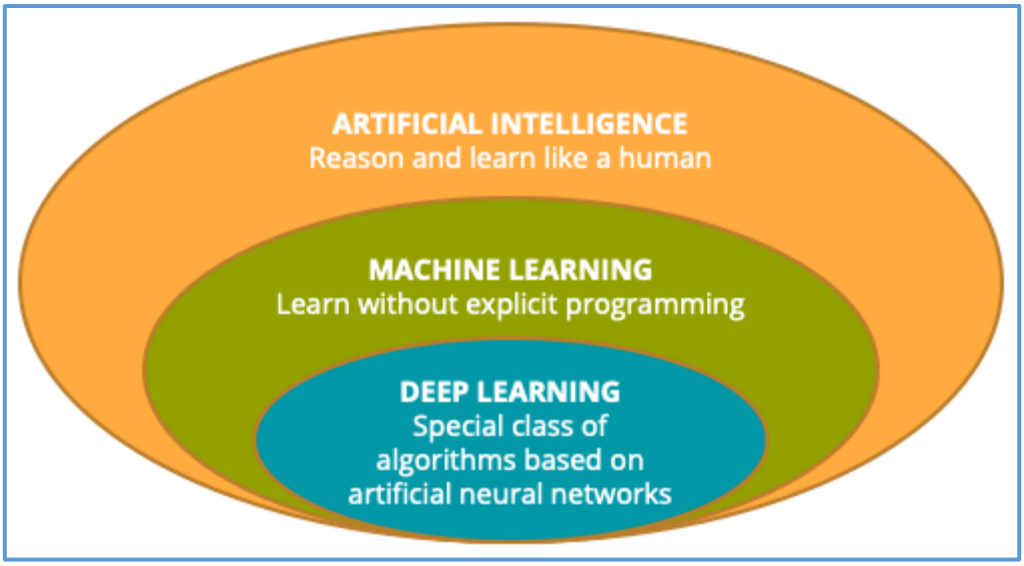


In ML most of the time there is more than a single solution to a problem.





As we just described, Artificial Intelligence (AI) *includes* Machine Learning (ML), which *includes* Deep Learning (DL). We can visualize the relationship like this:

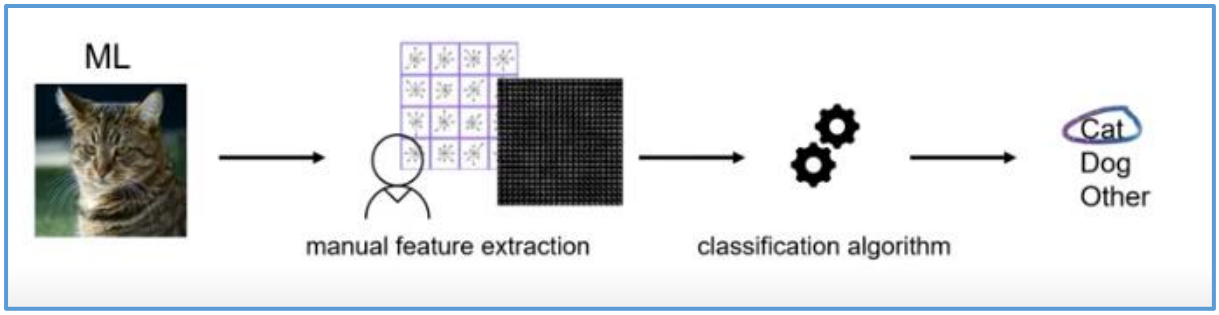


As the diagram shows, all deep learning algorithms are particular cases of machine learning algorithms—but it's *not* true that all machine learning algorithms are deep learning algorithms.

A More Detailed Comparison

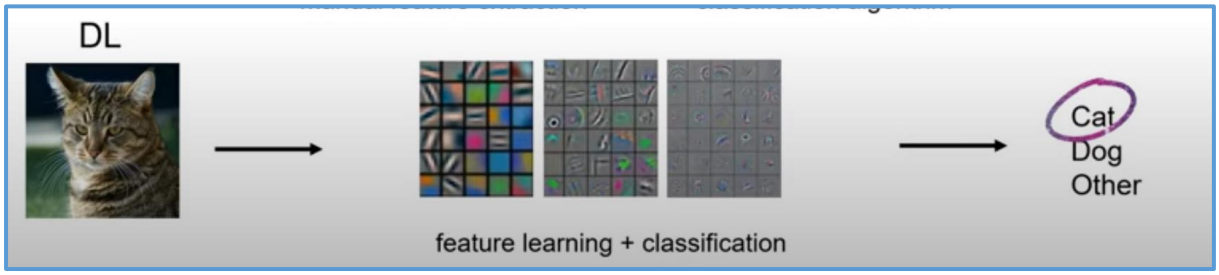
One of the most important fact that differentiates Deep learning from Classical ML is its inner capability of learning new features.

In case of Classical ML we have to extract features manually. Then use the selection of those features to train a Classification Algorithm, which then yields the Classification results.



However, in case of DL the internal layers of neural network have built-in capabilities that allow them to implicitly, learn features without you having to explicitly, encode those features.

Then with every layer, different and potentially more complex features are learned.



QUIZ QUESTION

Which of the following statements describes most accurately the relationship between classical Machine Learning and Deep Learning?

- Machine Learning algorithms are a special case of Deep Learning algorithms and have the capability of learning accurately complex, non-linear functions from data
- Deep Learning and Machine Learning algorithms are two completely different sets of algorithms
- ✓ Deep Learning algorithms are a special case of Machine Learning algorithms and have the capability of learning accurately complex, non-linear functions from data
- Machine Learning and Deep Learning algorithms are essentially equivalent, and can be easily translated into each other.

Chapter 3: What is Deep Learning?

ML is a subset of AI that focuses on creating programs that are capable of learning without explicit instructions.

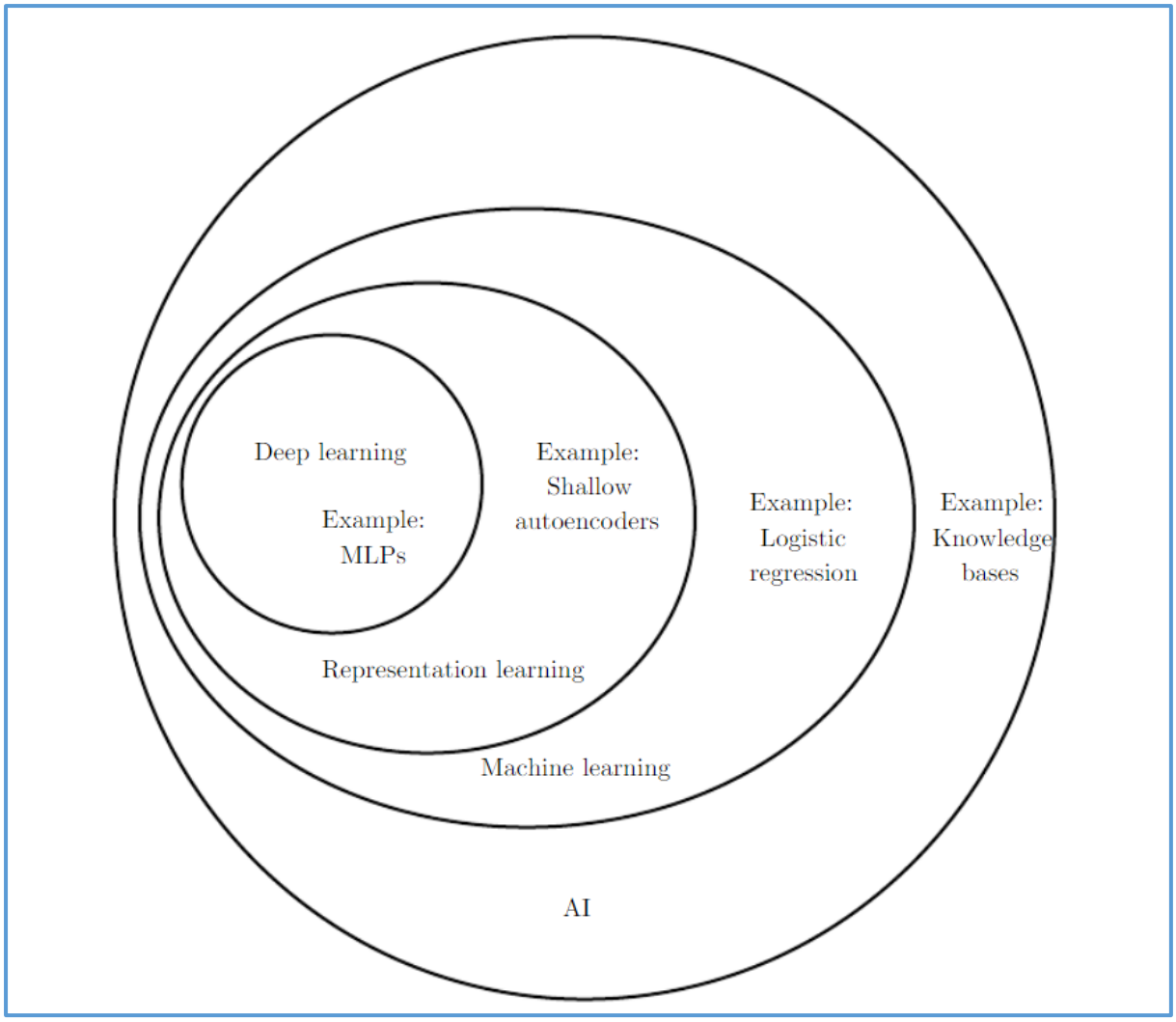
QUIZ QUESTION

Which field emerged to address the problem of *learning functions from data without explicit programming*?

- Artificial intelligence
- ✓ Machine learning
- Deep learning

Inspired from the organization of human brain, neural net were initially intensely, studied as part of ML. However, due to lack of H/W resources the area was, abandoned. Around Year-2000, fuelled by proper H/W [GPU]. The field of neural net came back. Complex Neural network with large number of hidden layers, with large number of nodes were in the grasp of data scientists. Hence the name Deep Learning.

DL is a special class of ML algorithms that are, based on artificial neural networks & ML is a subset of the wider discipline, which is, called Artificial Intelligence.



A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is, used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

Example of AI is problem of Knowledge bases. Initial use of the term was to describe one of the two parts of a knowledge base system. The other part being the inference engine that reasons about the facts that are, represented in the knowledge base. Thus, problem of implementing a Knowledge base is outside the realm of ML, rather it is in the pure realm of AI.

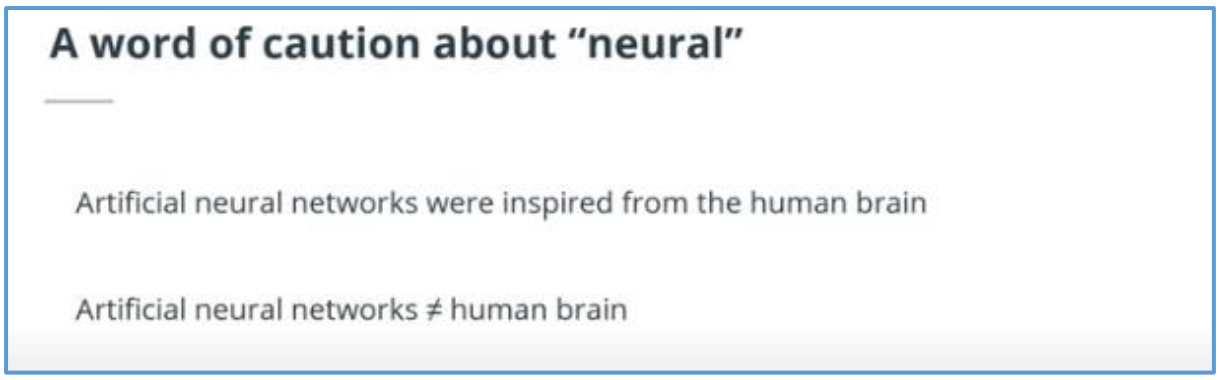
Example of classical algorithm used in ML would be Logistic regression.

Within Neural networks, there is also a separate distinction and DL is actually a subset of the wider set of neural net algorithms.

Shallow auto-encoder is a type of algorithm, which is a neural network, yet it is not a deep neural network.

MLP – Multi layer perceptron is, considered as deep learning algorithm.

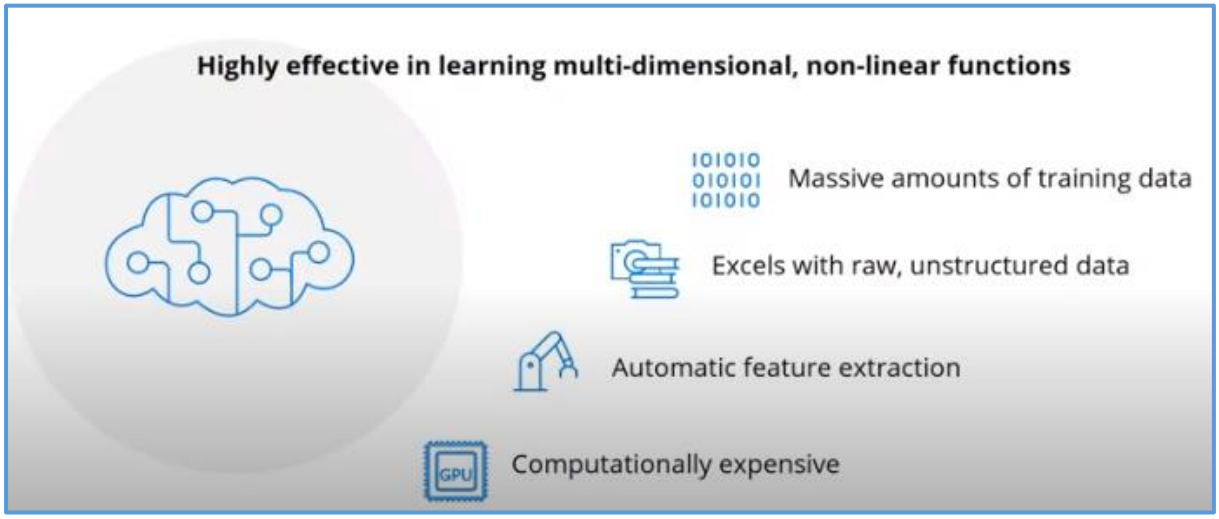
Within the context of neural networks, we have regular neural networks and deep neural networks. Deep neural networks are the ones, which are actually, tied to the field of ML.



Average number of neurons in the human brain ranges between 86 to 88 billion neurons & they have ~1000 Trillion synaptic connections between them. Finally using electrochemical signals information is, transmitted through the synaptic connections.

Chapter 4: Characteristics of Deep Learning

- Most important characteristic of DL is its effectiveness in learning multidimensional non-linear functions.
- Training a model is nothing more than learning a function & the form of the function decides the type of ML algorithm that one can use.
- DL is capable of handling massive amount of training data.
- DL excels with Raw and unstructured data. Since it has the capability of learning new features at every single hidden layer of the neural network.
- DL has the capability of automatic feature extraction.
- Training of DL models is computationally expensive.



QUIZ QUESTION

Which of the below statements are true about deep learning?

(Select all that apply.)

- [x] It can be very computationally expensive, especially as model complexity increases
- [ ] It requires well-processed, highly structured data
- [x] It works well with massive amounts of training data
- [x] It is capable of extracting features automatically

Chapter 5: Prelaunch Lab

Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 6: Benefits & Applications of Deep Learning

The non-parametric approach taken by neural nets allows them to learn arbitrarily complex functions

Capability to learn arbitrarily complex functions. They take a non-parametric approach, which enable them to learn complex functions without actually putting a cap on the complexity of the learned functions.

Majority of ML models, which take parametric approach are limited to the form of the function they can learn.

Hence, they can learn complex patterns without explicitly seeing them

Effective across various "incarnations" of data (numbers, images, text)

Work on (very) large sets of data

Lately, can learn time-related patterns (RNN = Recurrent Neural Network)

Speech, when it is translated to data, it is highly time-related type of data, because speech occurs as a continuous stream of sound, of frequencies that are time based.

Capable of reaching on-par performance with certain human activities

QUIZ QUESTION

Which one of the following statements about deep learning algorithms is *not* true?

- They can be used on very large sets of data.
- They can learn complex patterns without explicitly seeing those patterns.
- They can be distributed for parallel training.
- A key benefit of deep learning algorithms is the ability to learn arbitrarily complex functions.
- ✓ They usually use less computational resources as compared to other approaches.

Applications of Deep Learning

- **Language Translation –** DL models have been, proven to be, equal to humans. Task is taking one input text in one language and translating that into an output text of another language.
- **Image Recognition –** Apply DL to images and DL is capable of automatically extracting increasingly complex features out of those images.
- **Speech Recognition –** Speech when saved into numerical format it is, translated into time series based information.
- **Forecasting – Other types of time series based ML tasks**
- **Predictive Analytics**
- **Autonomous vehicles – Combining Image recognition, Speech recognition & reinforcement learning**

Chapter 7: Lab - Train a simple neural net

Train a simple neural net model

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

Neural network regression is a supervised learning method, and therefore requires a tagged dataset, which includes a label column. Because a regression model predicts a numerical value, the label column must be a numerical data type.

Lab Overview

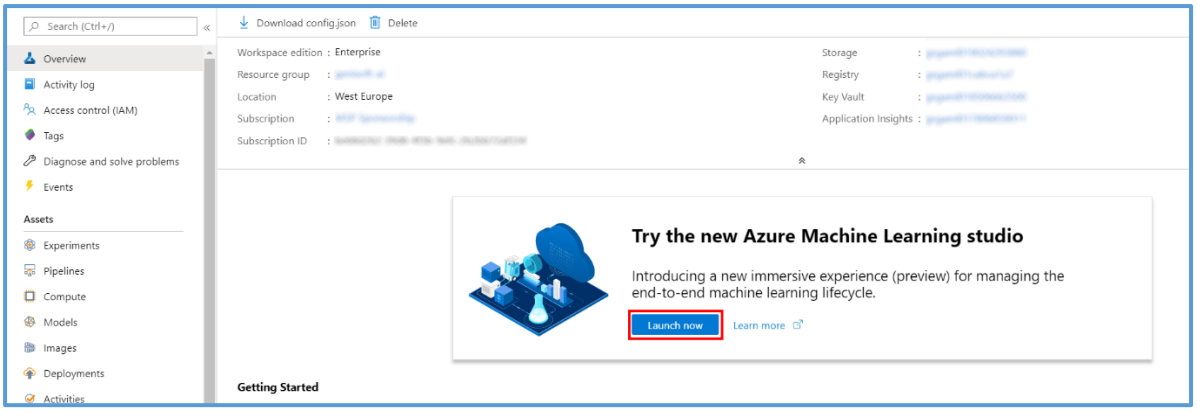
In this lab we will be using a subset of NYC Taxi & Limousine Commission - green taxi trip records available from Azure Open Datasets. The data is enriched with holiday and weather data. Based on the enriched dataset, we will configure the prebuilt Neural Network Regression module to create a regression model using a customizable neural network algorithm. We will train the model by providing the model and the NYC taxi dataset as an input to Train Model. The trained model can then be used to predict NYC taxi fares. We will do all of this from the Azure Machine Learning designer without writing a single line of code.

Exercise 1: Register Dataset with Azure Machine Learning studio

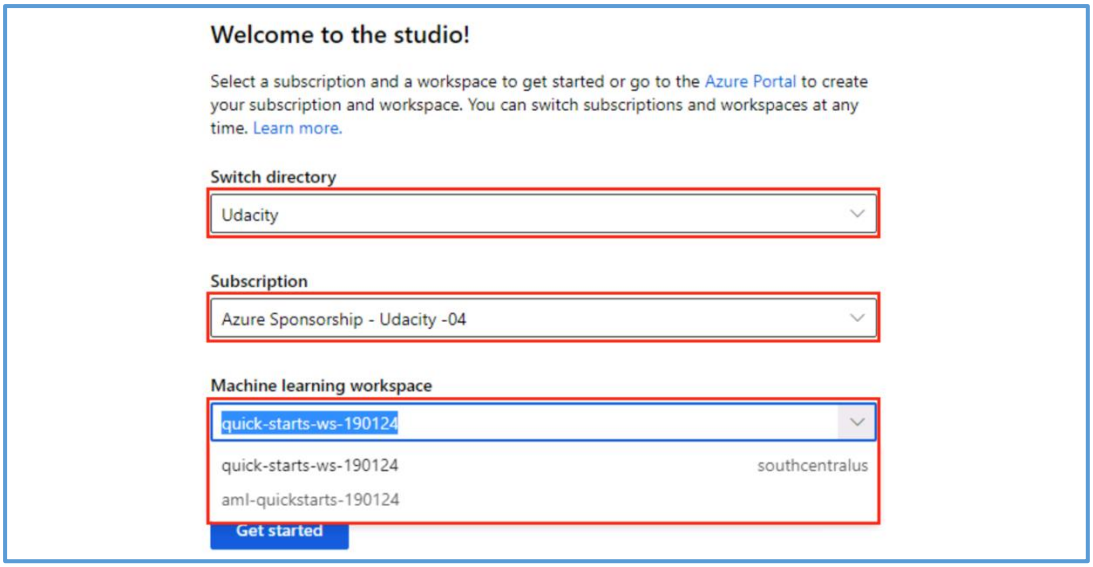
Task 1: Upload Dataset

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

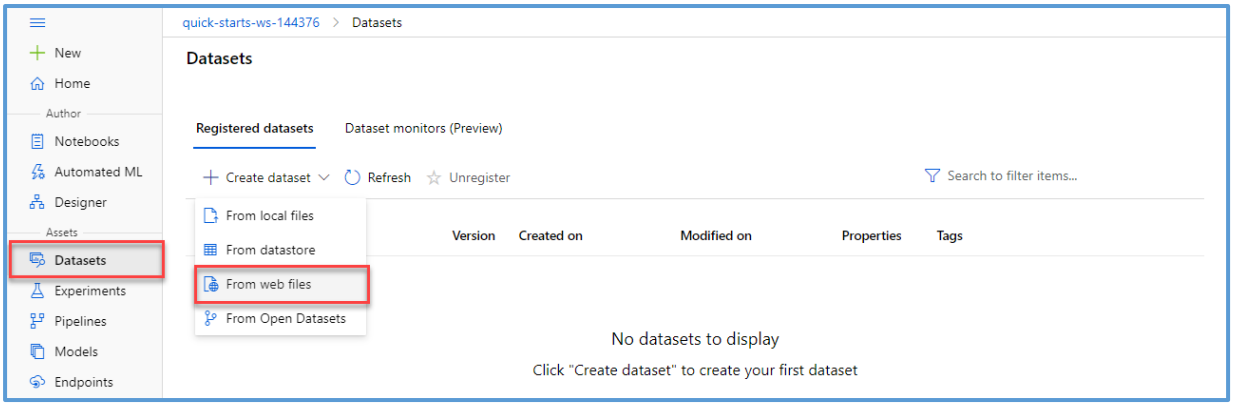


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:



For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it does not matter which) and then click **Get started**.

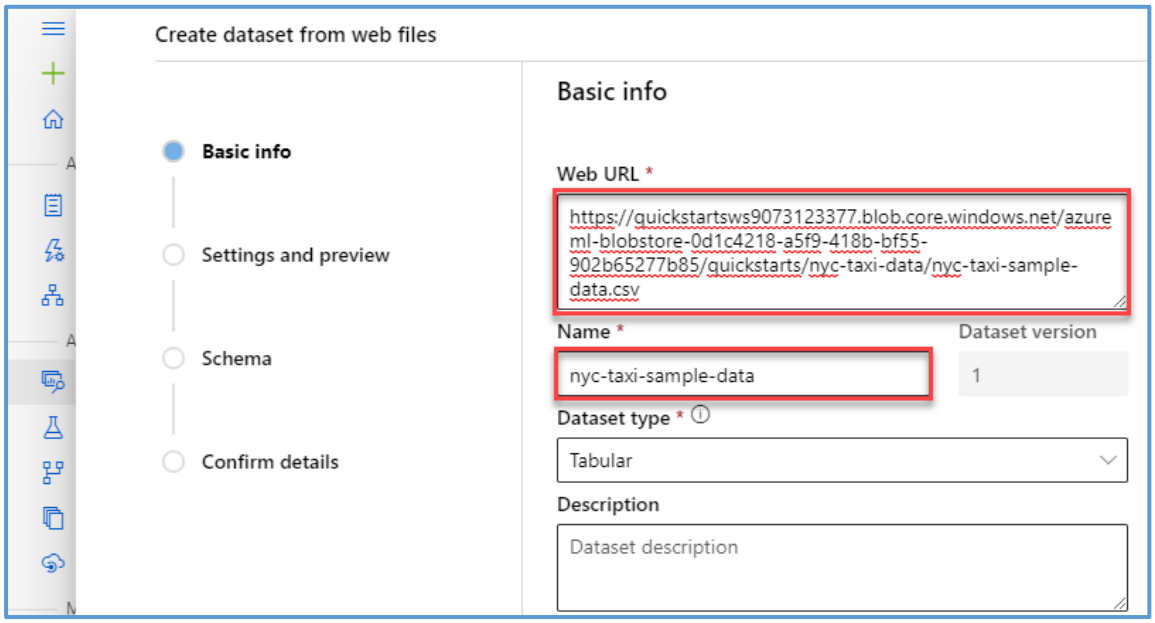
4. From the studio, select **Datasets, + Create dataset, from web files**. This will open the Create dataset from web files dialog on the right.



5. In the Web URL field provide the following URL for the training data file:

   https://introtomlsampledata.blob.core.windows.net/data/nyc-taxi/nyc-taxi-sample-data.csv

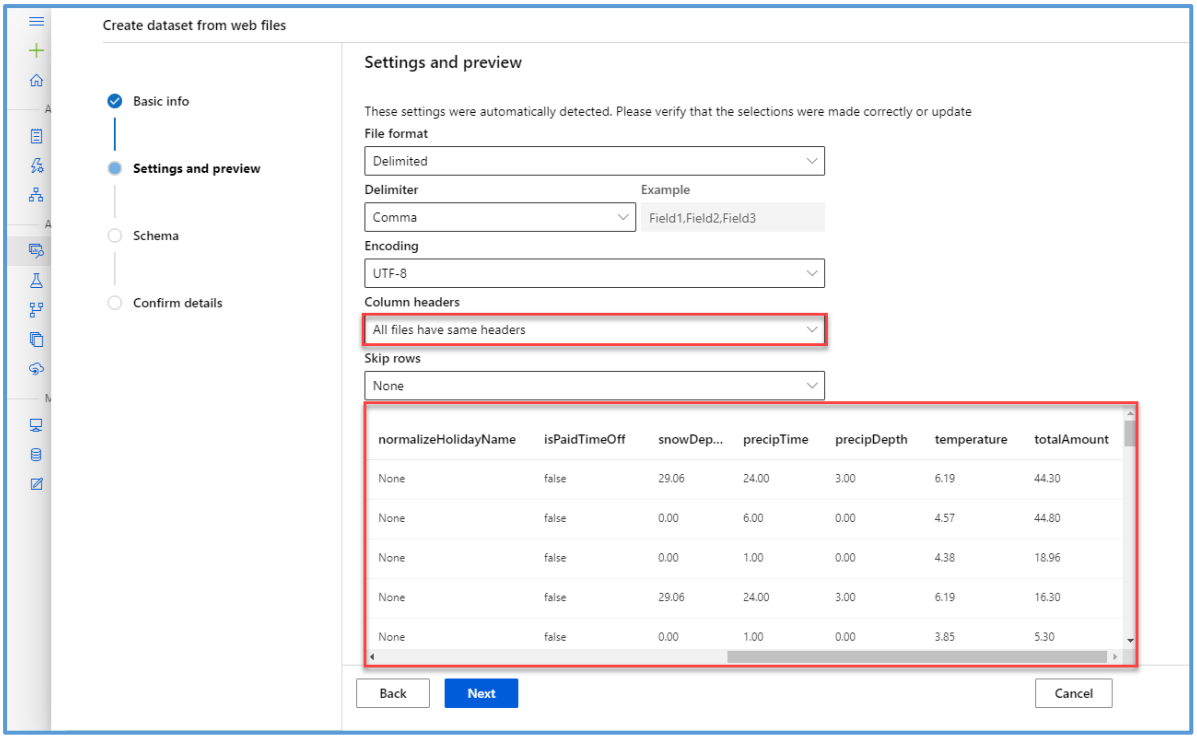
6. Provide `nyc-taxi-sample-data` as the Name, leave the remaining values at their defaults and select **Next**.



Task 2: Preview Dataset

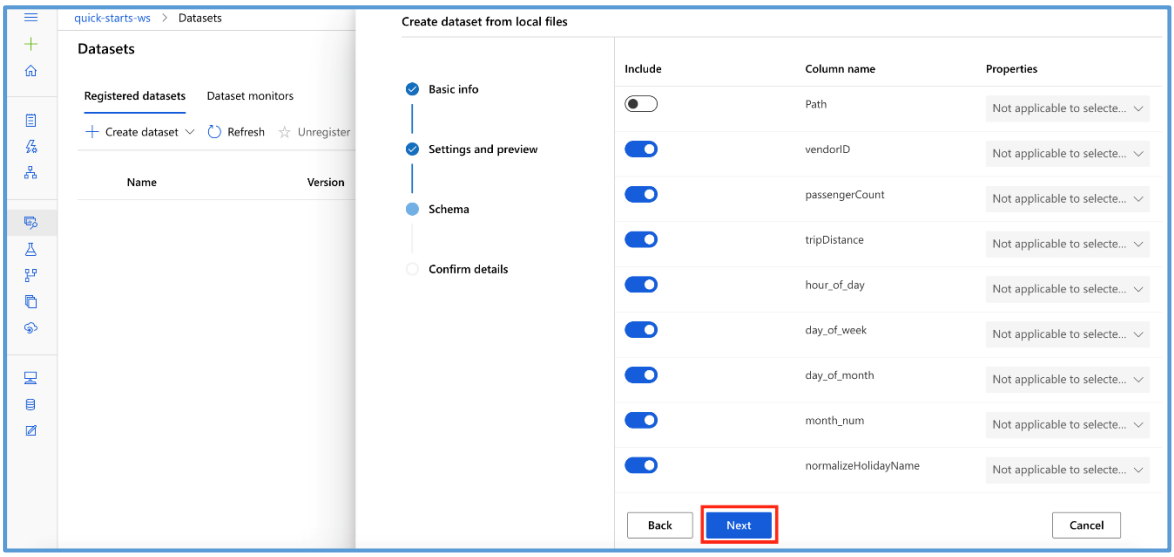
1. On the Settings and preview panel, set the column headers drop down to `All files have same headers`.

2. Scroll the data preview to right to observe the target column: `totalAmount`. After you are done reviewing the data, select **Next**



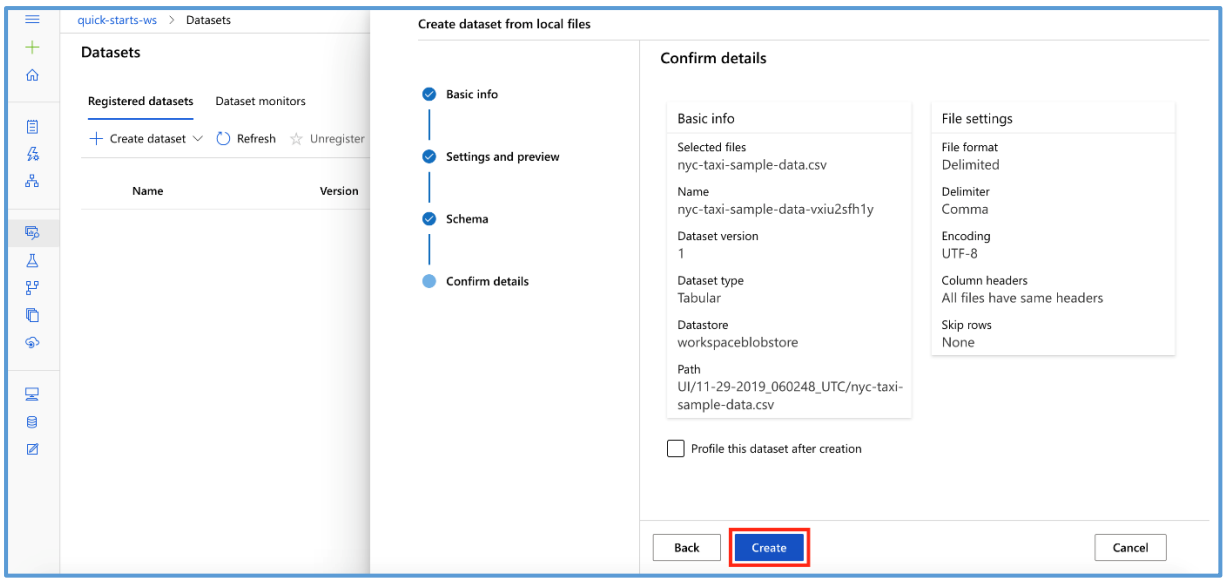
Task 3: Select Columns

1. Select columns from the dataset to include as part of your training data. Leave the default selections and select **Next**



Task 4: Create Dataset

1. Confirm the dataset details and select **Create**



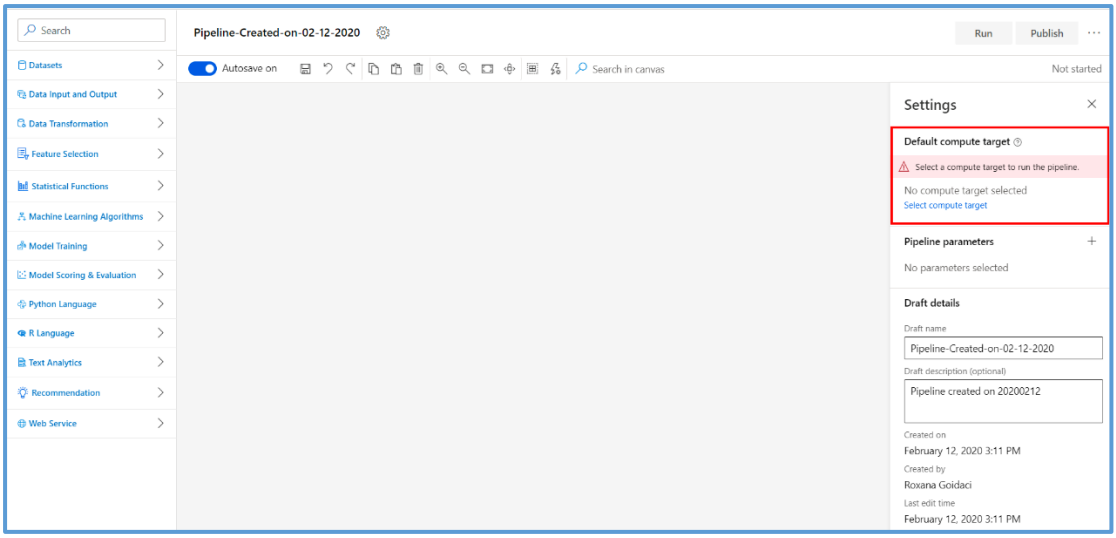
Exercise 2: Create New Training Pipeline

Task 1: Open Pipeline Authoring Editor

1. From the studio, select **Designer, +**. This will open a visual pipeline authoring editor.

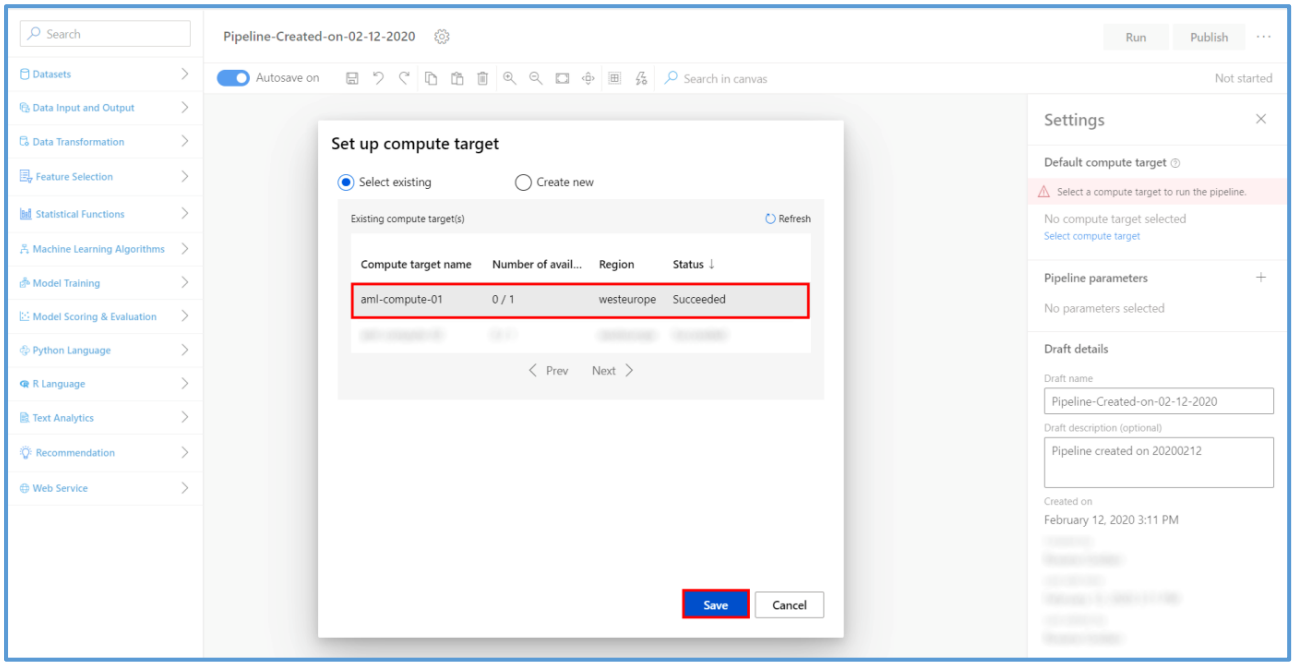
Task 2: Setup Compute Target

1. In the settings panel on the right, select **Select compute target**.



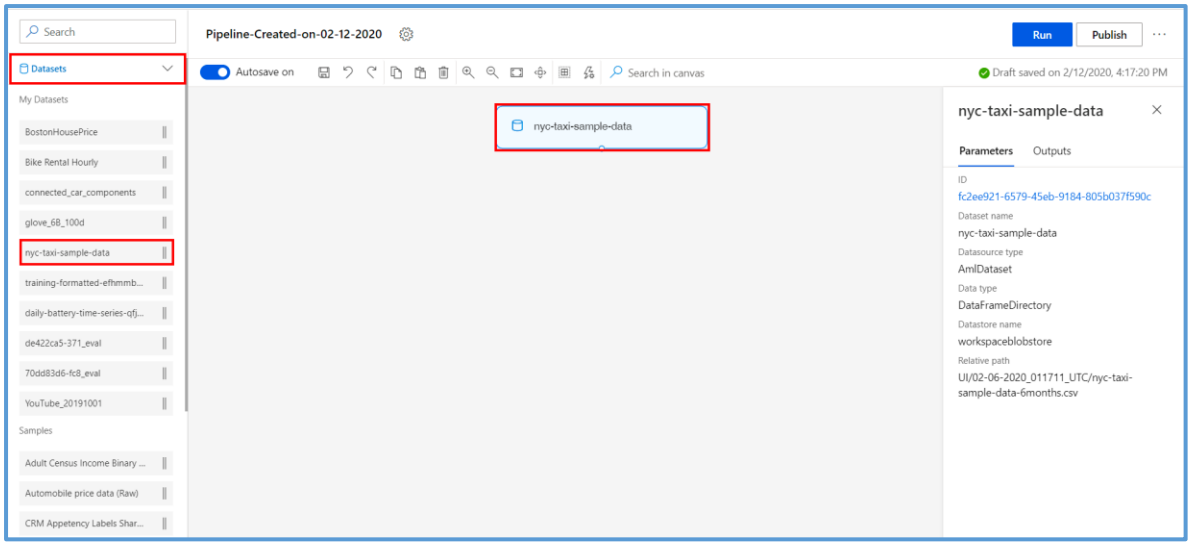
2. In the Set up compute target editor, select the available compute, and then select **Save**.

Note: If you are facing difficulties in accessing pop-up windows or buttons in the user interface, please refer to the Help section in the lab environment.



Task 3: Add Dataset

1. Select **Datasets** section in the left navigation. Next, select **My Datasets, nyc-taxi-sample-data** and drag and drop the selected dataset on to the canvas.



Task 4: Split Dataset

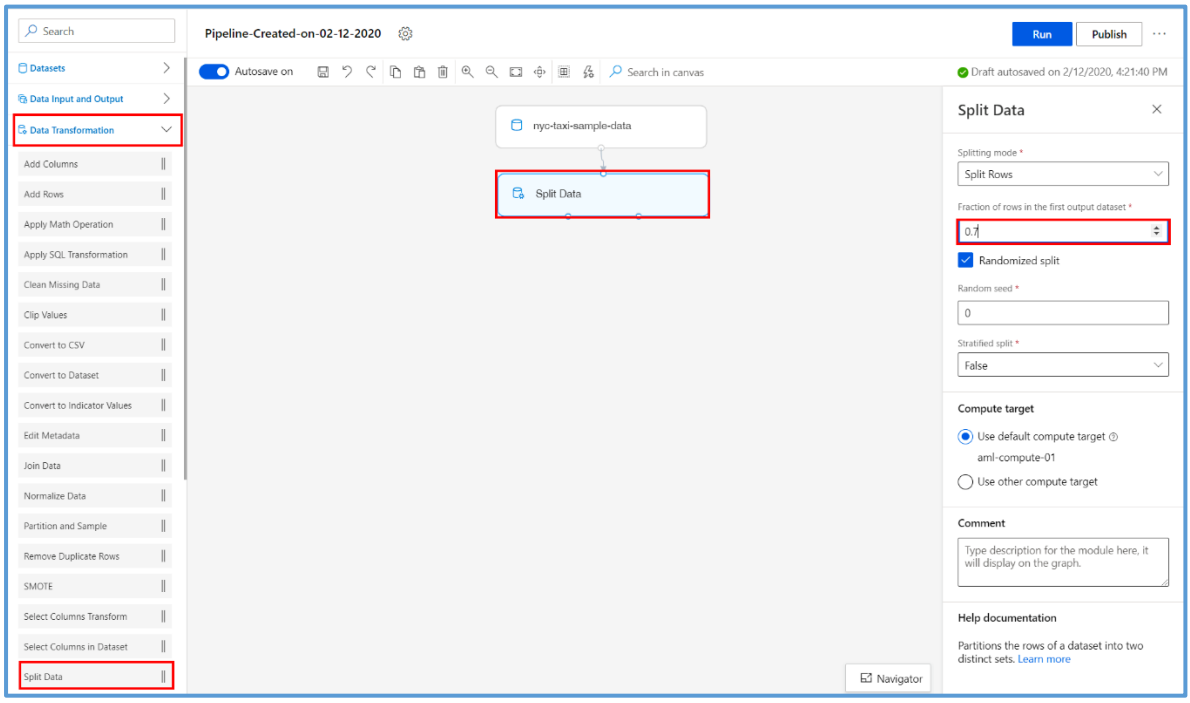
1. Select **Data Transformation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Split Data** prebuilt module

    2. Drag and drop the selected module on to the canvas

    3. Fraction of rows in the first output dataset: **0.7**

    4. Connect the Dataset to the Split Data module



Note that you can submit the pipeline at any point to peek at the outputs and activities. Running pipeline also generates metadata that is available for downstream activities such selecting column names from a list in selection dialogs.

Task 5: Initialize Regression Model

1. Select **Machine Learning Algorithms** section in the left navigation. Follow the steps outlined below:

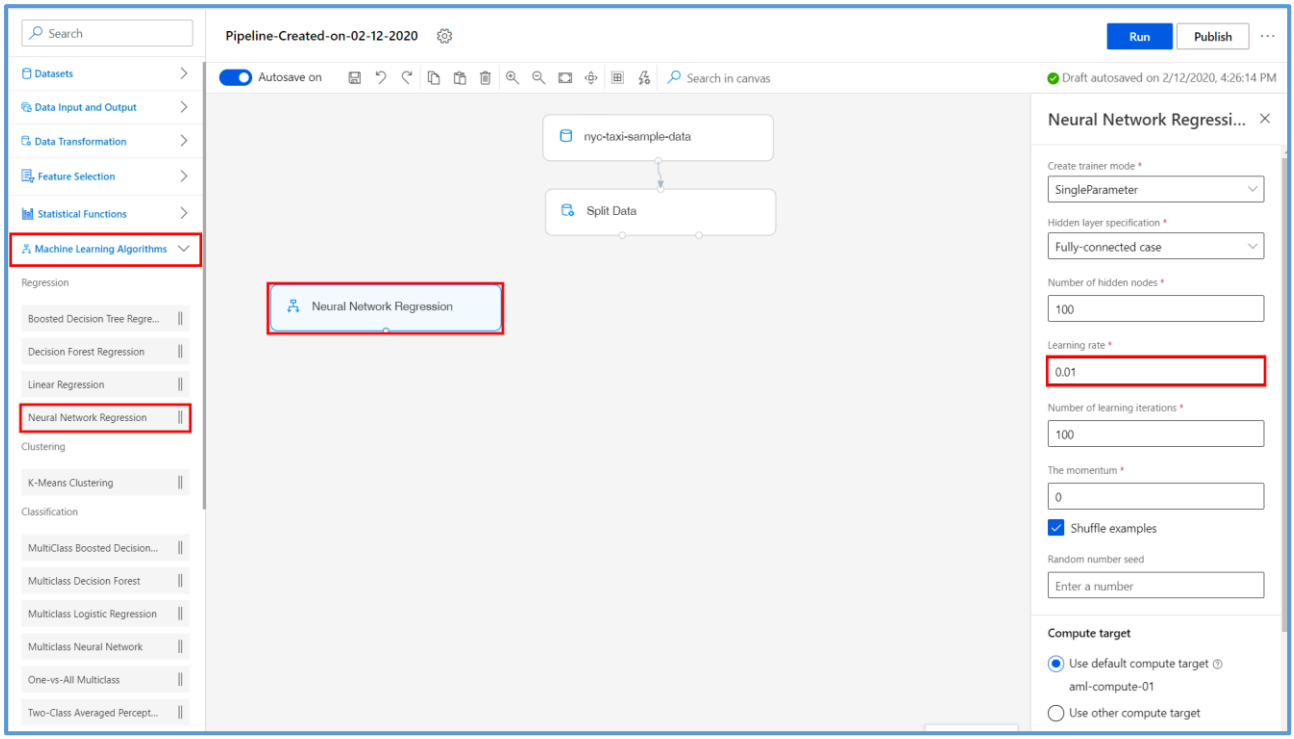
    1. Select the **Neural Network Regression** prebuilt module, in the Regression category.

    2. Drag and drop the selected module on to the canvas

    3. Create trainer mode **Single Parameter**. This option indicates how you want the model to be, trained.

    4. Hidden layer specification: **Fully connected case**.

    5. For Learning rate: **0.01**.

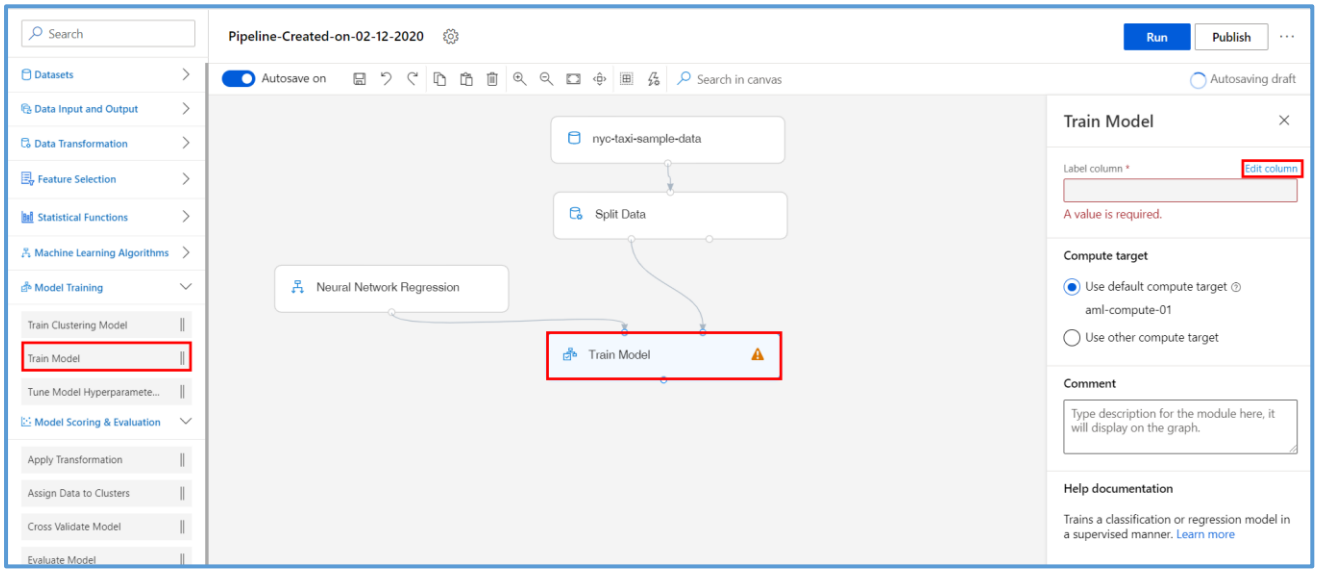


2. *Note: Because the number of nodes, in the input layer, is determined, by the number of features, in the training data, in a regression model there can be only one node in the output layer.*

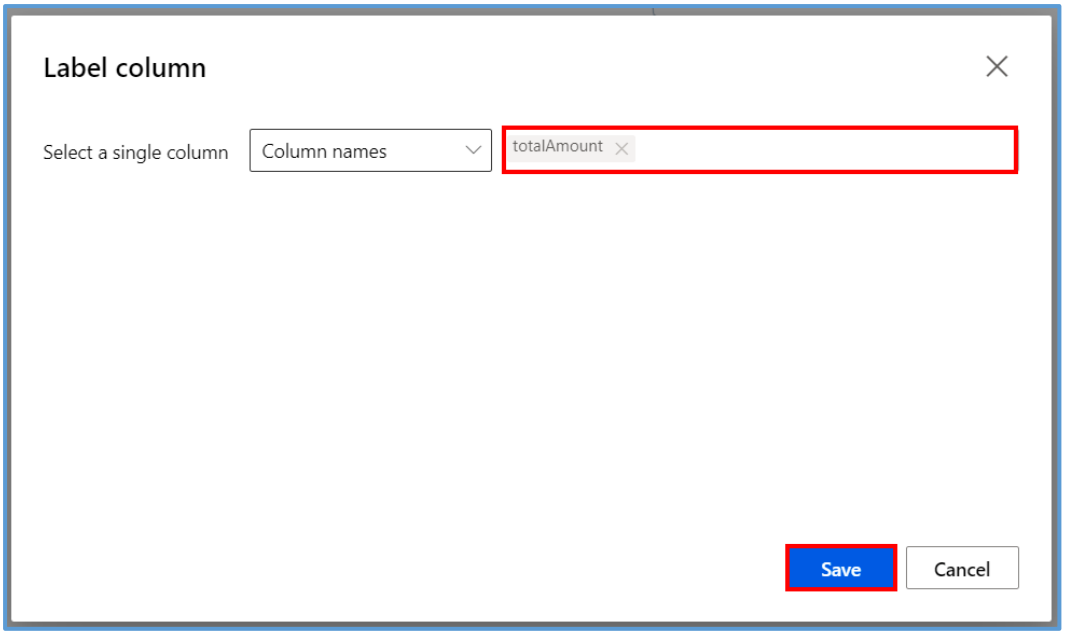
Task 6: Setup Train Model Module

1. Select **Model Training** section in the left navigation. Follow the steps outlined below:

1. Select the **Train Model** prebuilt module
2. Drag and drop the selected module on to the canvas
3. Connect the Neural Network Regression module to the first input of the Train Model module
4. Connect the first output of the Split Data module to the second input of the Train Model module
5. Select the **Edit column** link to open the Label column editor



2. The Label column editor allows you to specify your Label or Target column. Type in the label column name **totalAmount** and then select **Save**.



Task 7: Setup Score Model Module

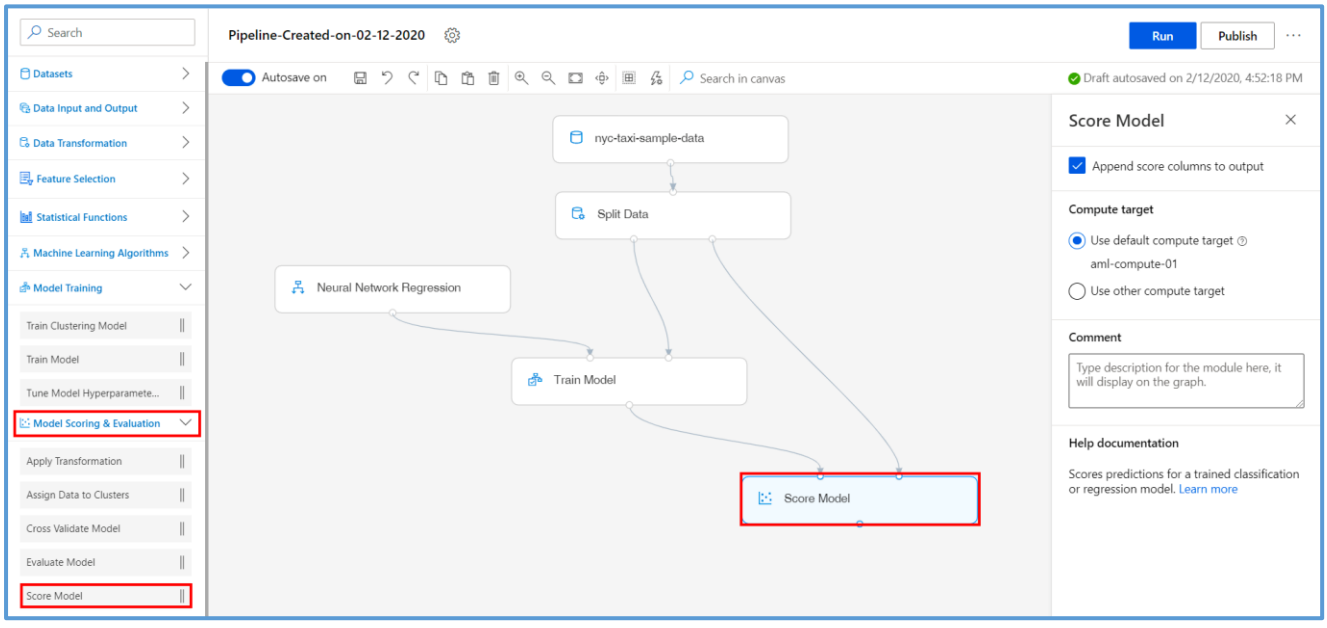
1. Select **Model Scoring & Evaluation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Score Model** prebuilt module

    2. Drag and drop the selected module on to the canvas

    3. Connect the Train Model module to the first input of the Score Model module

    4. Connect the second output of the Split Data module to the second input of the Score Model module



Note that Split Data module will feed data for both model training and model scoring. The first output (0.7 fraction) will connect with the Train Model module and the second output (0.3 fraction) will connect with the Score Model module.

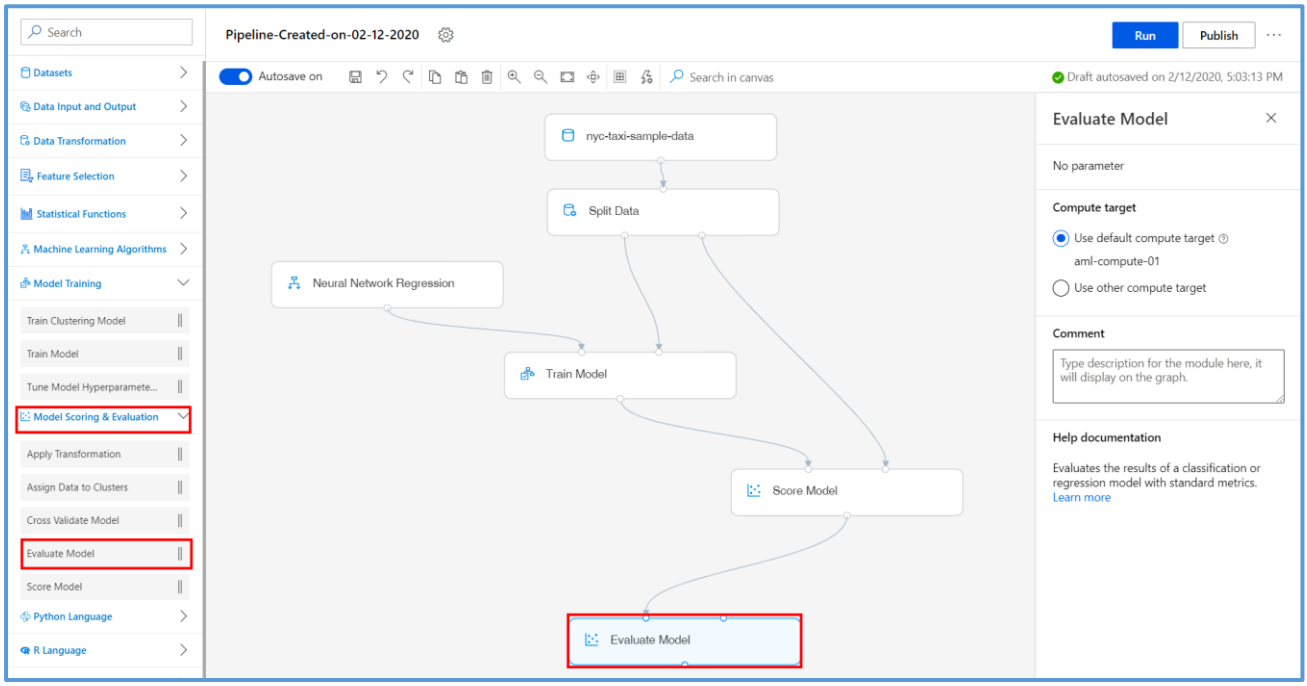
Task 8: Setup Evaluate Model Module

1. Select **Model Scoring & Evaluation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Evaluate Model** prebuilt module

    2. Drag and drop the selected module on to the canvas

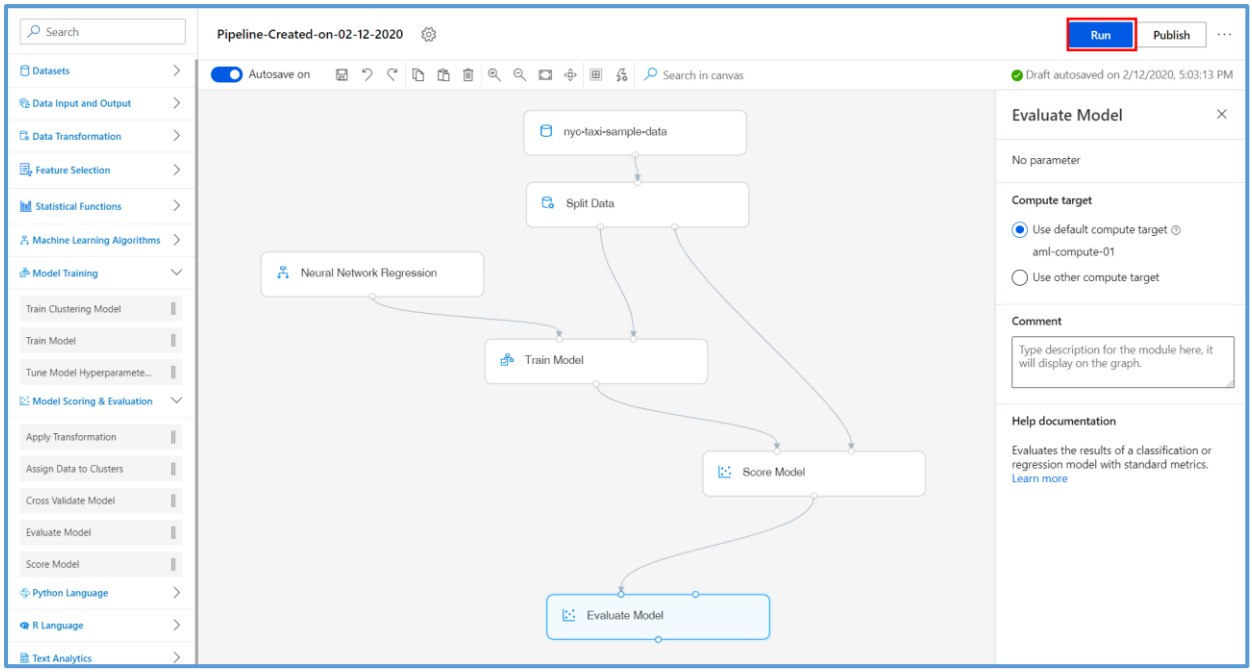
    3. Connect the Score Model module to the first input of the Evaluate Model module



Exercise 3: Submit Training Pipeline

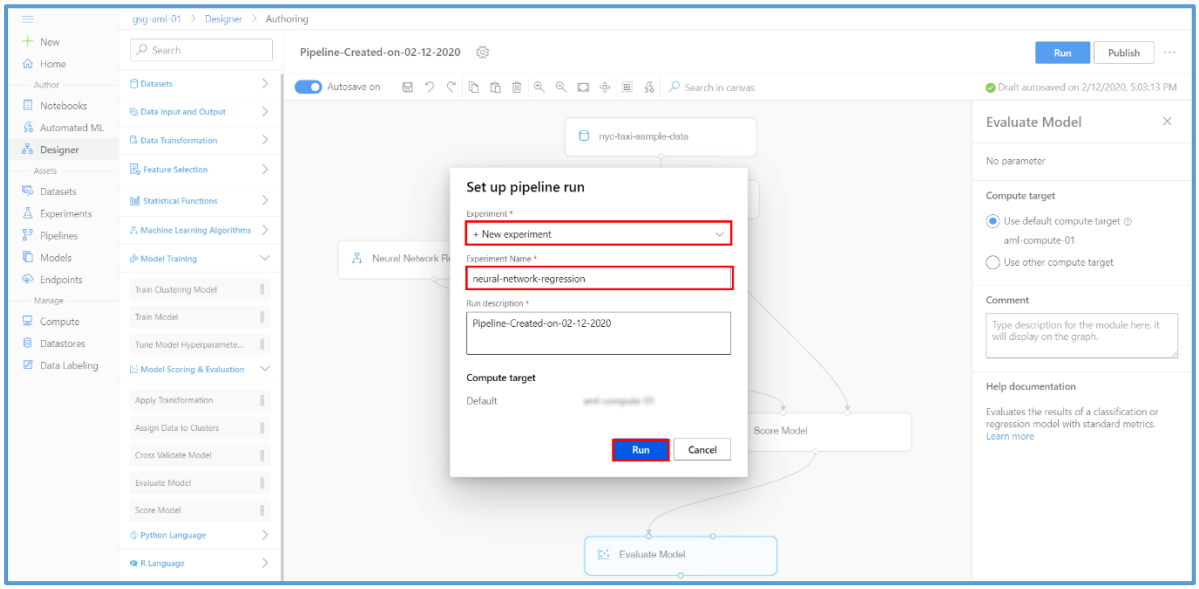
Task 1: Create Experiment and Submit Pipeline

1. Select **Submit** to open the Setup pipeline run editor.



Please note that the button name in the UI is, changed from **Run** to **Submit**.

2. In the Setup pipeline run editor, select **Experiment, Create new** and provide New experiment name: **neural-network-regression**, and then select **Submit**.



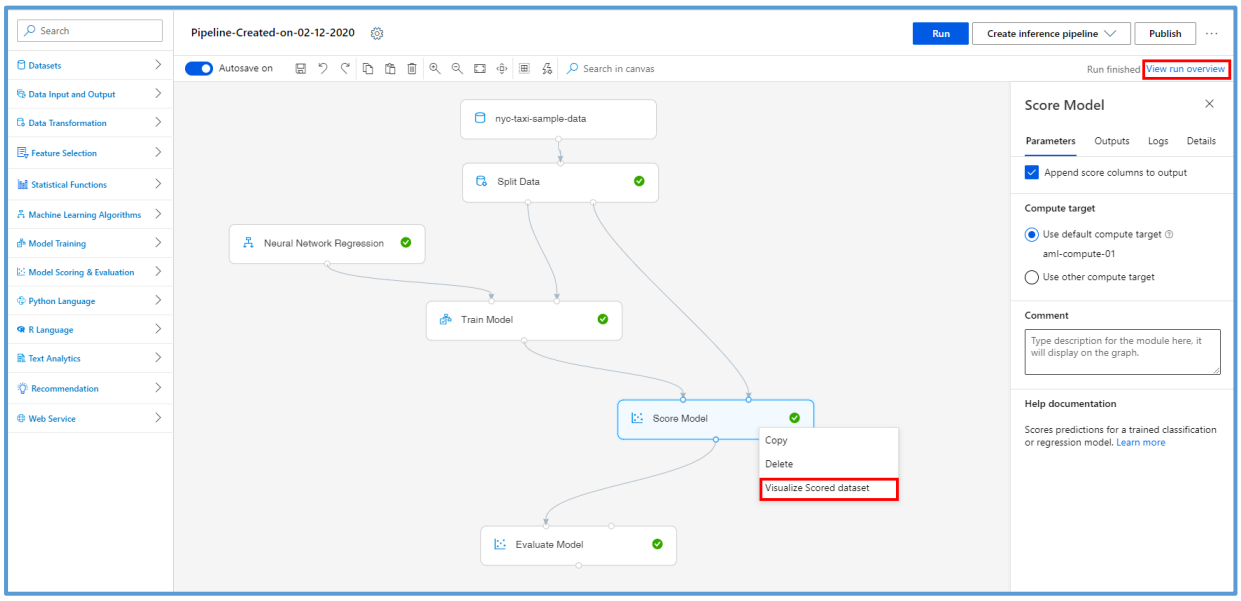
3. Wait for pipeline run to complete. It will take around **8 minutes** to complete the run.

4. While you wait for the model training to complete, you can learn more about the training algorithm used in this lab by selecting Neural Network Regression module.

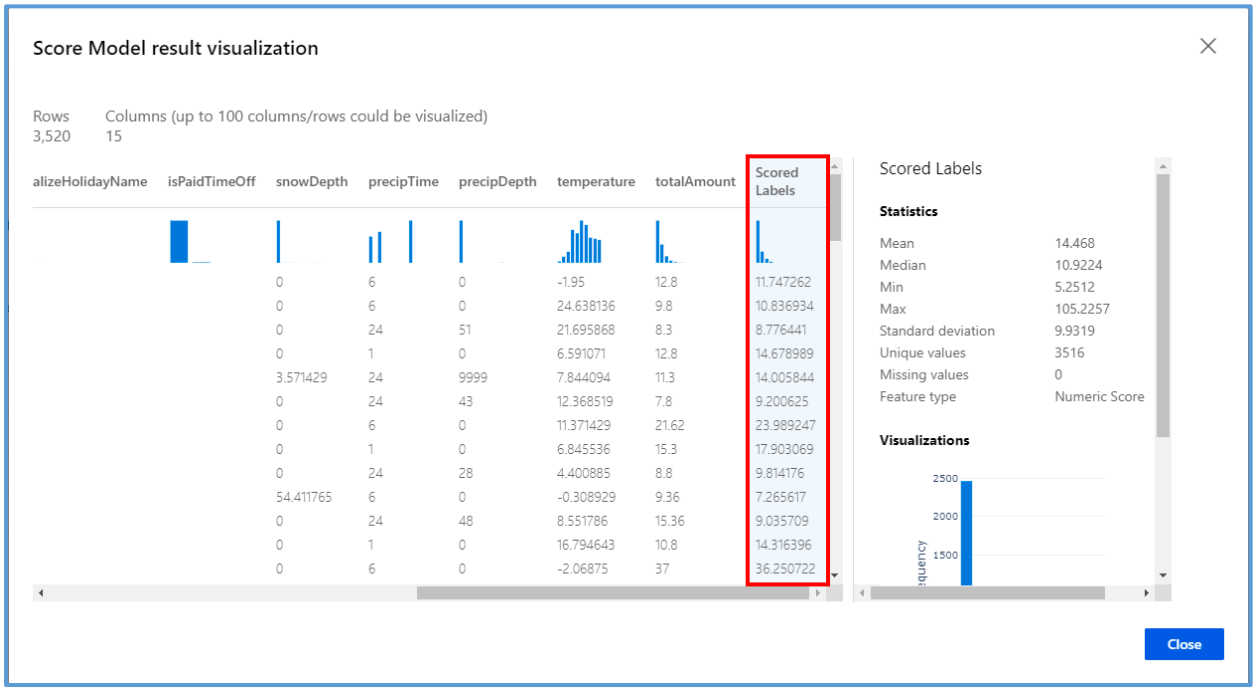
Exercise 4: Visualize Training Results

Task 1: Visualize the Model Predictions

1. Select **Score Model, Outputs, Visualize** to open the Score Model result visualization dialog or just simply right-click the Score Model module and select **Visualize Scored Dataset**.

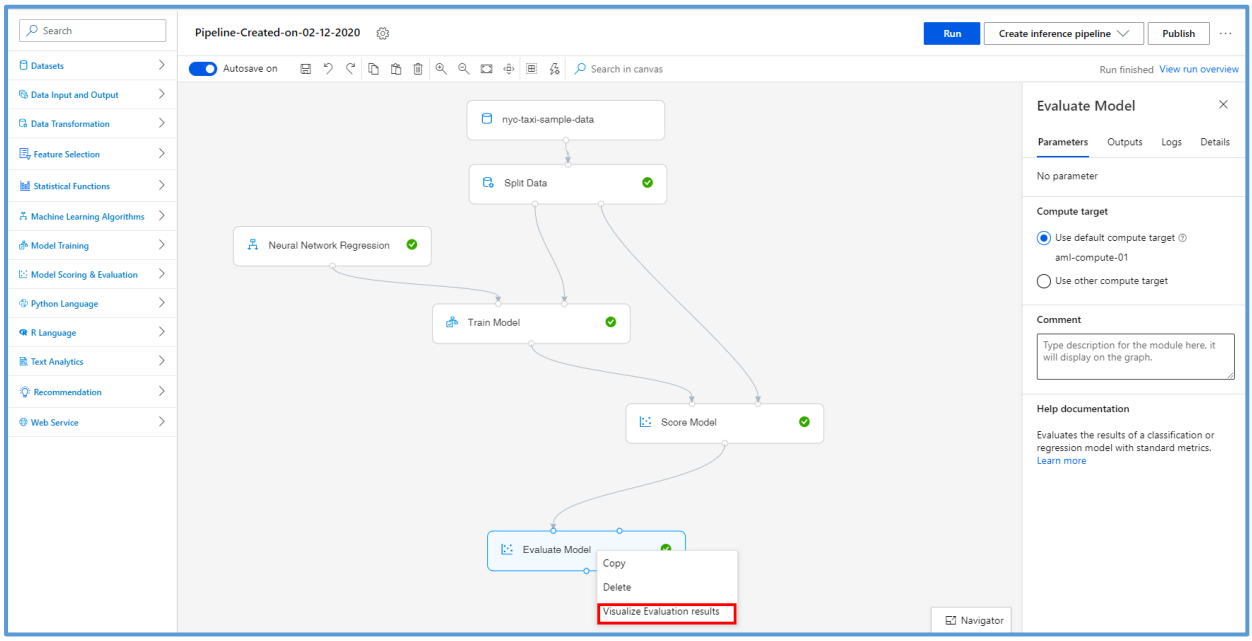


2. Observe the predicted values under the column **Scored Labels**. You can compare the predicted values (Scored Labels) with actual values (totalAmount).

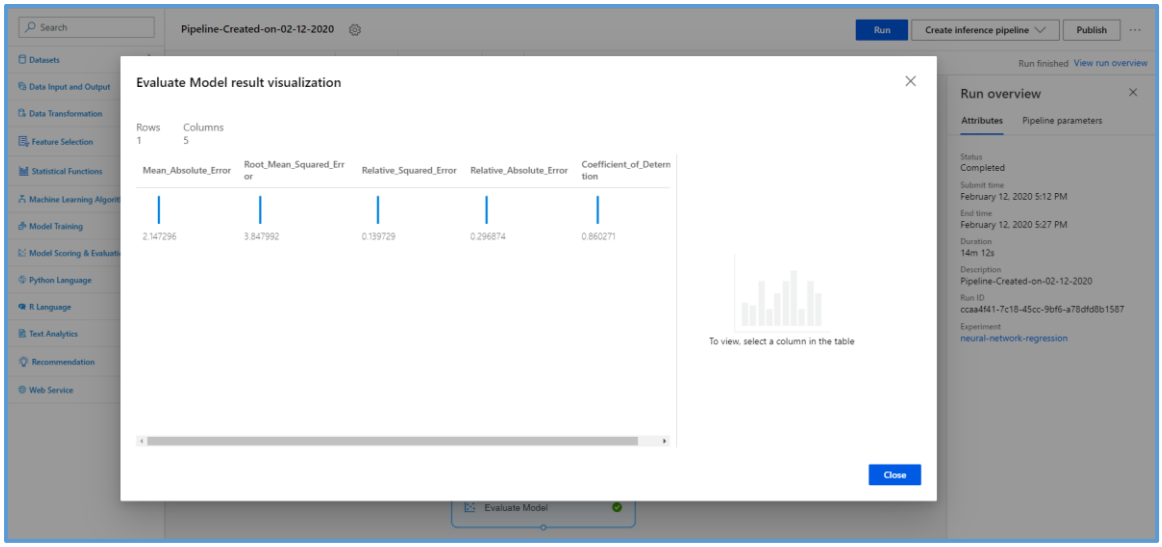


Task 2: Visualize the Evaluation Results

1. Select **Evaluate Model, Outputs, Visualize** to open the Evaluate Model result visualization dialog or just simply right-click the Evaluate Model module and select **Visualize Evaluation Results**.



2. Evaluate the model performance by reviewing the various evaluation metrics, such as **Mean Absolute Error**, **Root Mean Squared Error**, etc.



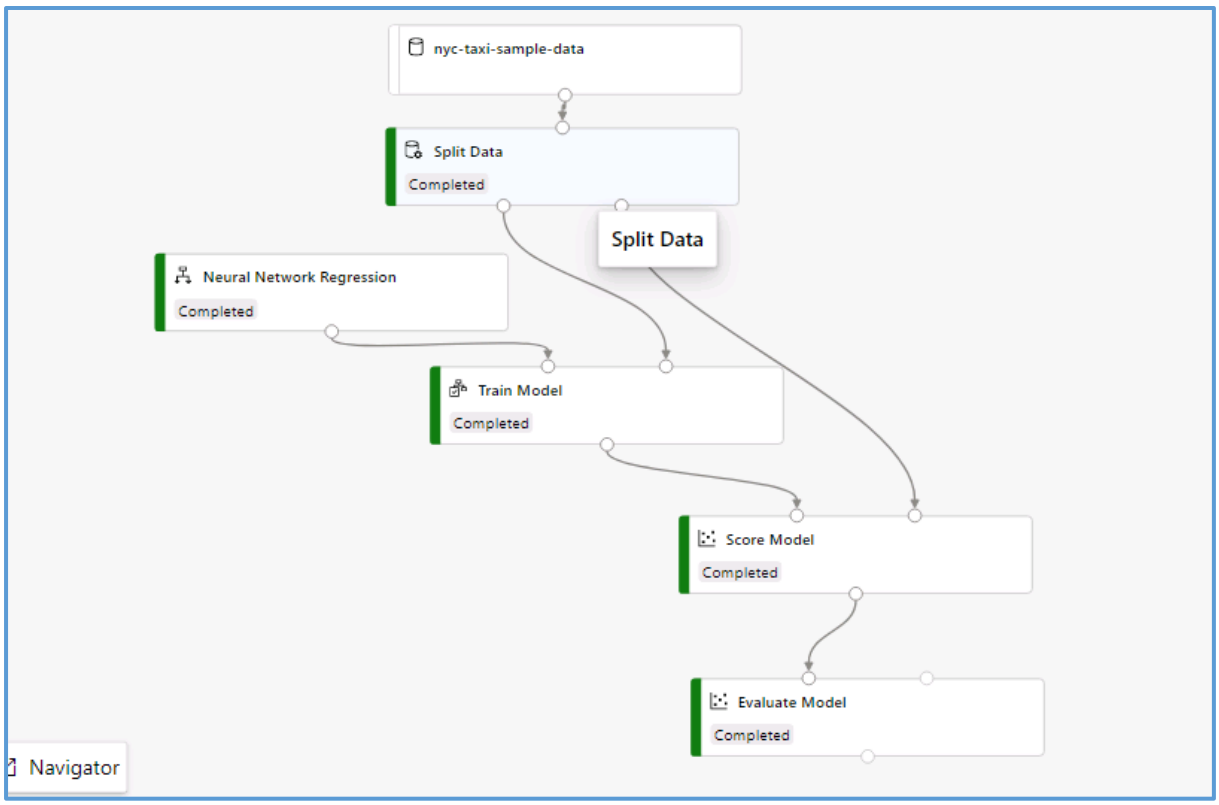
Next Steps

Congratulations! You have trained a simple neural net model using the prebuilt Neural Network Regression module in the AML visual designer. You can continue to experiment in the

environment but are free to close the lab environment tab and return to the Udacity portal to continue with the lesson.

Chapter 8: Walkthrough - Train a simple neural net

- Create the pipeline in the designer and set compute target as default.
- Registered the dataset
- Load the data and then Split the data. Splitting mode is Split Rows and using fraction of 0.7 for the split.[70% - Training data and 30% - Testing data]
- Select neural network regression as the algorithm with defined parameters.
- Finally Train >> Score >> Evaluate the model.
- In the Score model task, check the predictions (Scored Labels) made by the trained model in comparison to the output (totalAmount)



Chapter 9: Specialized Cases of Model Training

In this section, we will have a look at some of the specialized cases of model training. However, first, let us review the main types of machine learning approaches that we introduced back in the first lesson.

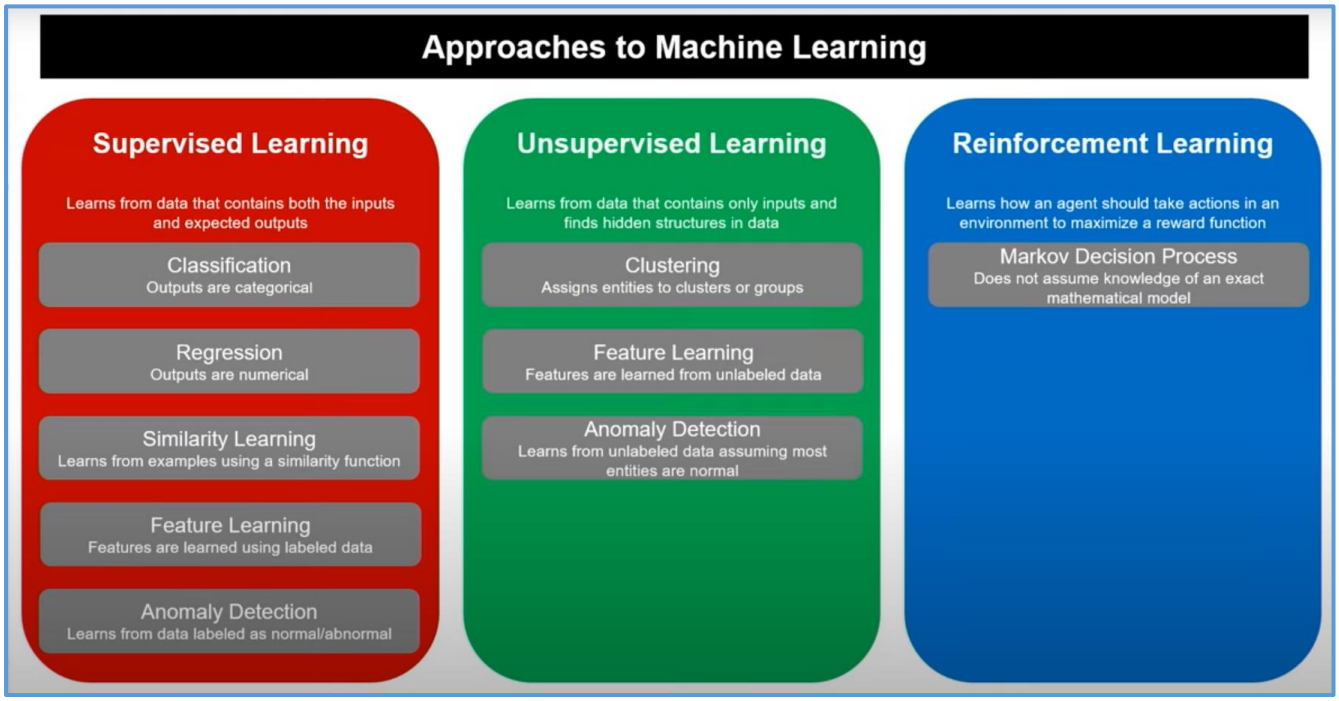
A Review of Approaches to Machine Learning

As you now know, there are three main approaches to machine learning:

- **Supervised learning**
- **Unsupervised learning**
- **Reinforcement learning**
- **Semi-supervised = Supervised + Unsupervised Learning**

Classes of Problems

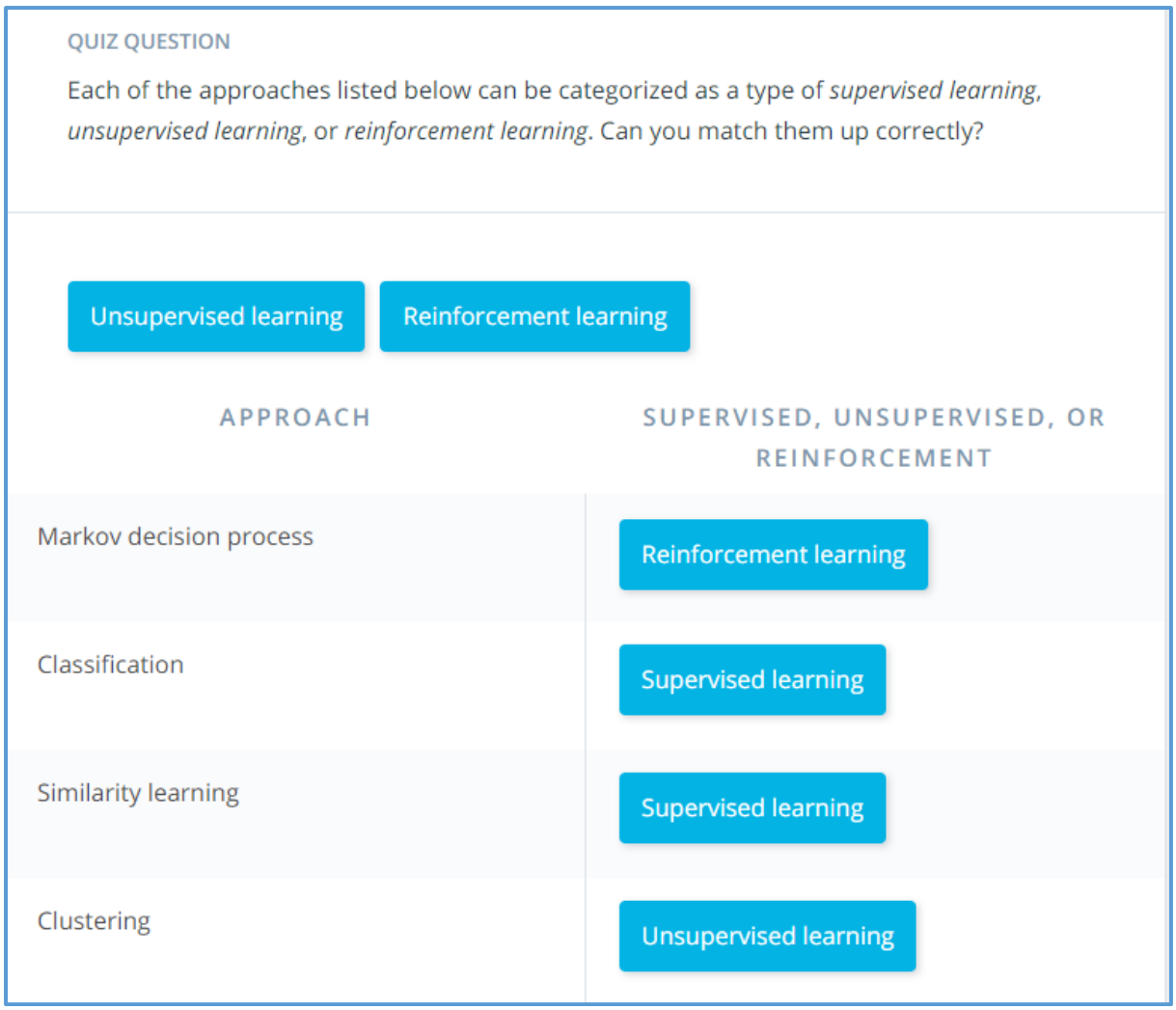
- **Classification**
- **Regression**
- **Clustering**



Specialized Cases of Model Training

Specific variations of the generic classes can be treated individually

Specialized case	Approach
Similarity Learning	Supervised
Text Classification	Supervised (classification)
Feature Learning	Supervised (classification) Unsupervised (clustering)
Anomaly Detection	Supervised (classification) Unsupervised (clustering)
Forecasting	Supervised



Chapter 10: Prelaunch Lab

Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 11: Similarity Learning

- SL is closely related to both Classification and Regression
- Uses a different type of Objective Function
- Most widely used in Recommendation systems, where aim is to understand the behavioural pattern of an user/customer and then provide recommendations of items to that user which could be products
- Often used for solving verification problems [Speech, Face and the like]

- SL is treated as a Supervised learning problem and, approachable as a Classification problem.
- Similarity function would map pairs of entities to a finite number of similarity levels ranging from 0/1 to any number of levels. Thus Similarity function is not a continuous function, but it is a Discrete function & the output would be a certain level of similarity that was calculated or predicted for the inputs
- SL can also be, treated as a problem of Regression.
- Similarity function would map pairs of entities to numerical values.

A variation of this approach where the supervision form is weakened from an exact measure to an ordering measure is known as **ranking similarity learning**. This approach is a better fit for real-life large-scale problems.

Recommender Systems

As we mentioned above, one of the most common uses of similarity learning is in creating *recommender systems*. Let us consider these in more detail.

Recommender systems are typical application of similarity learning where the objective is to recommend one or more items to the user of the system.

The main aim of a recommendation system is to recommend one or more items to users of the system.

Examples of an item might be a movie, restaurant, book, or song.

A user might be a person, a group of persons, or another entity with item preferences.

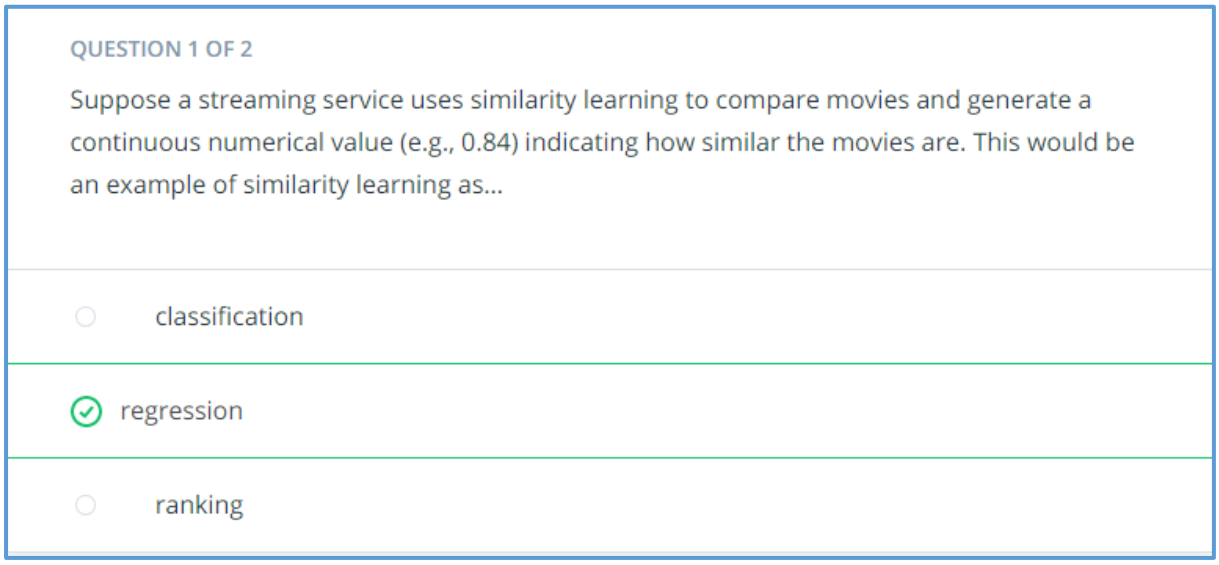
Recommender systems - approaches

Content-based	Collaborative filtering
Makes use of features for both users and items	Uses only identifiers for users and items
User properties: age, gender, region etc.	Gets information from a matrix of ratings
Item properties: author, manufacturer etc.	Ratings can be explicit or implicit

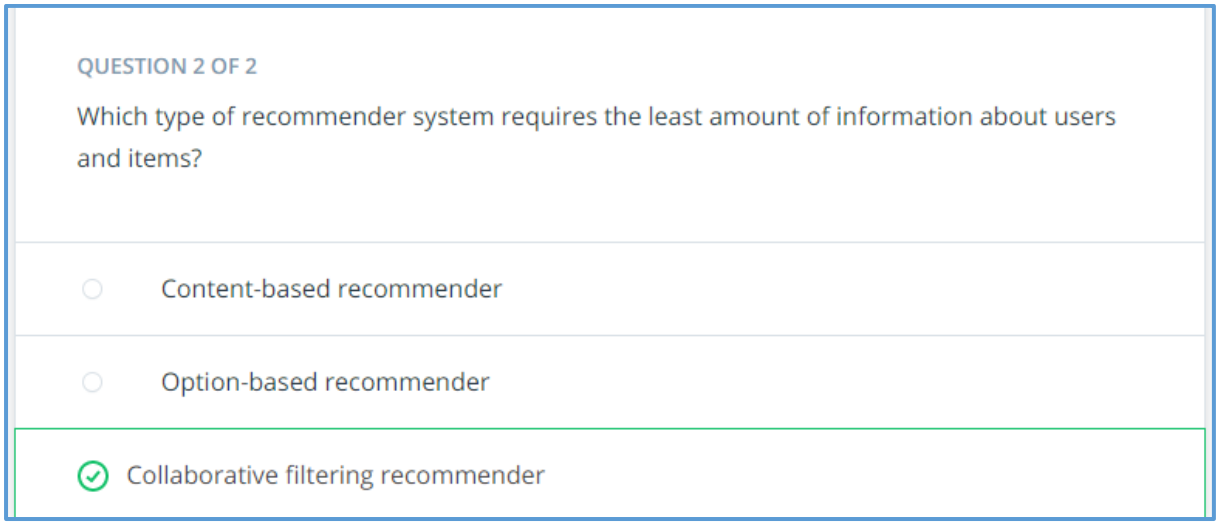
Content Based – We are making use of the properties or of the features of both users and items.

Collaborative filtering – We take into account only identifiers for users and items; we do not take into account their properties. We calculate the recommender properties or ratings from a Matrix. We create a matrix from preferences between users and items and we get information from the matrix of ratings.

Ratings can be explicit [Recorded In a system] or implicit [History of purchases or browsing history of a website etc.].



Note - Collaborative filtering recommender only requires identification information for both users and items. It gets the rest of the information it needs from a matrix of ratings that can be either explicit or implicit.



Chapter 12: Lab: Train a Simple Recommender

Train a simple recommender

The main aim of a recommendation system is to recommend one or more items to users of the system. Examples of an item to be, recommended might be a movie, restaurant, book, or song. In

general, the user is an entity with item preferences such as a person, a group of persons, or any other type of entity you can imagine.

There are two principal approaches to recommender systems:

- The content-based approach, which makes use of features for both users and items. Users can be, described by properties such as age or gender. Items can be, described by properties such as the author or the manufacturer. Typical examples of content-based recommendation systems can be, found on social matchmaking sites.
- The Collaborative filtering approach, which uses only identifiers of the users and the items. It is, based on a matrix of ratings given by the users to the items. The main source of information about a user is the list the items they have rated and the similarity with other users who have rated the same items.

The SVD recommender module in Azure Machine Learning designer is, based on the Single Value Decomposition algorithm. It uses identifiers of the users and the items, and a matrix of ratings given by the users to the items. It is a typical example of collaborative recommender.

Lab Overview

In this lab, we make use of the Train SVD Recommender module available in Azure Machine Learning designer (preview), to train a movie recommender engine. We use the collaborative filtering approach: the model learns from a collection of ratings made by users on a subset of a catalog of movies. Two open datasets available in Azure Machine Learning designer are used the IMDB Movie Titles dataset joined on the movie identifier with the Movie Ratings dataset. The Movie Ratings data consists of approximately 225,000 ratings for 15,742 movies by 26,770 users, extracted from Twitter using techniques described in the original paper by Dooms, De Pessemier and Martens. The paper and data can be, found on GitHub.

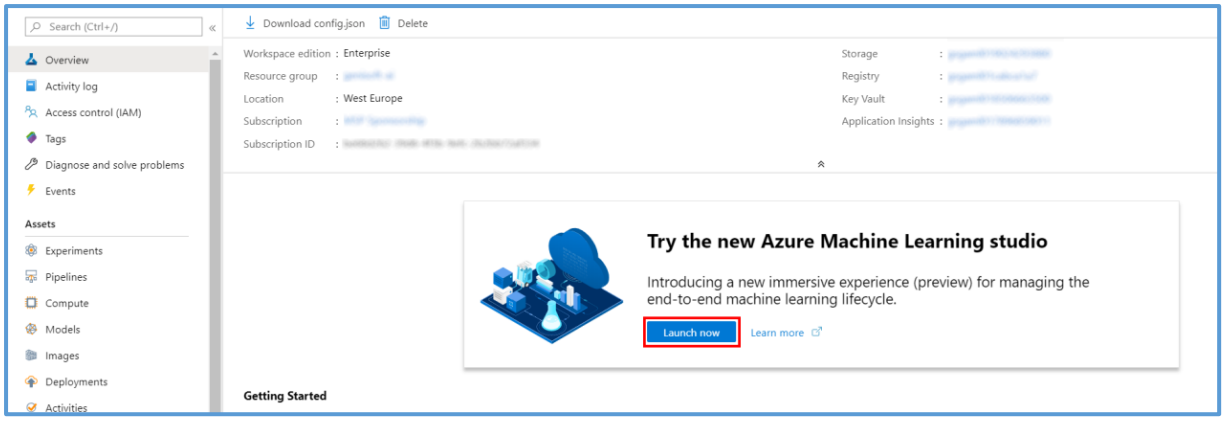
We will both train the engine and score new data, to demonstrate the different modes in which a recommender can be, used and evaluated. The trained model will predict what rating a user will give to unseen movies, so we will be able to recommend movies that the user is most likely to enjoy. We will do all of this from the Azure Machine Learning designer without writing a single line of code.

Exercise 1: Create New Training Pipeline

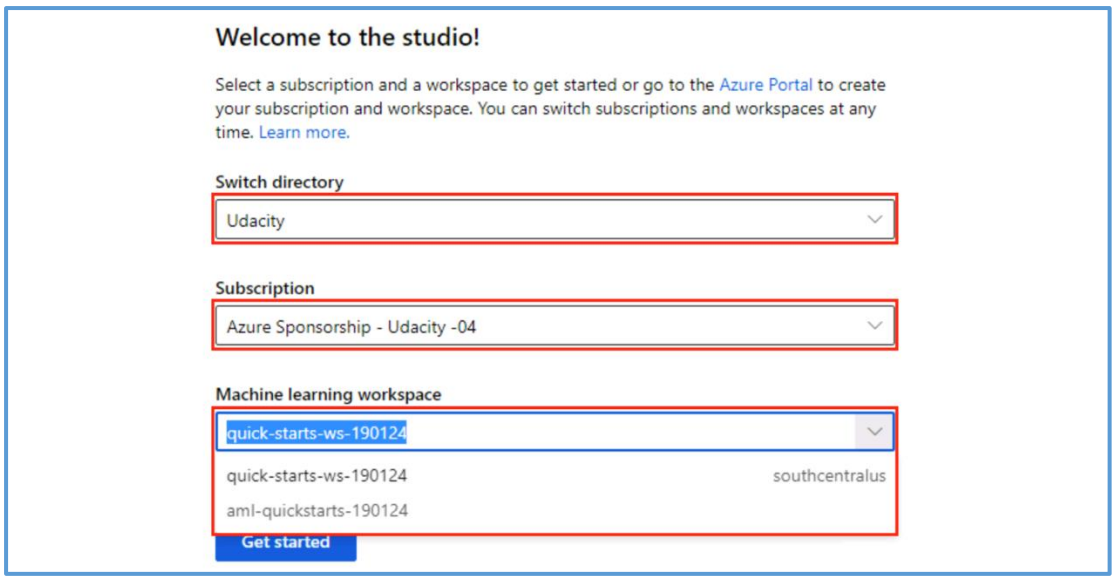
Task 1: Open Pipeline Authoring Editor

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

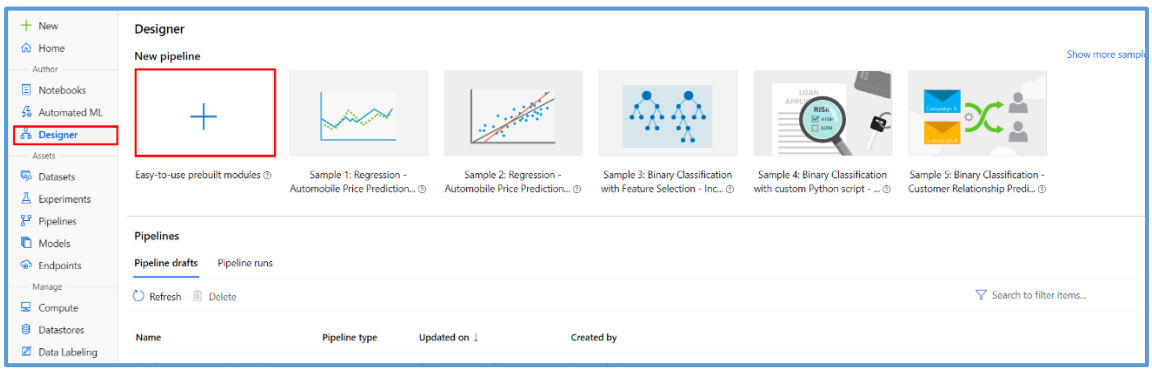


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:



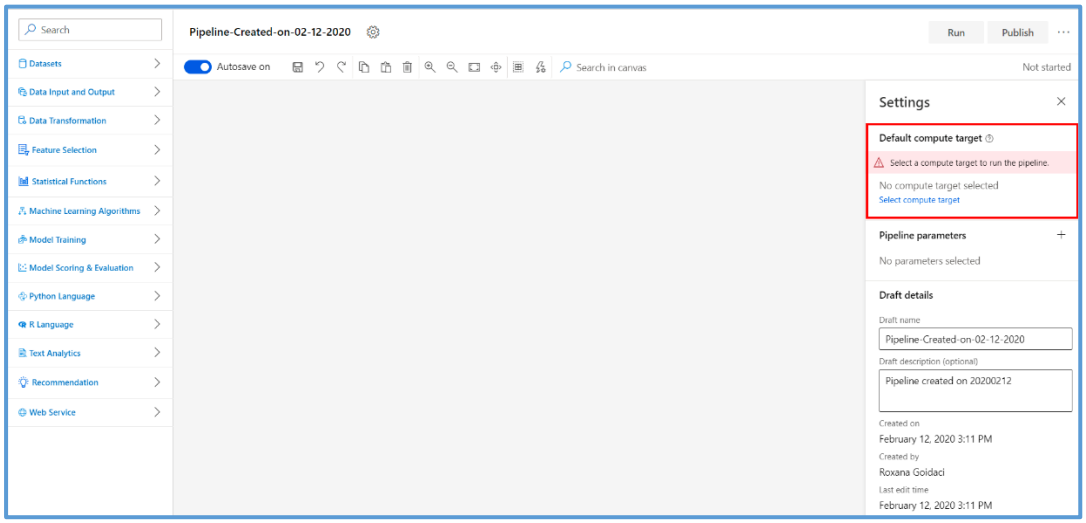
For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it does not matter which) and then click **Get started**.

4. From the studio, select **Designer, +**. This will open a visual pipeline authoring editor.



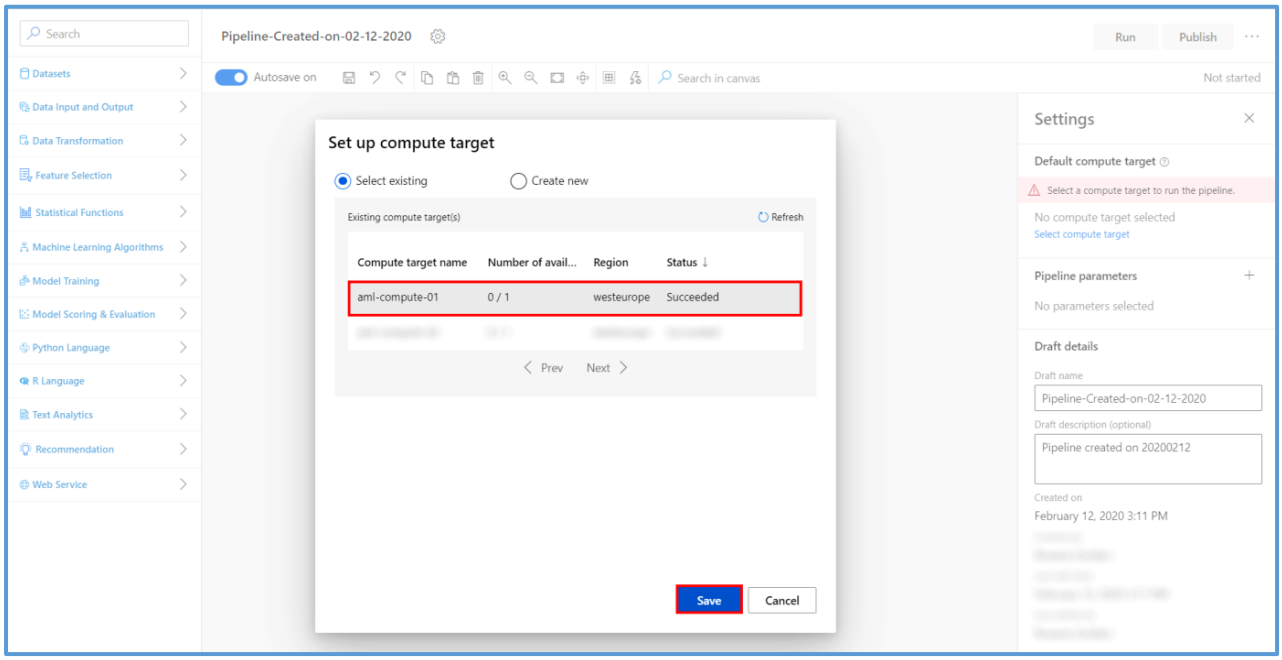
Task 2: Setup Compute Target

1. In the settings panel on the right, select **Select compute target**.



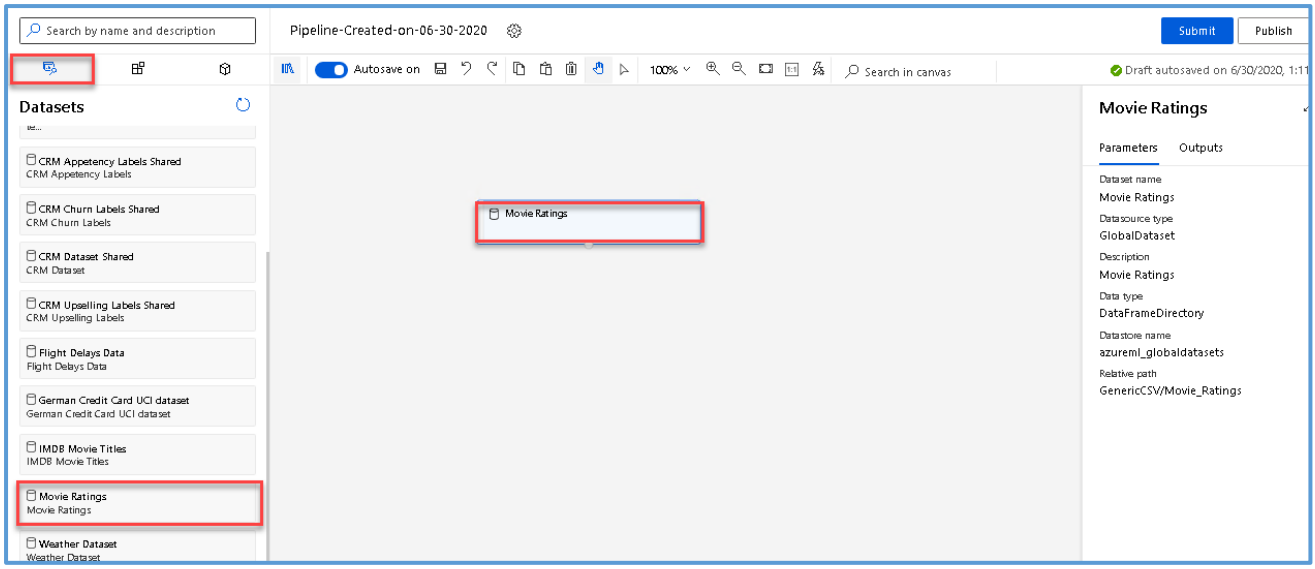
2. In the Set up compute target editor, select the available compute, and then select **Save**.

Note: If you are facing difficulties in accessing pop-up windows or buttons in the user interface, please refer to the Help section in the lab environment.

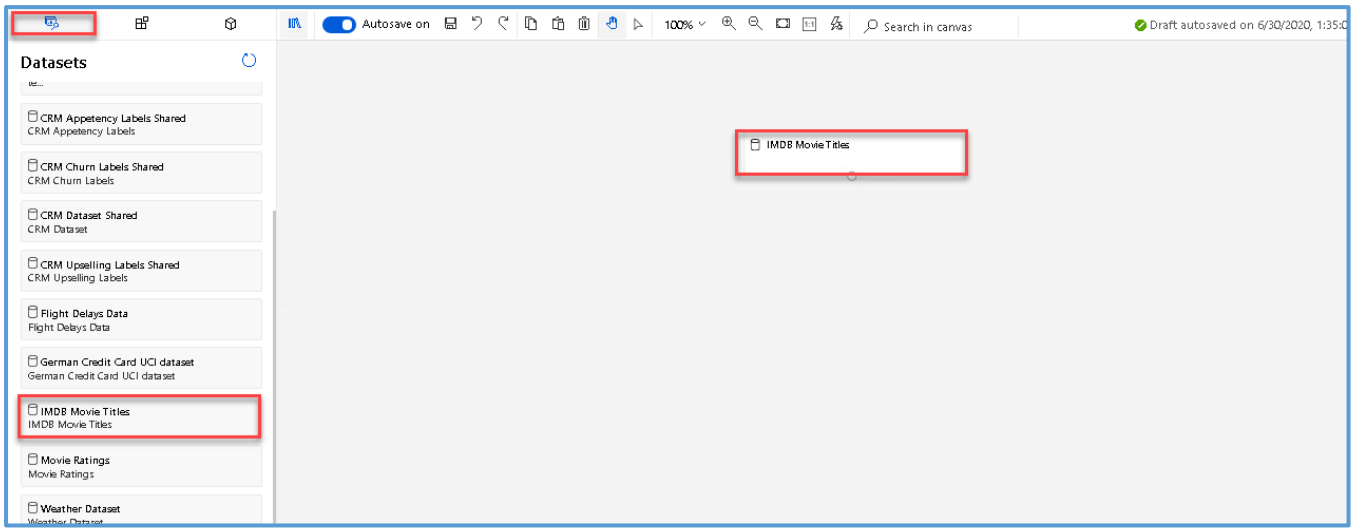


Task 3: Add Sample Datasets

1. Select **Datasets** section in the left navigation. Next, select **Samples, Movie Ratings** and drag and drop the selected dataset on to the canvas.



2. Select **Datasets** section in the left navigation. Next, select **Samples, IMDB Movie Titles** and drag and drop the selected dataset on to the canvas.



Task 4: Join the two datasets on Movie ID

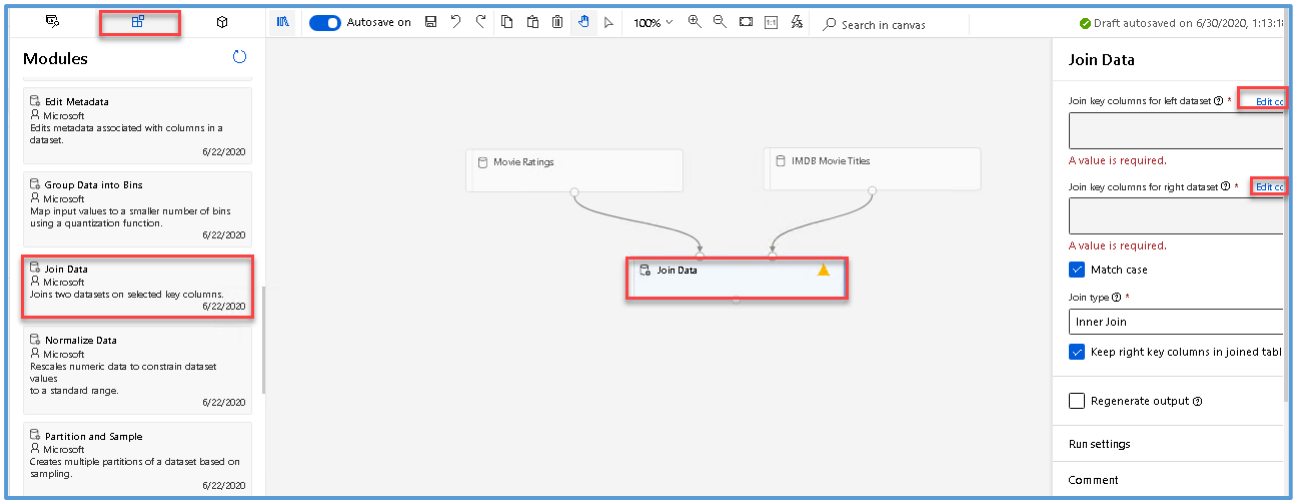
1. Select **Data Transformation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Join Data** prebuilt module

    2. Drag and drop the selected module on to the canvas

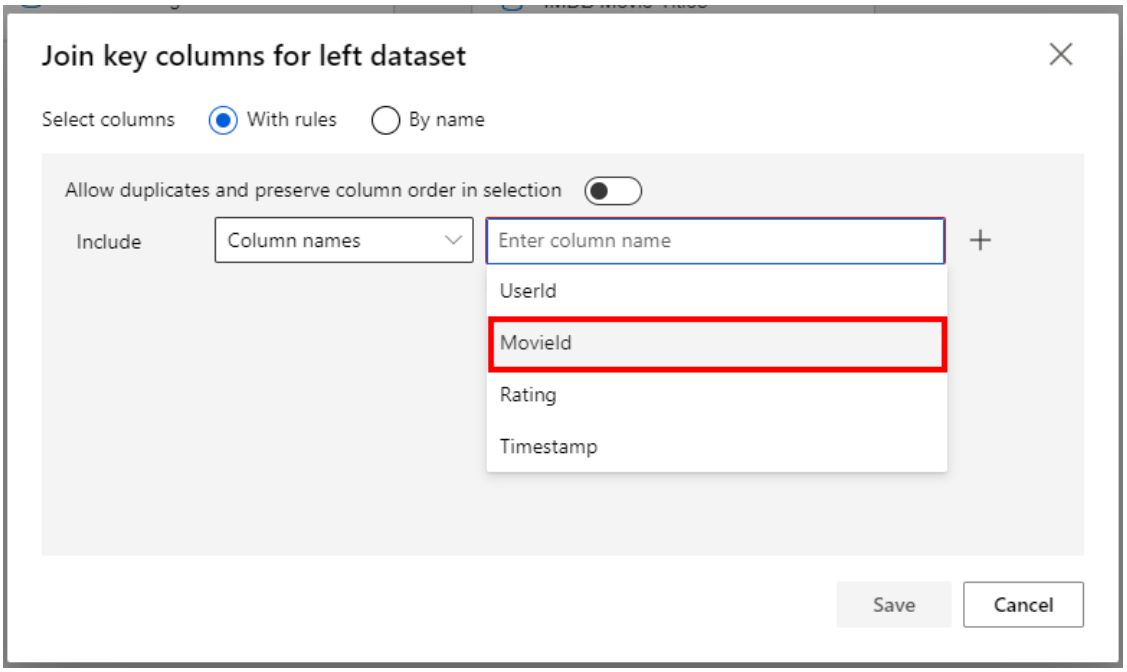
    3. Connect the output of the `Movie Ratings` module to the first input of the `Join Data` module.

    4. Connect the output of the `IMDB Movie Titles` module to the second input of the `Join Data` module.

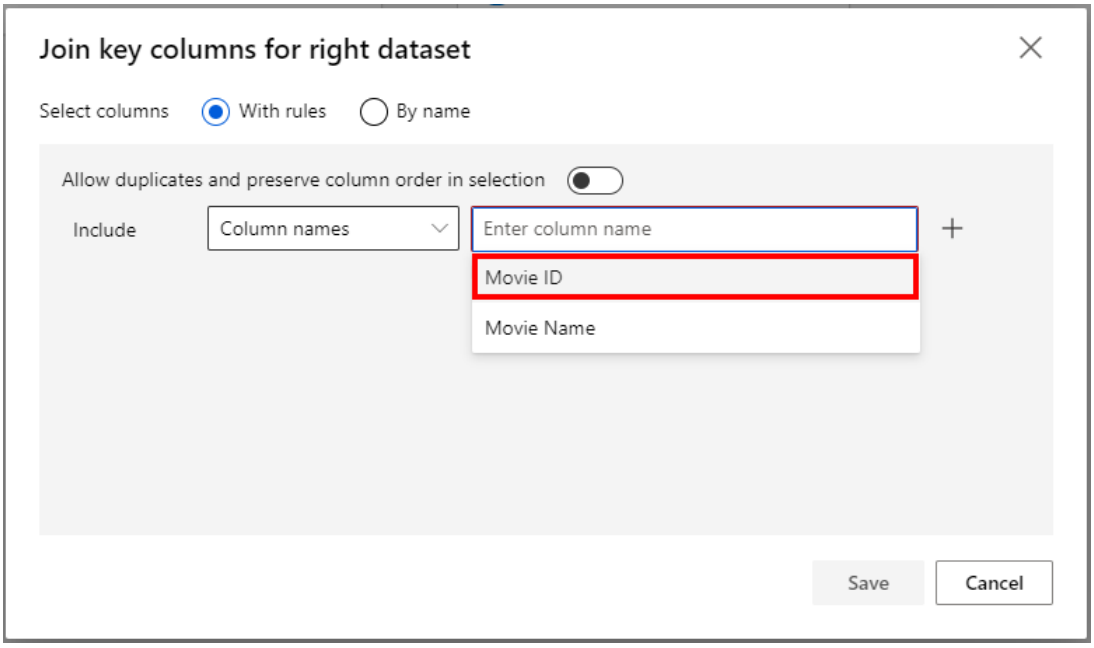


2. Select the Join Data module.

3. Select the **Edit column** link to open the Join key columns for left dataset editor. Select the **MovieId** column in the Enter column name field.



4. Select the **Edit column** link to open the Join key columns for right dataset editor. Select the **Movie ID** column in the Enter column name field.



Note that you can submit the pipeline at any point to peek at the outputs and activities. Running pipeline also generates metadata that is available for downstream activities such selecting column names from a list in selection dialogs.

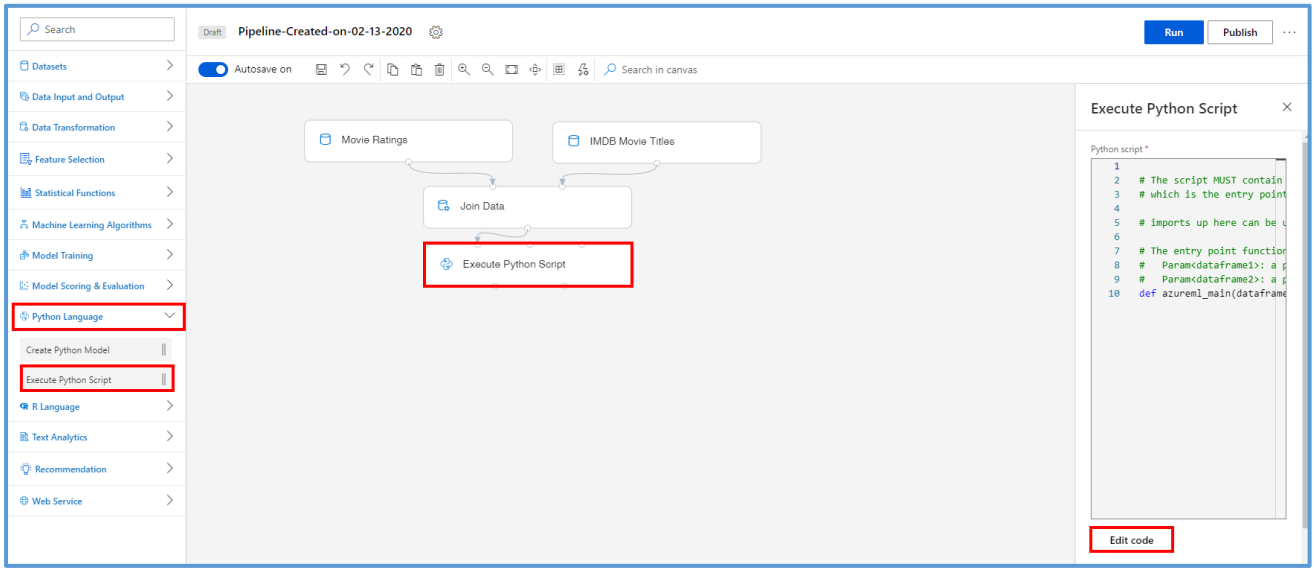
Task 5: Select Columns UserId, Movie Name, Rating using a Python script

1. Select **Python Language** section in the left navigation. Follow the steps outlined below:

    1. Select the **Execute Python Script** prebuilt module.

    2. Drag and drop the selected module on to the canvas.

    3. Connect the Join Data output to the input of the Execute Python Script module.



2. Select **Edit code** to open the `Python script` editor, clear the existing code and then enter the following lines of code to select the UserId, Movie Name, Rating columns from the joined dataset. Please ensure that there is no indentation for the first line and the second and third lines are indented.

```
def azureml_main(dataframe1 = None, dataframe2 = None):
    df1 = dataframe1[['UserId','Movie Name','Rating']]
    return df1,
```

Note: In other pipelines, for selecting a list of columns from a dataset, we could have used the `Select Columns from Dataset` prebuilt module. This one returns the columns in the same order as in the input dataset. This time we need the output dataset to be in the format: user id, movie name, rating.This column order is required at the input of the Train SVD Recommender module.

Task 6: Remove duplicate rows with same Movie Name and UserId

1. Select **Data Transformation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Remove Duplicate Rows** prebuilt module.

    2. Drag and drop the selected module on to the canvas.

    3. Connect the first output of the `Execute Python Script` to the input of the `Remove Duplicate Rows` module.

    4. Select the **Edit columns** link to open the `Select columns` editor and then enter the following list of columns to be included in the output dataset: **Movie Name**, **UserId**.

        ![Image shows the Remove Duplicate Rows module added to the canvas` editor.](images/10.png 'Add Remove Duplicate Rows module')

Task 7: Split the dataset into training set (0.5) and test set (0.5)

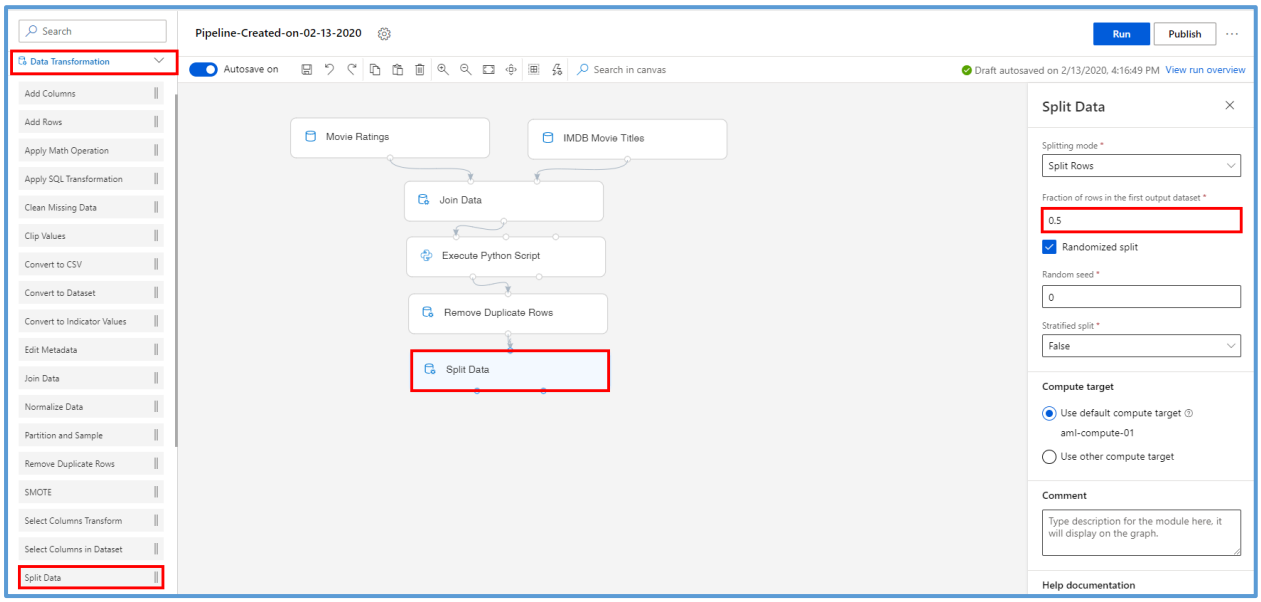
1. Select **Data Transformation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Split Data** prebuilt module

    2. Drag and drop the selected module on to the canvas

    3. Fraction of rows in the first output dataset: **0.5**

4. Connect the Dataset to the Split Data module



Task 8: Initialize Recommendation Module

1. Select **Recommendation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Train SVD Recommender** prebuilt module.

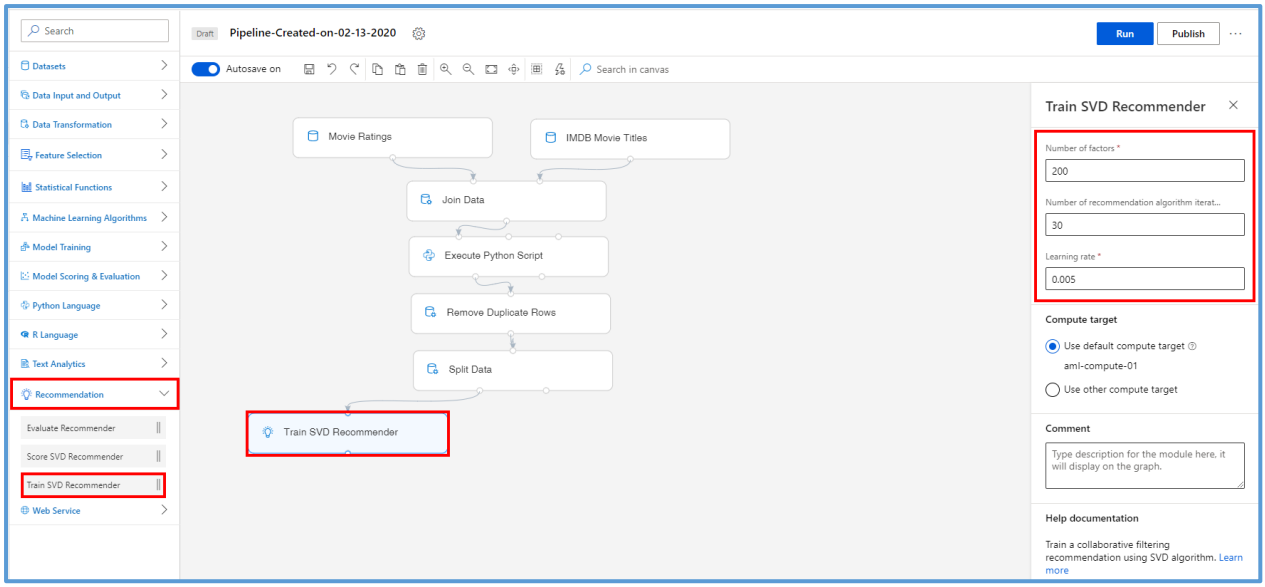
    2. Drag and drop the selected module on to the canvas

    3. Connect the first output of the Split Data module to the input of the Train SVD Recommender module

    4. Number of factors: **200**. This option specify the number of factors to use with the recommender. With the number of users and items increasing, it's better to set a larger number of factors. But if the number is too large, performance might drop.

    5. Number of recommendation algorithm iterations: **30**. This number indicates how many times the algorithm should process the input data. The higher this number is, the more accurate the predictions are. However, a higher number means slower training. The default value is 30.

    6. For Learning rate: **0.001**. The learning rate defines the step size for learning.



Task 9: Select Columns UserId, Movie Name from the test set

1. Select **Data Transformation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Select Columns in Dataset** prebuilt module.

    2. Drag and drop the selected module on to the canvas.

    3. Connect the `Split Data` second output to the input of the `Select columns in Dataset` module.

    4. Select the **Edit columns** link to open the `Select columns` editor and then enter the following list of columns to be included in the output dataset: **UserId**, **Movie Name**.

        ![Image shows the Select Columns in Dataset module added to the canvas` editor.](images/13.png 'Add Select Columns in Dataset module')

Task 10: Configure the Score SVD Recommender

1. Select **Recommendation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Score SVD Recommender** prebuilt module.

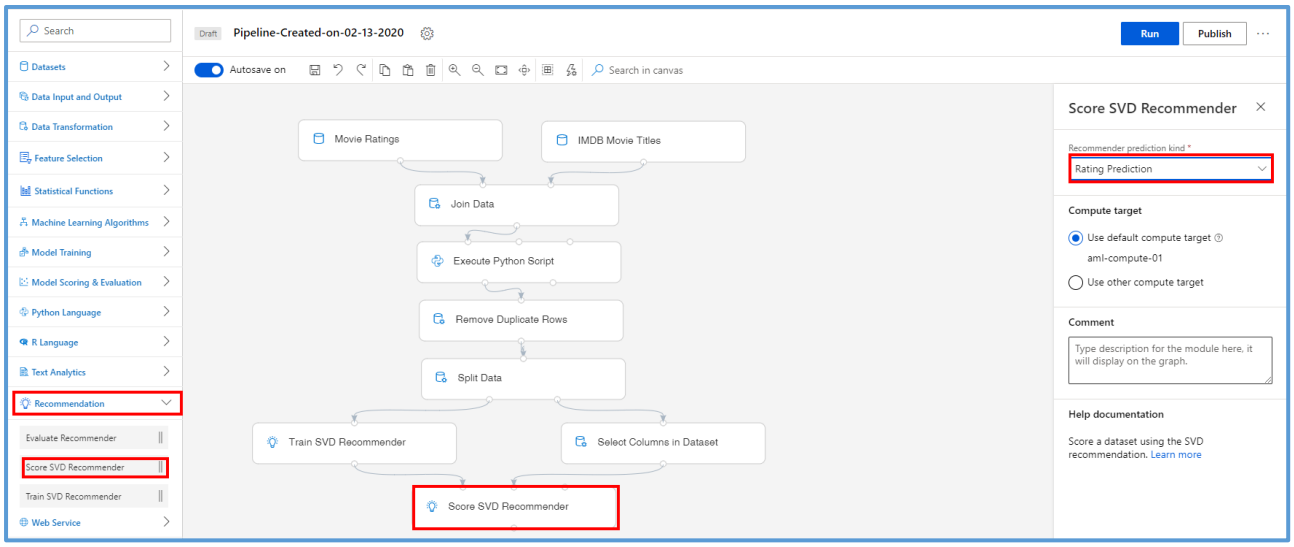
    2. Drag and drop the selected module on to the canvas

    3. Connect the output of the `Train SVD Recommender` module to the first input of the `Score SVD Recommender` module, which is the Trained SVD Recommendation input.

4. Connect the output of the Select Columns in Dataset module to the second input of the Score SVD Recommender module, which is the Dataset to score input.

5. Select the Score SVD Recommender module on the canvas.

6. Recommender prediction kind: **Rating Prediction**. For this option, no other parameters are required. When you predict ratings, the model calculates how a user will react to a particular item, given the training data. The input data for scoring must provide both a user and the item to rate.



Task 11: Setup Evaluate Recommender Module

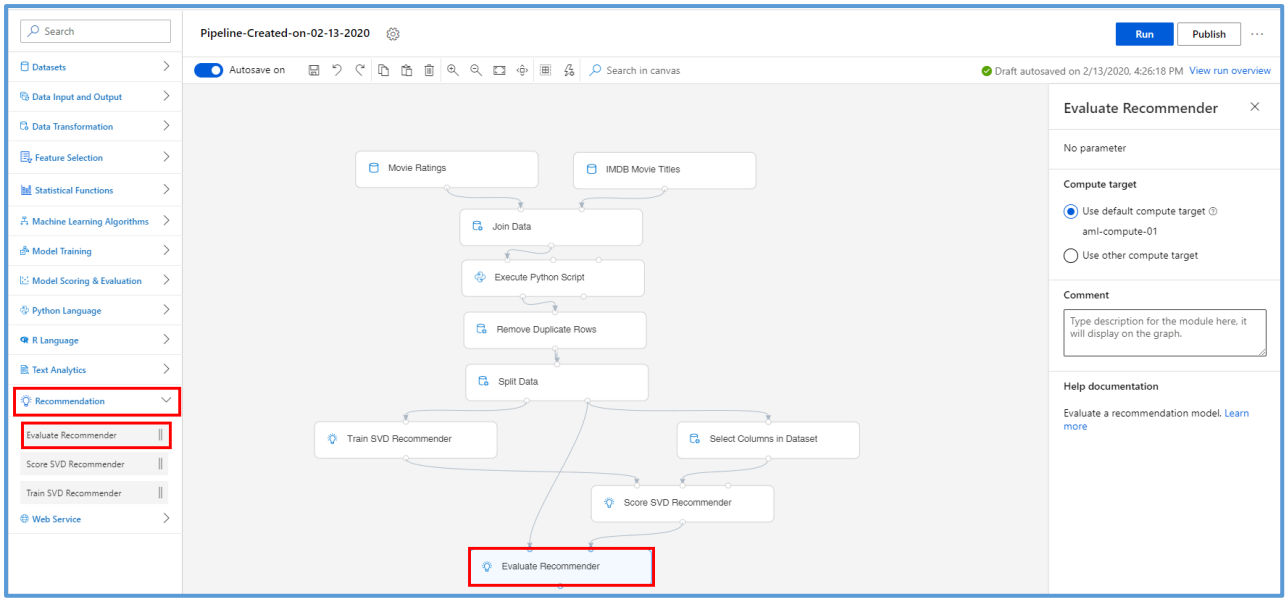
1. Select **Recommendation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Evaluate Recommender** prebuilt module

    2. Drag and drop the selected module on to the canvas

    3. Connect the Score SVD Recommender module to the second input of the Evaluate Recommender module, which is the Scored dataset input.

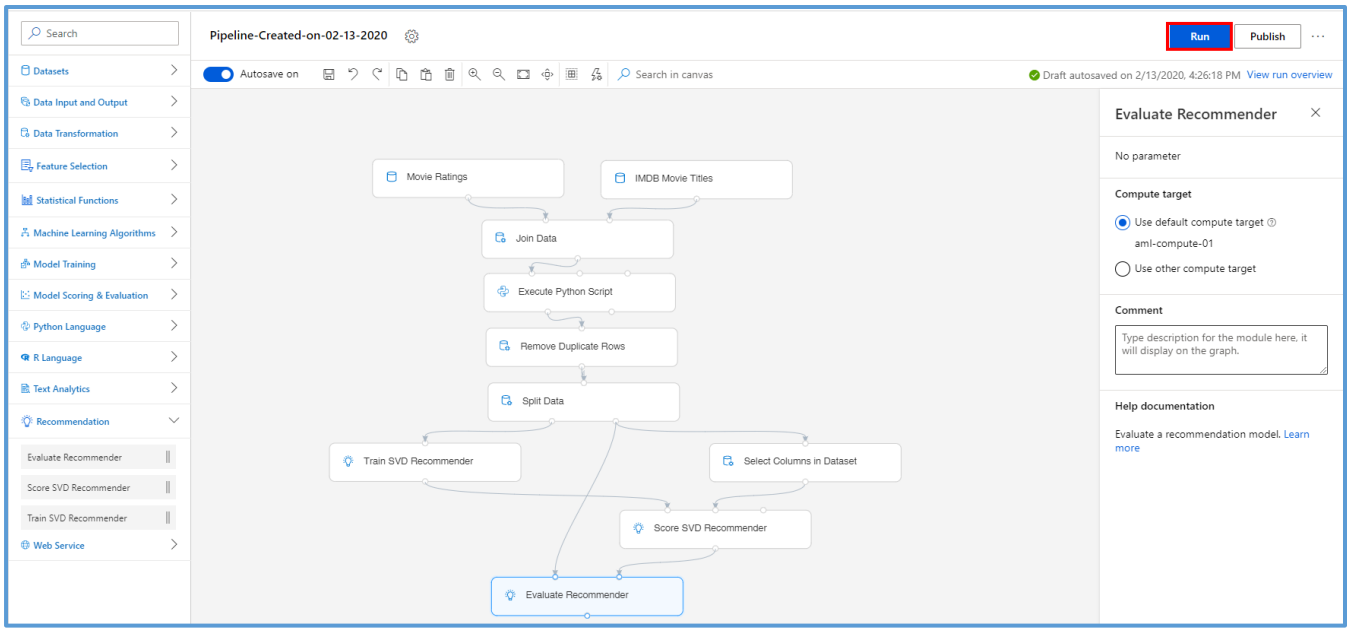
    4. Connect the second output of the Split Data module (train set) to the first input of the Evaluate Recommender module, which is the Test dataset input.



Exercise 2: Submit Training Pipeline

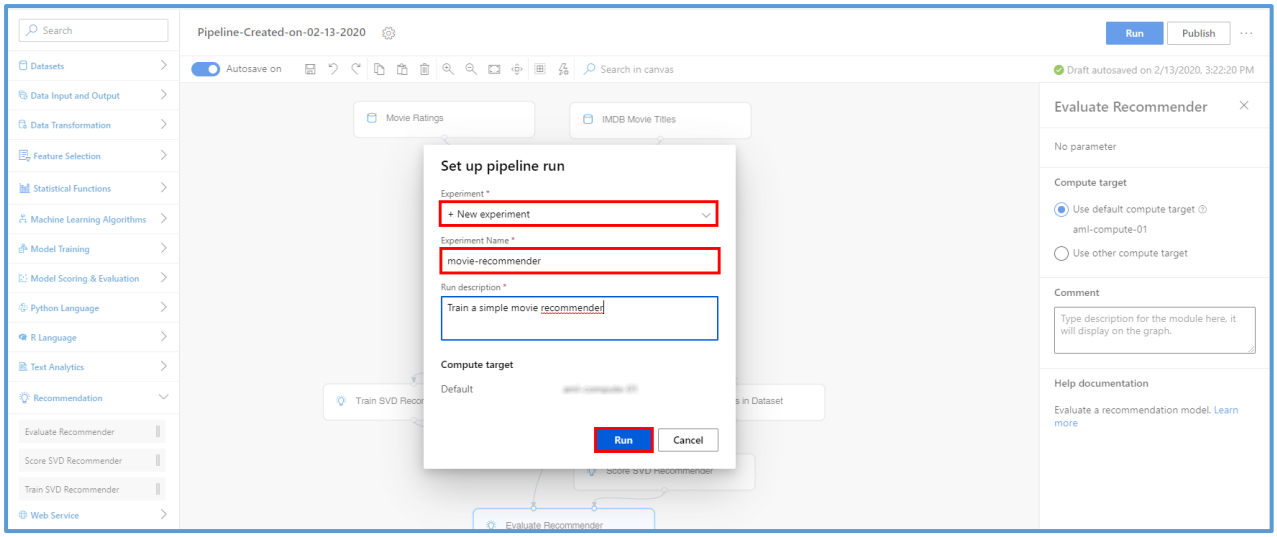
Task 1: Create Experiment and Submit Pipeline

1. Select **Submit** on the right corner of the canvas to open the Setup pipeline run editor.



Please note that the button name in the UI is, changed from **Run** to **Submit**.

2. In the Setup pipeline run editor, select **Experiment, Create new** and provide New experiment name: **movie-recommender**, and then select **Submit**.



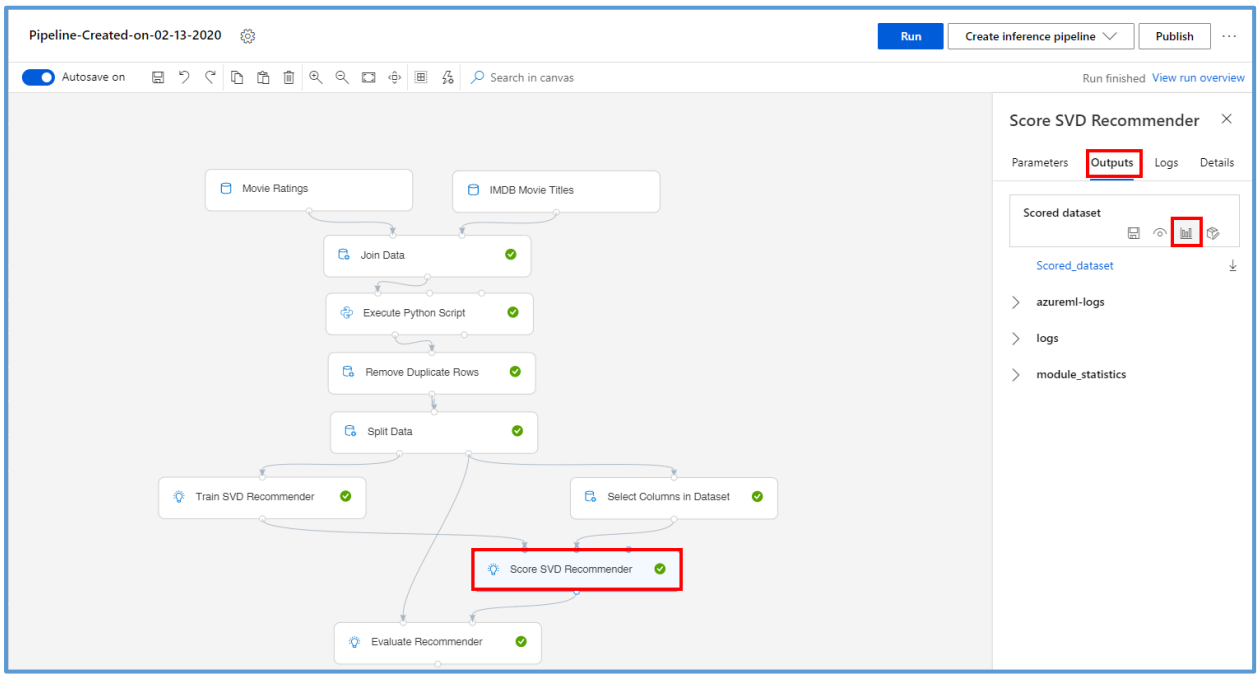
3. Wait for pipeline run to complete. It will take around **20 minutes** to complete the run.

4. While you wait for the model training to complete, you can learn more about the SVD algorithm used in this lab by selecting Train SVD Recommender.

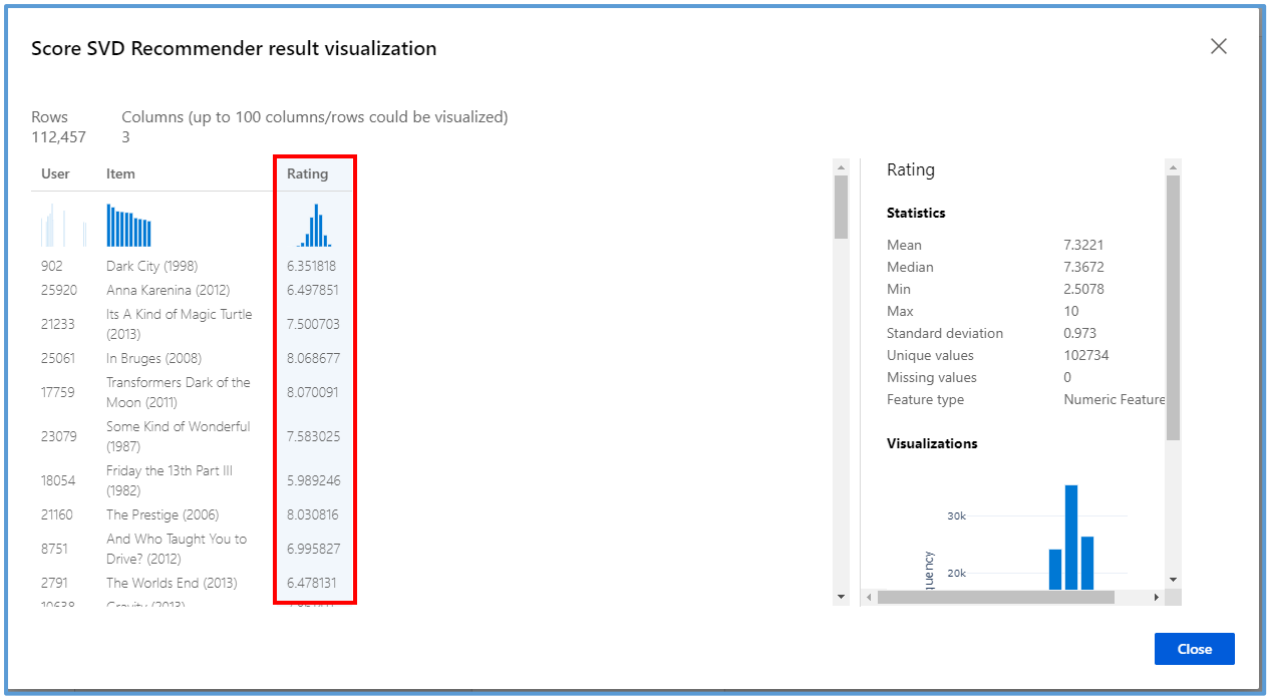
Exercise 3: Visualize Scoring Results

Task 1: Visualize the Scored dataset

1. Select **Score SVD Recommender, Outputs, Visualize** to open the Score SVD Recommender result visualization dialog or just simply right-click the Score SVD Recommender module and select **Visualize Scored dataset**.

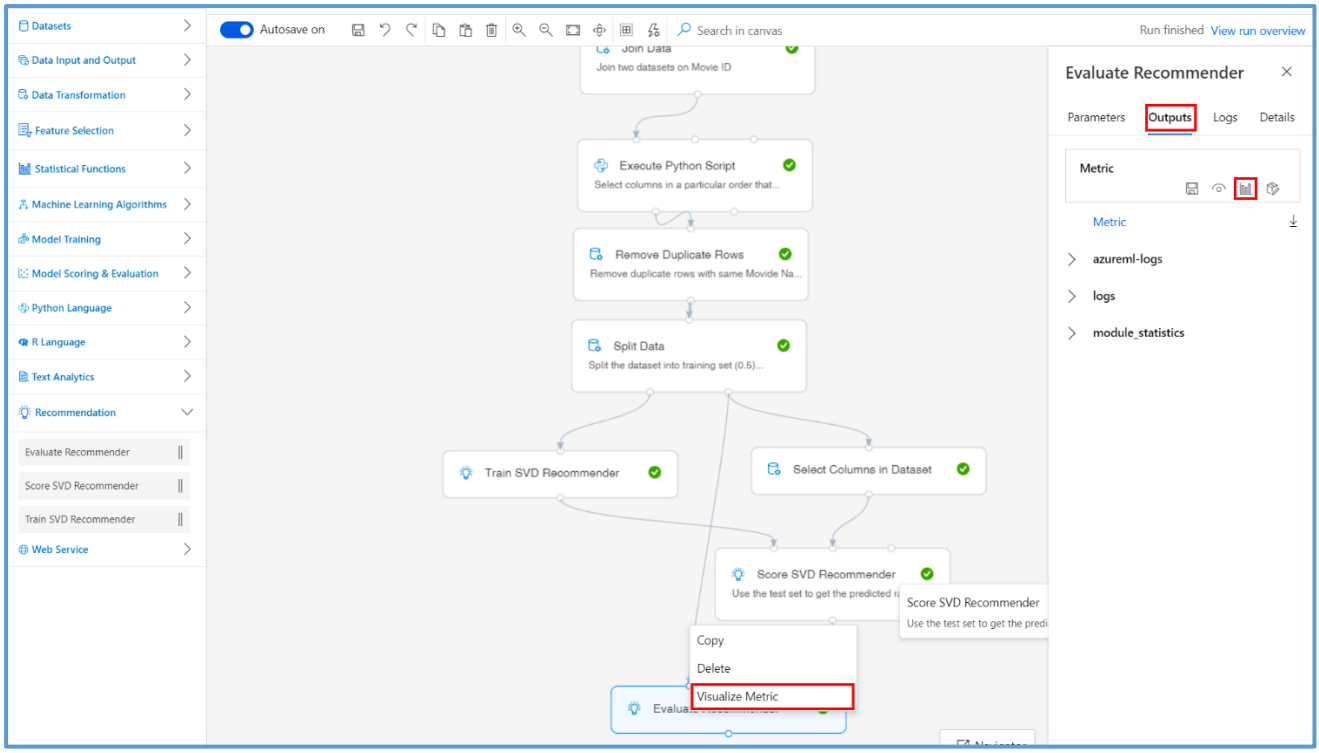


2. Observe the predicted values under the column **Rating**.

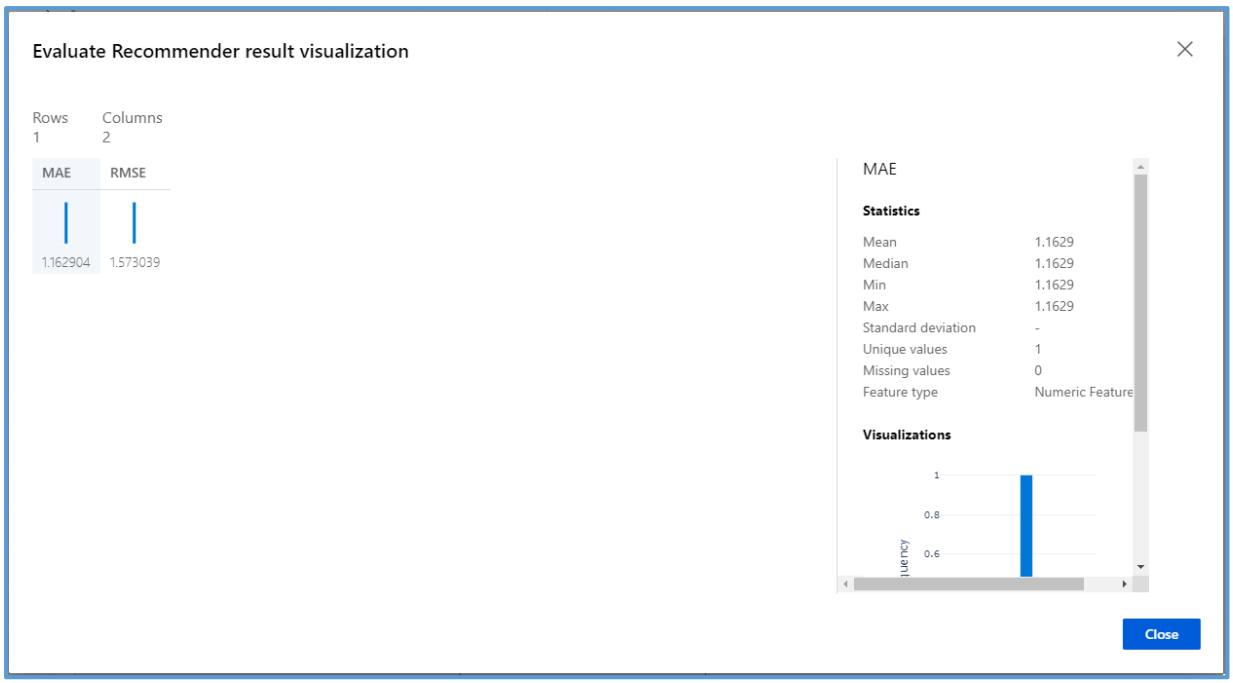


Task 2: Visualize the Evaluation Results

1. Select **Evaluate Recommender, Outputs, Visualize** to open the Evaluate Recommender result visualization dialog or just simply right-click the Evaluate Recommender module and select **Visualize Evaluation Results**.



2. Evaluate the model performance by reviewing the various evaluation metrics, such as **Mean Absolute Error**, **Root Mean Squared Error**, etc.

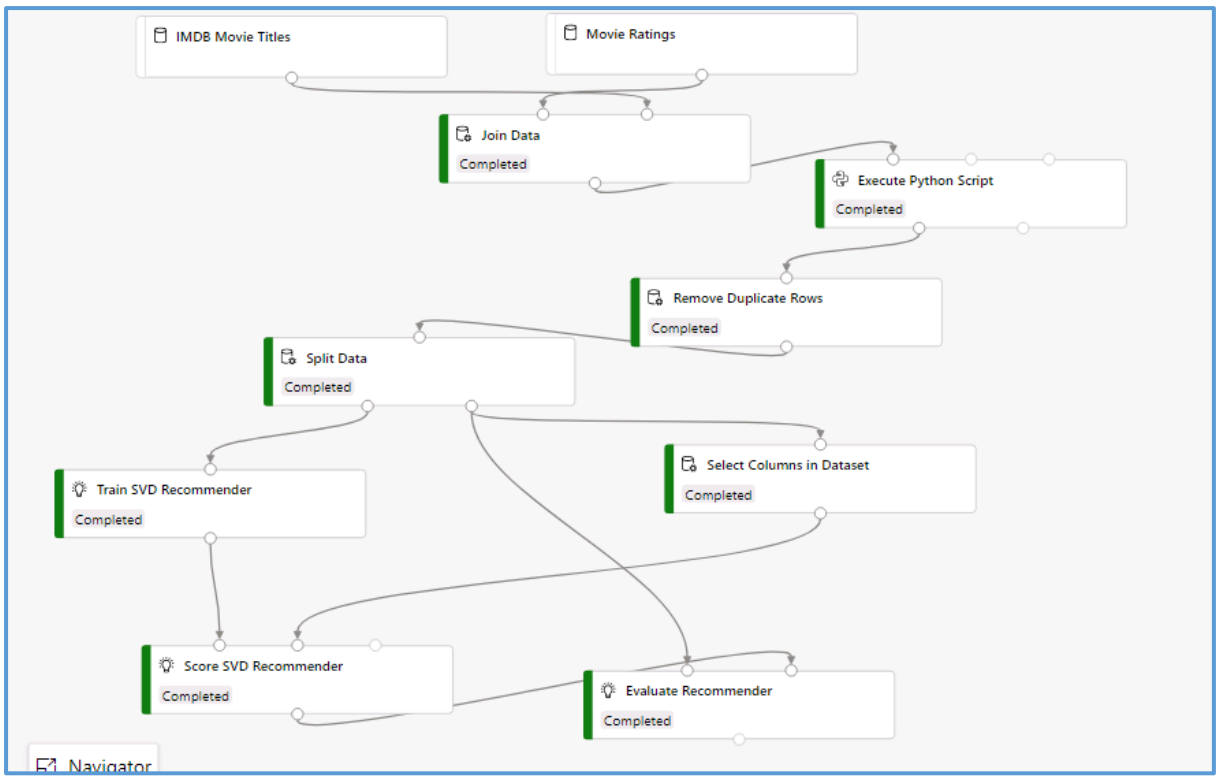


Next Steps

Congratulations! You have trained a simple movie recommender using the prebuilt Recommender modules in the AML visual designer. You can continue to experiment in the environment but are free to close the lab environment tab and return to the Udacity portal to continue with the lesson.

Chapter 13: Walkthrough: Train a Simple Recommender

- **One application of Similarity Learning is Recommender System.**
- **Build a simple Recommender system for which we have used two model datasets.**
- **Movie Ratings and IMDB Movie Titles dataset have been, selected.**
- **Both datasets have been joined using the "MovieId"[Join keys for left and right dataset] columns**
- **Python script has been, executed to reorder the columns, which is important for a recommender system.**
- **Duplicate rows have been, removed for Movie Name and UserId columns. Thus, we would have unique rows.**
- **Data is splitted – 50% Training and 50% Test data**
- **Train SVD recommender helps in selecting the ML algorithm.**
- **Transformation on Test data is required. Use Select Columns in Dataset. This ensures that Test dataset has only two columns left UserId and Movie Name.**
- **Score the SVD recommender using the Training data out from Train SVD Recommender and Test dataset.**
- **Finally, in the evaluation process Test data is required at the first Input, which comes from, Split Dataset and the Second input is from Scored Model output.**



Chapter 14: Prelaunch Lab

Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 15: Text Classification

Remember that before we can do text classification, the text first needs to be, translated into some kind of numerical representation—a process known as **text embedding**. The resulting numerical representation, which is usually in the form of vectors, can then be, used as an input to a wide range of classification algorithms.

Text embedding comes in two forms:

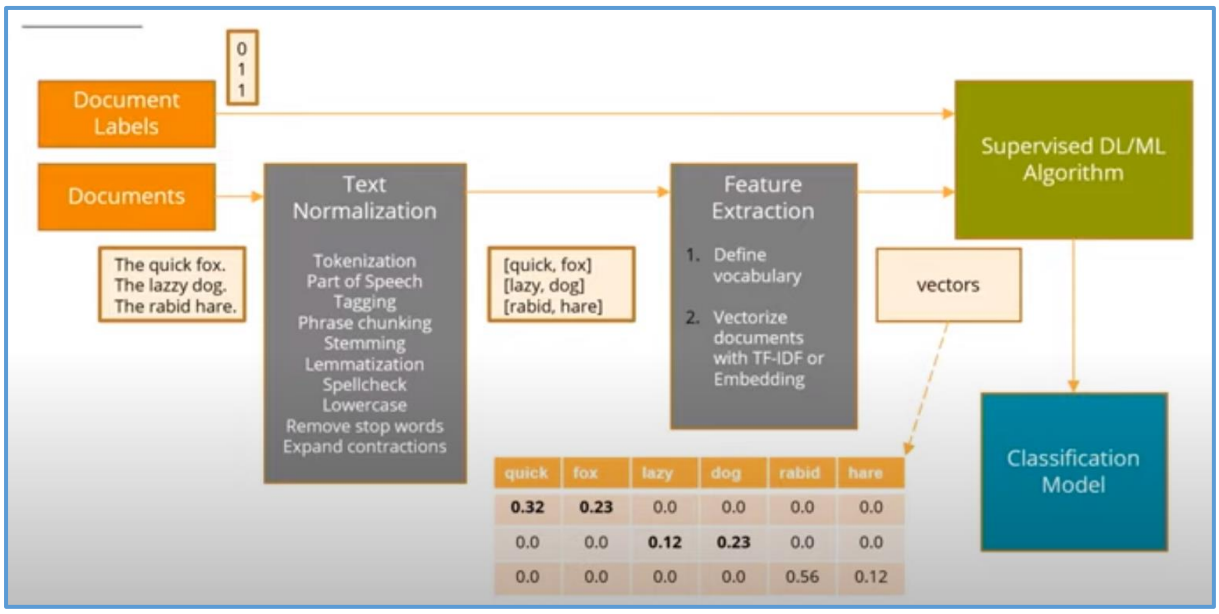
- Word Embedding – Transform every single word into a numerical vector that has a number of features or a certain dimensions, then use the representation for model training process. There is another form called as sentence embedding. We take the word embeddings and combine all the embeddings for all the words in the sentence to create a single embedding at the sentence level.
- Scoring – Scoring mechanism where the score is, related to the importance of a particular word in the text. It helps us in understanding the global properties of the text, like – Sentiment. However, it has limitations over complex tasks.

The resulting numerical representation (usually as vectors) is then used as an input to a wide range of classification algorithms

Training a Classification Model with Text

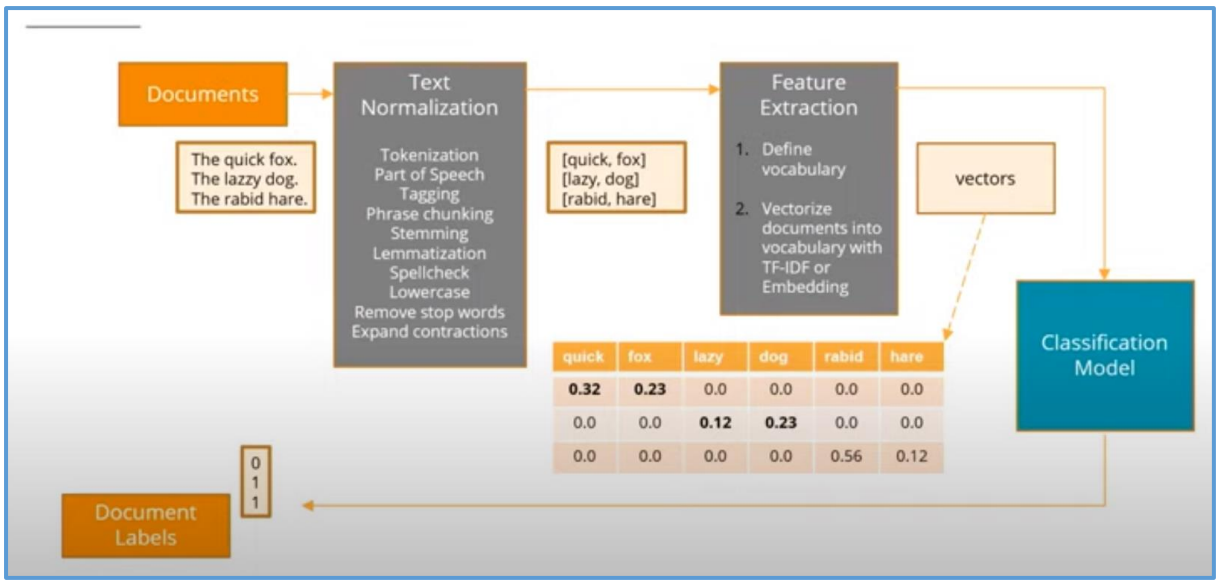
As we mentioned in the video, an important part of the pipeline is the process of vectorizing the text, using a technique such as Term Frequency-Inverse Document Frequency (TF-IDF) vectorization.

- Objective is to train the model to classify a set of documents. We start with a set of documents & their associated labels.
- Next step is to normalize the documents. [Tokenization, Parts of speech, Tagging, Phrase chunking, Stemming, Lemmatization, Spellcheck, Lowercase, Remove stop words, Expand contractions]
- Next step is Feature Extraction, which is a two-step process.
  a) Define vocabulary of the texts. Identifying individual words, based on their frequency and record them as vocabulary. Finally vectorize the documents.
  b) Once we have the numerical vector format [Using TF-IDF], we can use these dataset to train the supervised DL or Classical ML algorithm. Final output is a Classification Model.



Predicting classification from text

- Now the question is how to use this classification model for new datasets.
- Use the new set of texts and perform Text normalization, followed by Feature extraction.
- Important fact is that we need to use the same vocabulary, which we have used while training the model.
- Get the numerical representation of the new text and input the same into the Classification model.
- Classification model would produce the labels for the new documents.



QUIZ QUESTION

Which of the following is not a typical step in model training for text classification?

- ✓ Document labeling
- Text normalization
- Supervised learning model training
- Feature extraction

Chapter 16: Lab: Train a Simple Text Classifier

Train a simple text classifier

In text classification scenarios, the goal is to assign a piece of text, such as a document, a news article, a search query, an email, a tweet, support tickets, customer feedback, user product review, to predefined classes or categories. Some examples of text classification applications are: categorizing newspaper articles into topics, organizing web pages into hierarchical categories, spam email filtering, sentiment analysis, predicting user intent from search queries, support tickets routing, and customer feedback analysis.

Lab Overview

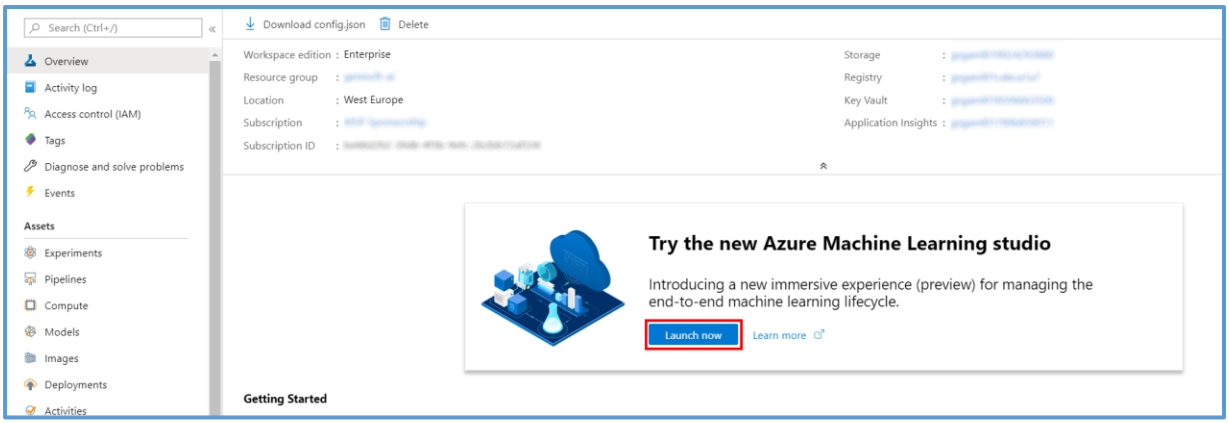
In this lab we demonstrate how to use text analytics modules available in Azure Machine Learning designer (preview) to build a simple text classification pipeline. We will create a training pipeline and initialize a **multiclass logistic regression classifier** to predict the company category with Wikipedia SP 500 dataset derived from Wikipedia. The dataset manages articles of each S&P 500 company. Before uploading to Azure Machine Learning designer, the dataset was processed as follows: extracted text content for each specific company, removed wiki formatting, removed non-alphanumeric characters, converted all text to lowercase, known company categories added. Articles could not be found for some companies, so that's why the number of records is less than 500.

Exercise 1: Create New Training Pipeline

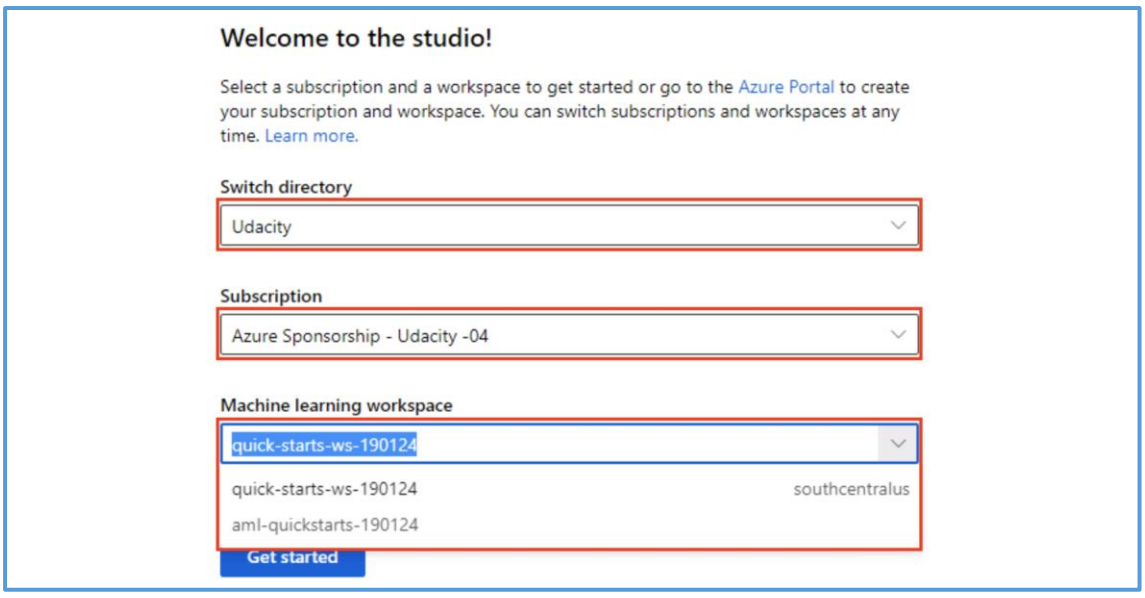
Task 1: Open Pipeline Authoring Editor

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.



3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:

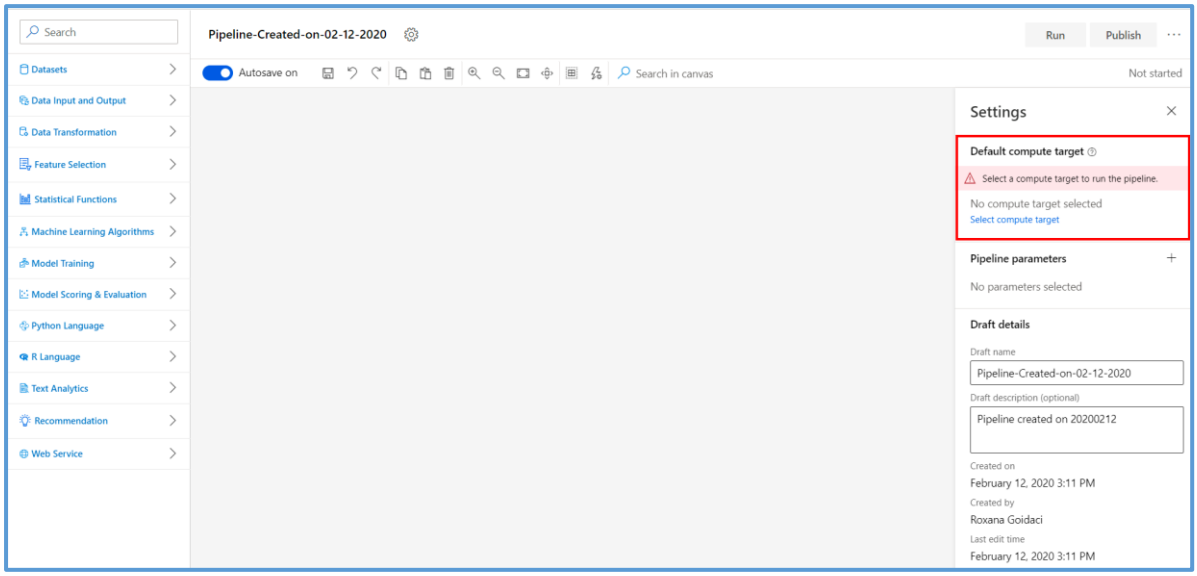


For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it does not matter which) and then click **Get started**.

4. From the studio, select **Designer, +**. This will open a visual pipeline authoring editor.

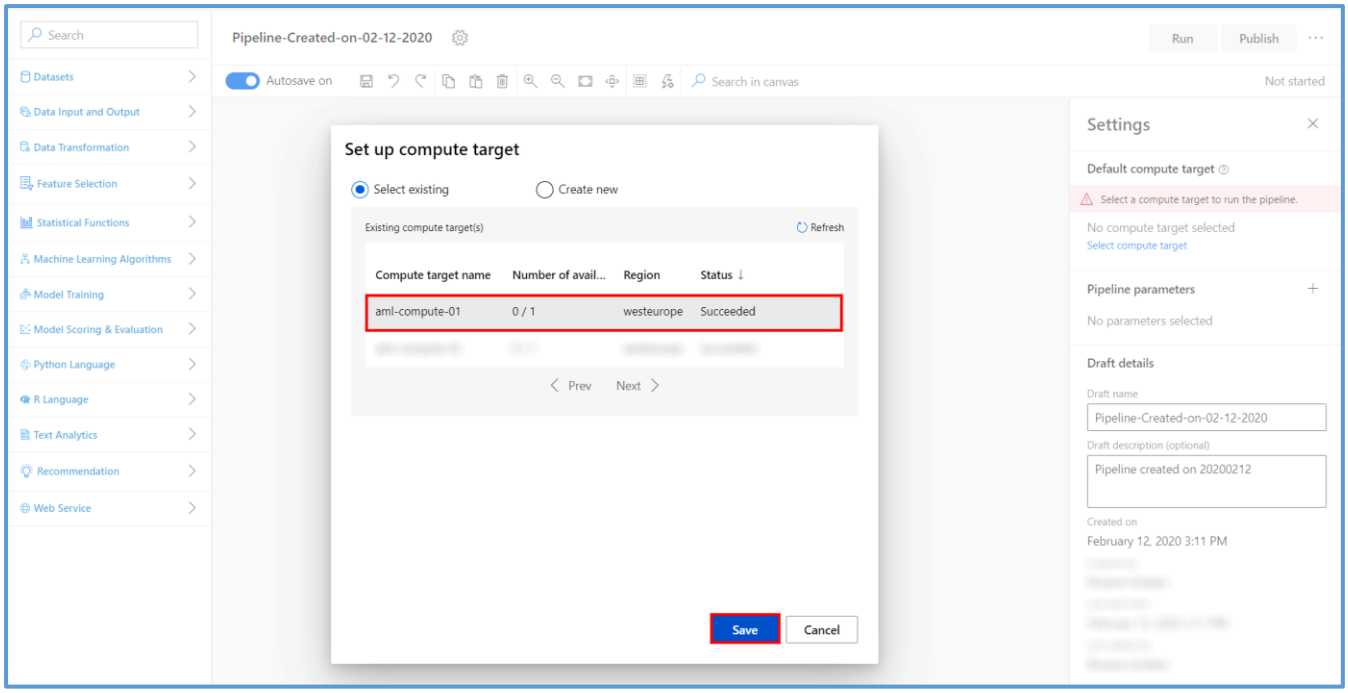
Task 2: Setup Compute Target

1. In the settings panel on the right, select **Select compute target**.



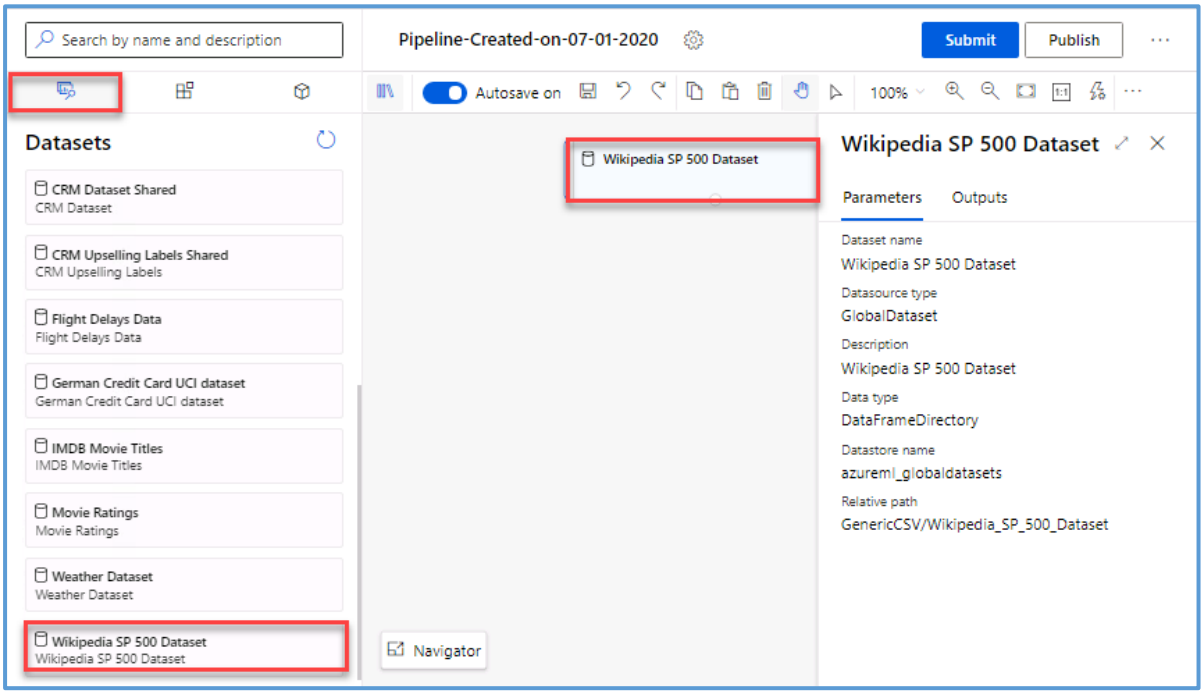
2. In the Set up compute target editor, select the available compute, and then select **Save**.

Note: If you are facing difficulties in accessing pop-up windows or buttons in the user interface, please refer to the Help section in the lab environment.



Task 3: Add Wikipedia SP 500 Sample Datasets

1. Select **Datasets** section in the left navigation. Next, select **Samples, Wikipedia SP 500 Dataset** and drag and drop the selected dataset on to the canvas.



Task 4: Preprocess text for following steps

1. Select **Text Analytics** section in the Modules. Follow the steps outlined below:

    1. Select the **Preprocess Text** prebuilt module.

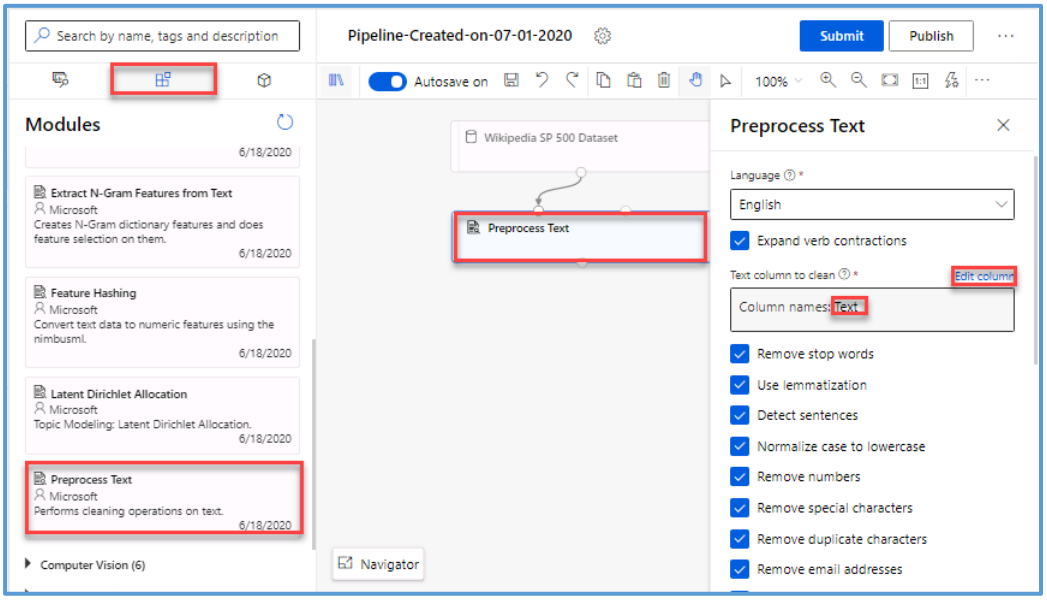
2. Drag and drop the selected module on to the canvas.

3. Connect the output of the Wikipedia SP 500 module to the input of the Preprocess Text module. The dataset input of this prebuilt module needs to be connected to a dataset that has at least one column containing text.

4. Select the language from the Language dropdown list: **English**.

5. Select the **Edit column** link to open the Text column to clean editor. Select the **Text** column in the Enter column name field.

6. Leave all the other options checked, as in the default configuration of the Preprocess Text module.



Task 5: Split the dataset into training set (0.5) and test set (0.5)

1. Select **Data Transformation** section in the Modules. Follow the steps outlined below:

    1. Select the **Split Data** prebuilt module.

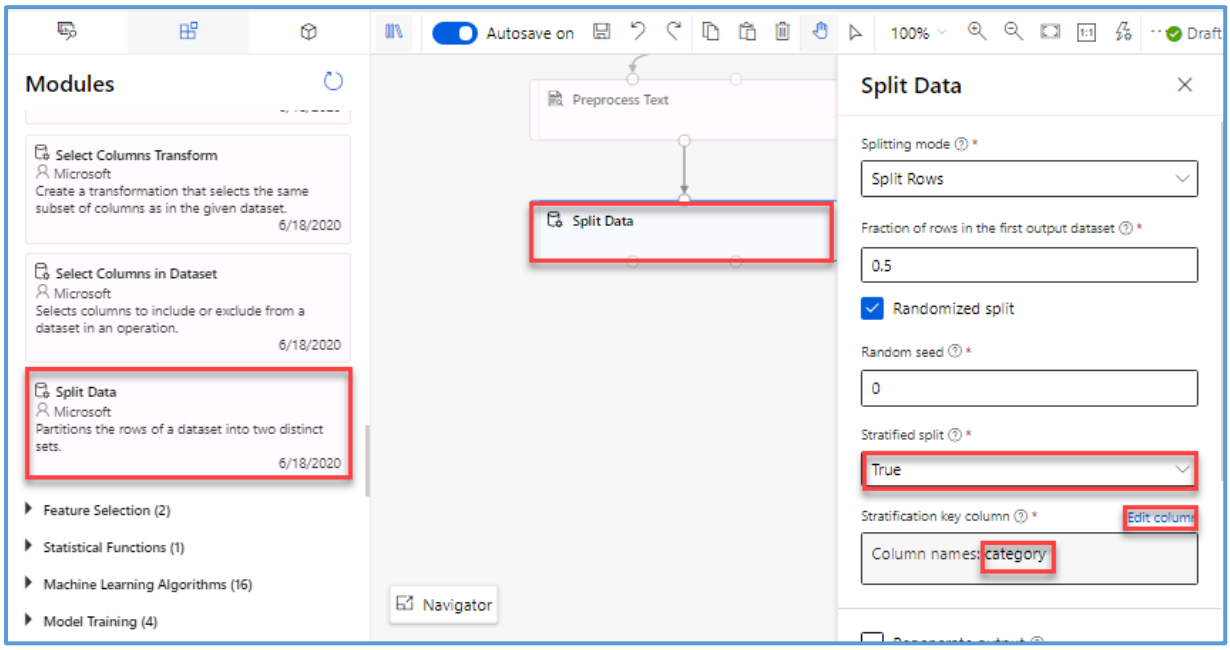
    2. Drag and drop the selected module on to the canvas.

    3. Fraction of rows in the first output dataset: **0.5**.

    4. Stratified seed: **True**.

    5. Select the **Edit column** link to open the Stratification key column editor. Select the **Category** column in the Enter column name field.

6. Connect the output of the Preprocess Text module to the input of the Split Data module.



Task 6: Convert the plain text of the articles to integers with Feature Hashing module, on the training set

1. Select **Text Analytics** section in the left navigation. Follow the steps outlined below:

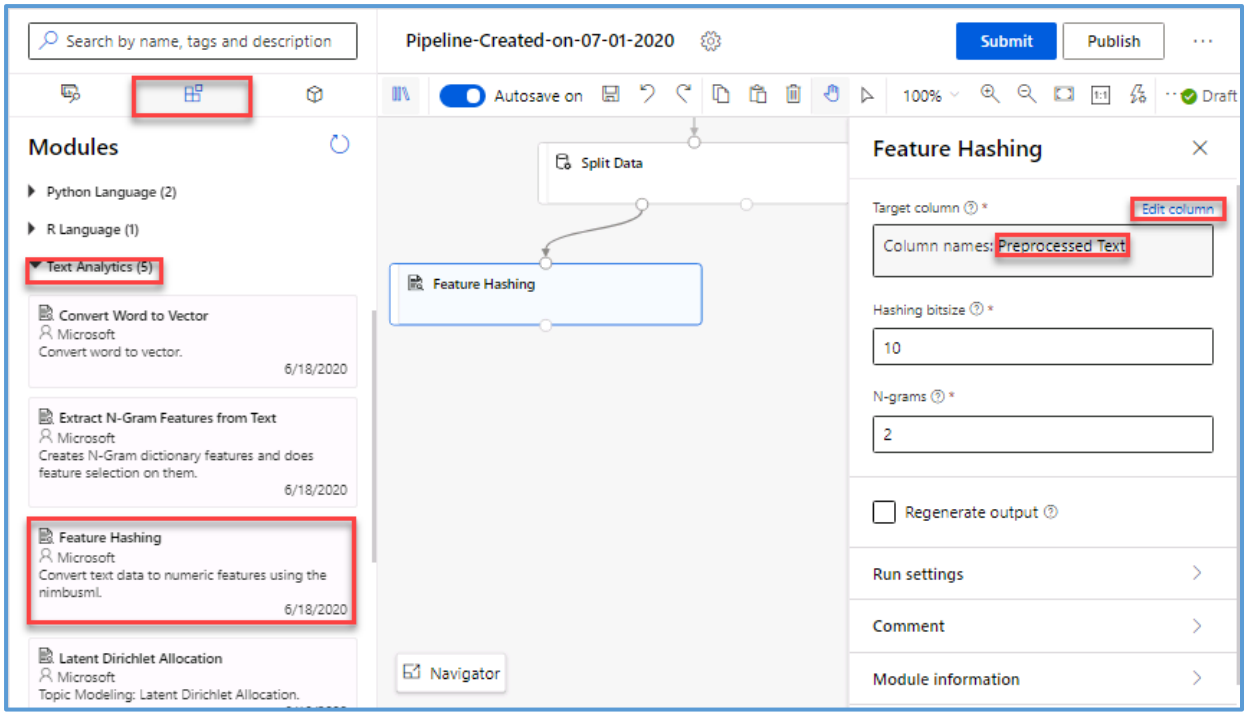
    1. Select the **Feature Hashing** prebuilt module.

    2. Drag and drop the selected module on to the canvas.

    3. Connect the first output of the Split Data to the input of the Feature Hashing module.

    4. Select the **Edit columns** link to open the Target column editor and then enter the **Preprocessed Text** column for the Target column field.

    5. Set Hashing bitsize : **10** and set the number of N-grams : **2**.



The goal of using feature hashing is to reduce dimensionality; also it makes the lookup of feature weights faster at classification time because it uses hash value comparison instead of string comparison.

Task 7: Featurize unstructured text data with Extract N-Gram Feature from Text module, on the training set

1. Select **Text Analytics** section in the left navigation. Follow the steps outlined below:

    1. Select the **Extract N-Gram Feature from Text** prebuilt module.

    2. Drag and drop the selected module on to the canvas.

    3. Connect the first output of the Split Data to the input of the Extract N-Gram Feature from Text module.

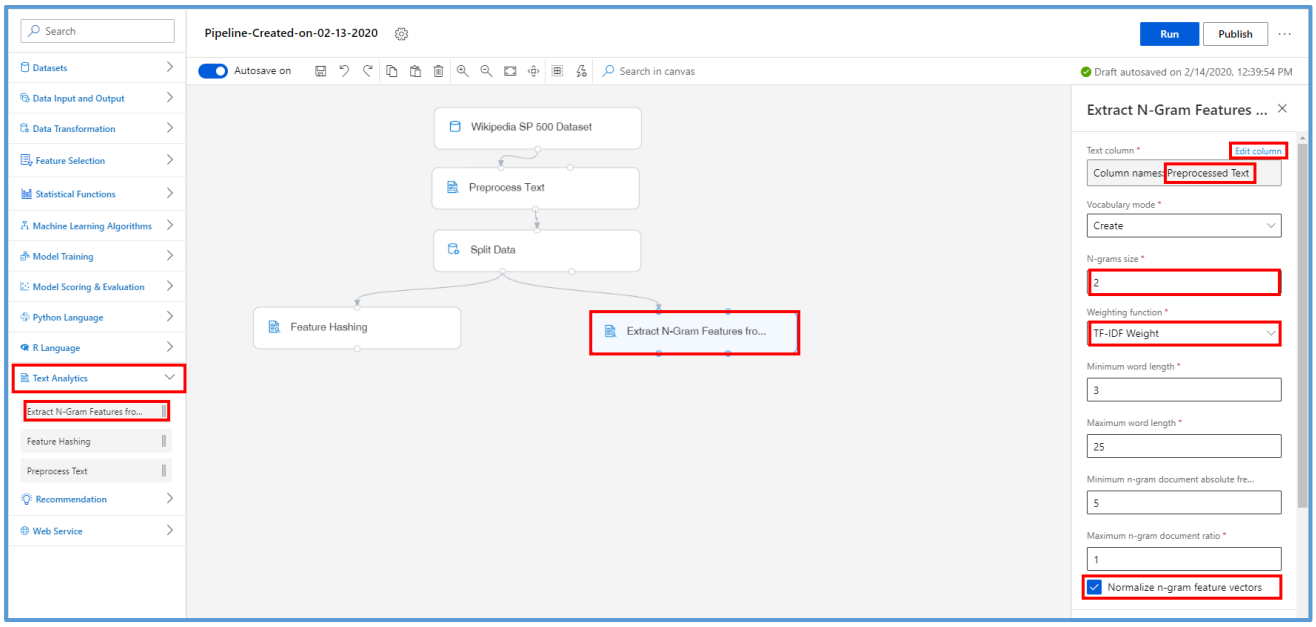
    4. Select the **Edit columns** link to open the Text column editor and then enter the **Preprocessed Text** column for the Target column field.

    5. Leave default selection of Vocabulary mode to **Create** to indicate that you're creating a new list of n-gram features.

    6. Set N-grams size : **2** (which is the maximum size of the n-grams to extract and store)

    7. Select Weighting function : **TF-IDF Weight**. (This function calculates a term frequency/inverse document frequency score and assigns it to each n-gram. The value for each n-gram is its TF score multiplied by its IDF score.)

8. Check the option to **Normalize n-gram feature vectors**. If this option is enabled, each n-gram feature vector is divided by its L2 norm.



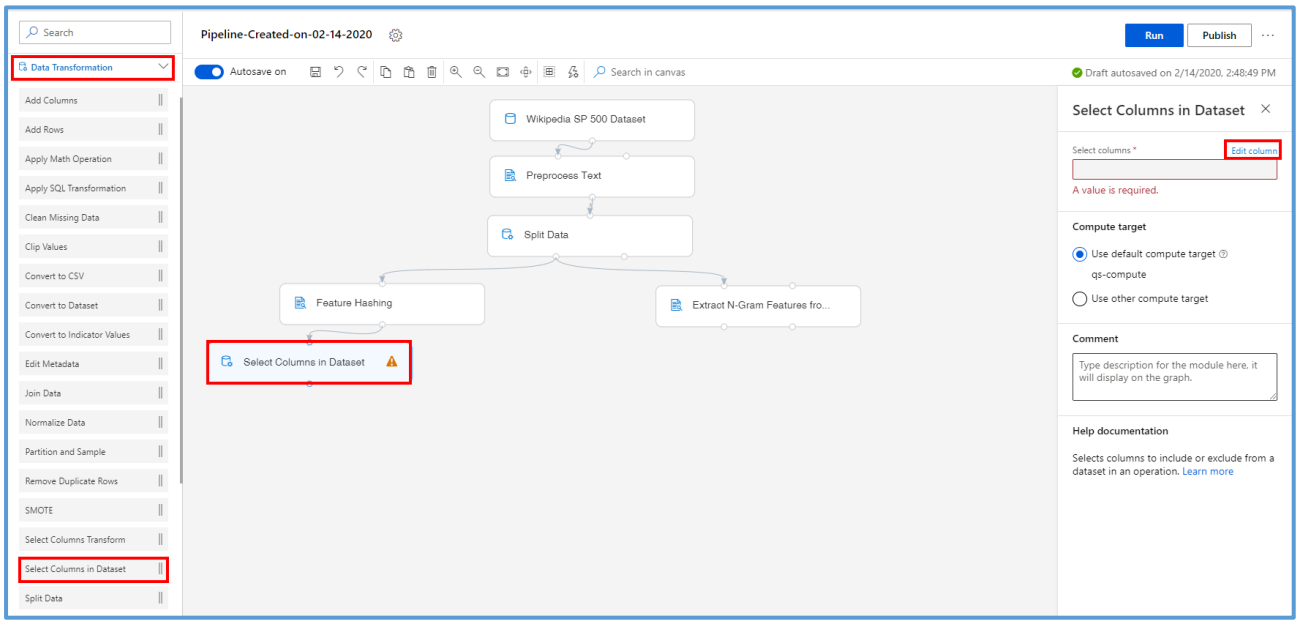
Task 8: Remove text columns from dataset

1. Select **Data Transformation** section in the left navigation. Follow the steps outlined below to add two Select Columns in Dataset modules on both featurization branches:

1. Select the **Select Columns in Dataset** prebuilt module

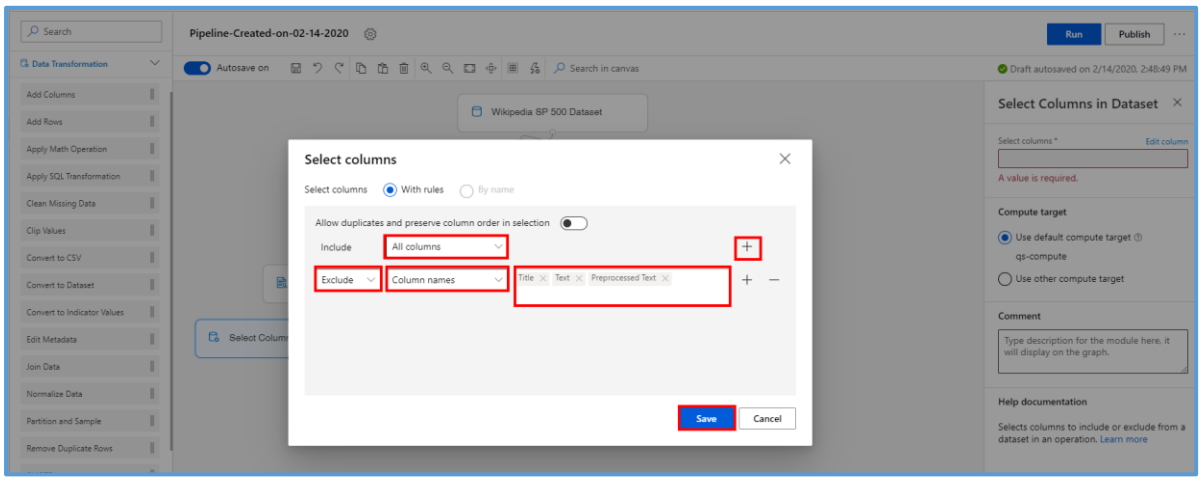
2. Drag and drop the selected module on to the canvas

3. Connect the input Select Columns in Dataset module to the Feature Hashing module output.

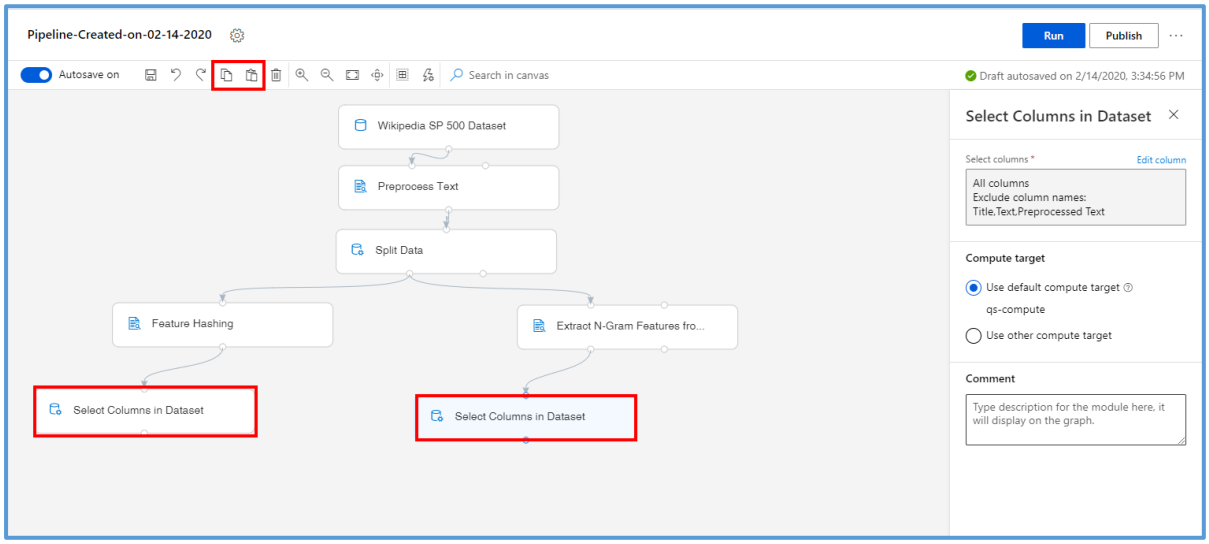


2. Select the **Edit columns** link to open the Select columns editor and then select **All columns**.

3. Click on the **+** icon to add another operation line and select the **Exclude, Column names** option. Enter to following list of columns to be excluded: **Title**, **Text**, **Preprocessed Text**. Select **Save** to close the editor.



4. Select the Select Columns in Dataset module and click on the **Copy** icon and then **Paste** it on the canvas. Connect the copied module to the Extract N-Gram Feature from Text module.



Task 9: Add the Train Model Modules

1. Select **Model Training** section in the left navigation. Follow the steps outlined below:

    1. Select the **Train Model** prebuilt module.

    2. Drag and drop the selected module on to the canvas

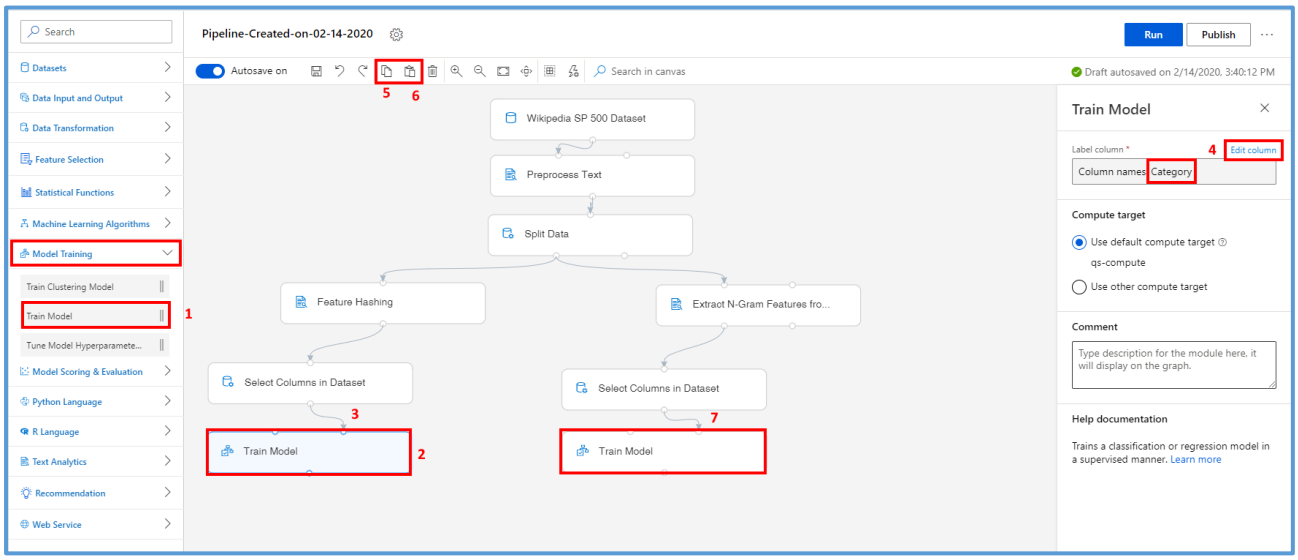
3. Connect the first output of the Select Columns in Dataset module on the left of the canvas, to the second input (the Dataset input) of the newly added Train Model module.

4. Select the **Edit columns** link to open the Label column editor and then enter the **Category** column. Select **Save** to close the editor.

5. Select the Train Model module and click on the **Copy** icon.

6. Click on the **Paste** icon to paste a second Train Model module on the canvas under the right branch of the pipeline tree.

7. Connect this last Train Model module, by linking at the second input (the Dataset input), the output of the right most Select Columns in Dataset module on the canvas.



Task 10: Initialize Multiclass Logistic Regression Model

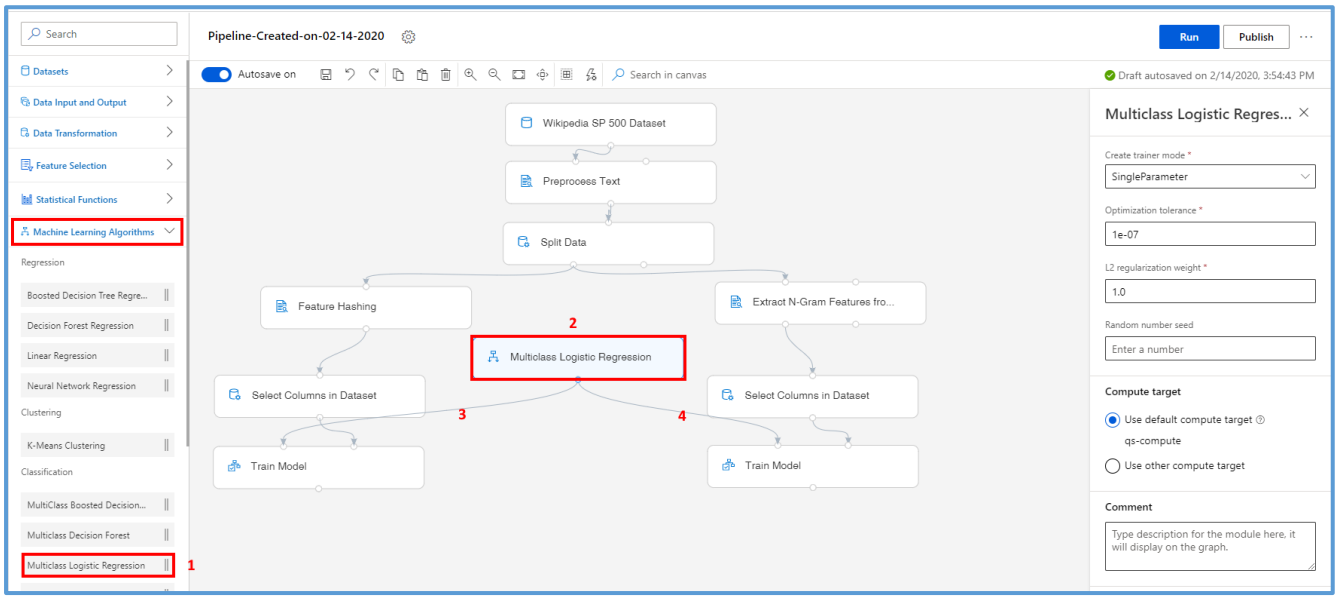
1. Select **Machine Learning Algorithms** section in the left navigation. Follow the steps outlined below:

    1. Select the **Multiclass Logistic Regression** prebuilt module, in the Classification category.

    2. Drag and drop the selected module on to the canvas

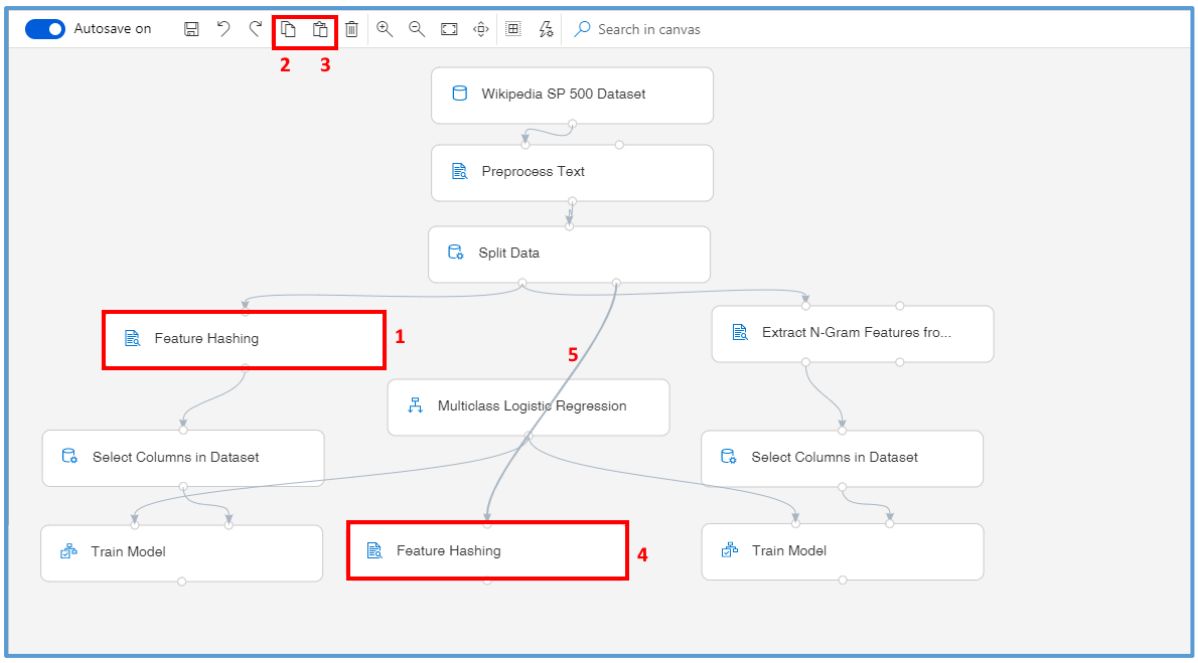
    3. Connect the output of the Multiclass Logistic Regression module to the first input of the left branch (the Feature Hashing approach) Train Model module.

4. Connect the output of the Multiclass Logistic Regression module to the first input of the right branch (the N-Gram Features approach) Train Model module.



Task 11: Convert the plain text of the articles to integers with Feature Hashing module, on the test set

1. Select the existing **Feature Hashing** module.
2. Copy the **Feature Hashing** module using the **Copy** option from the top menu.
3. **Paste** the copied module on the canvas.
4. Position the new **Feature Hashing** module between the two Train Model modules on the canvas.
5. Connect the second output of the Split Data, the test set output, to the input of the newly added Feature Hashing module.



Task 12: Featurize unstructured text data with Extract N-Gram Feature from Text module, on the test set

1. Select the existing **Extract N-Gram Feature from Text** module.

2. Copy the **Extract N-Gram Feature from Text** module using the **Copy** option from the top menu.

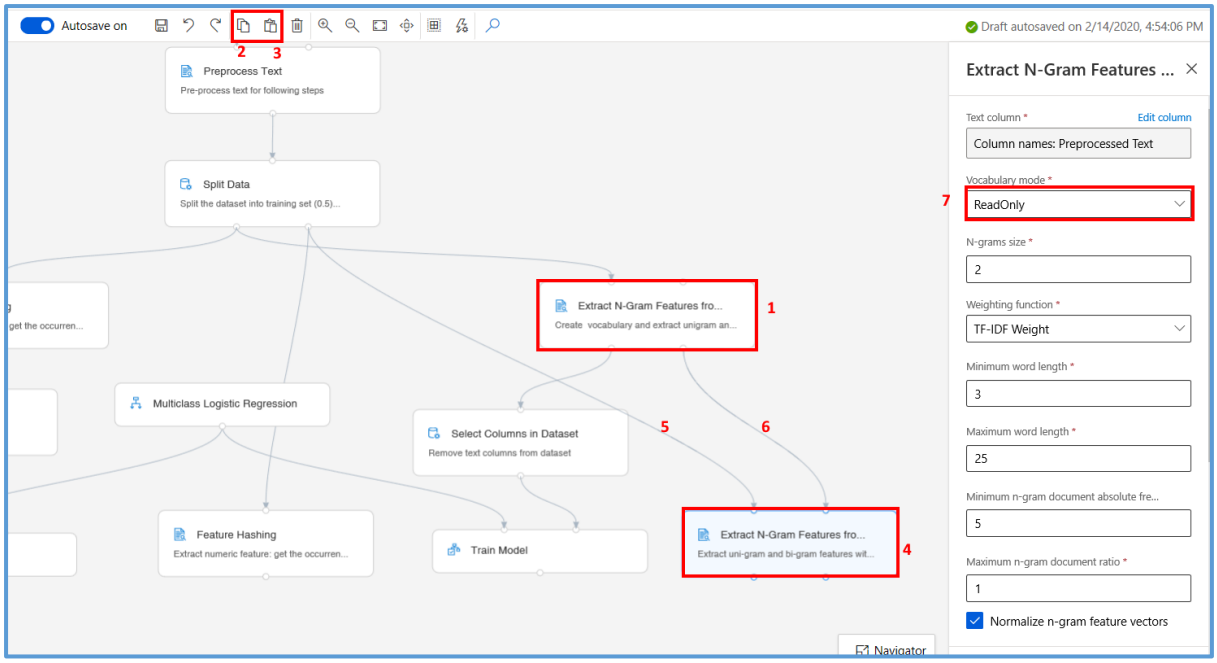
3. **Paste** the copied module on the canvas.

4. Position the new **Extract N-Gram Feature from Text** module near the rightmost Train Model module on the canvas.

5. Connect the second output of the Split Data, the test set output, to the first input of the newly added Extract N-Gram Feature from Text module.

6. Connect the second output of the uppermost Extract N-Gram Feature from Text, to the second input of the copied Extract N-Gram Feature from Text module.

7. Select the newly added module and in the right settings pane, set the Vocabulary mode to **ReadOnly**



Task 13: Setup Score Model Modules

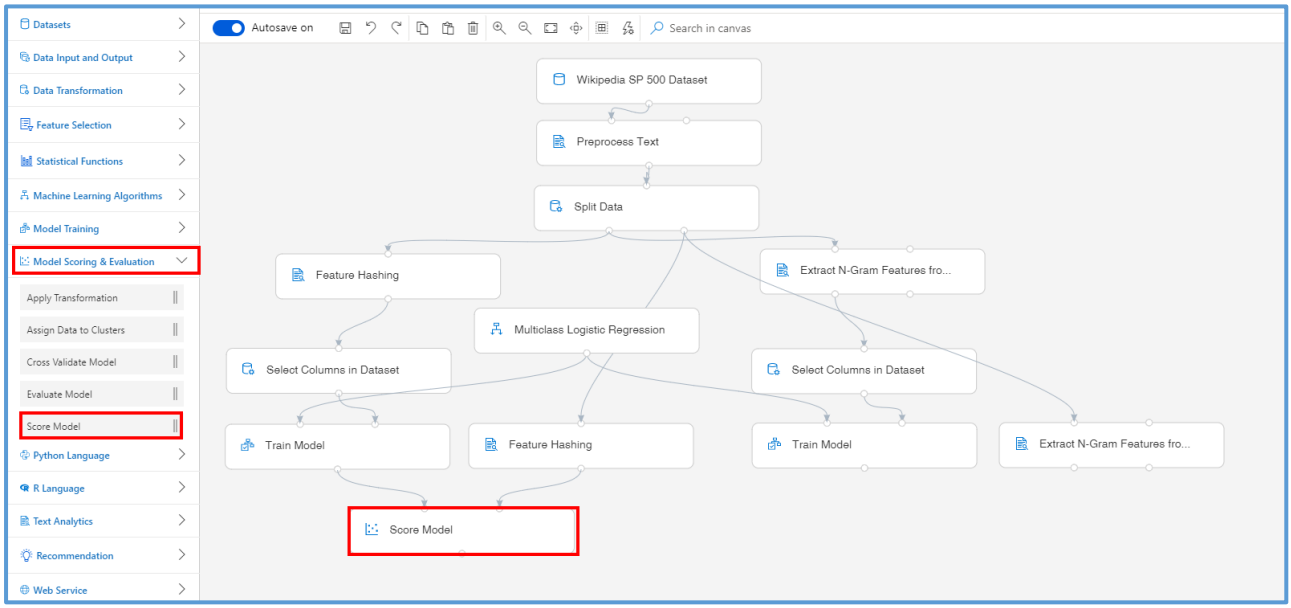
1. Select **Model Scoring & Evaluation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Score Model** prebuilt module

    2. Drag and drop the selected module on to the canvas

    3. Connect the leftmost Train Model module to the first input of the Score Model module

    4. Connect the output of the Feature Hashing module (the lower one, on the test set branch) to the second input of the Score Model module



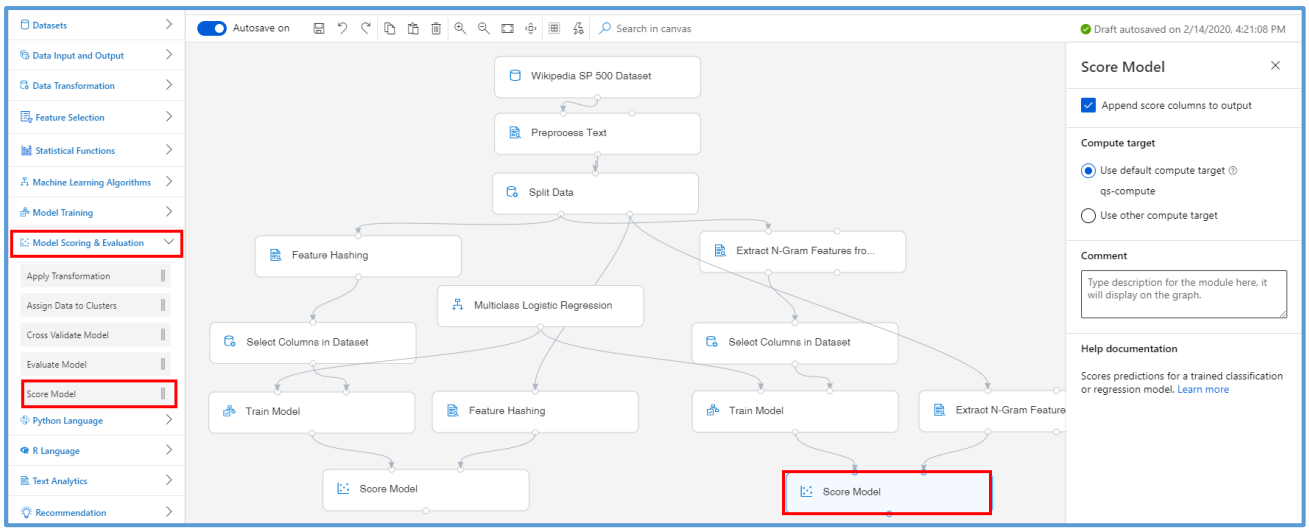
2. Select **Model Scoring & Evaluation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Score Model** prebuilt module

    2. Drag and drop the selected module on to the canvas

    3. Connect the rightmost `Train Model` module to the first input of the `Score Model` module

    4. Connect the first output of the `Extract N-Gram Feature from Text` module (the lower one, on the test set branch) to the second input of the rightmost `Score Model` module.



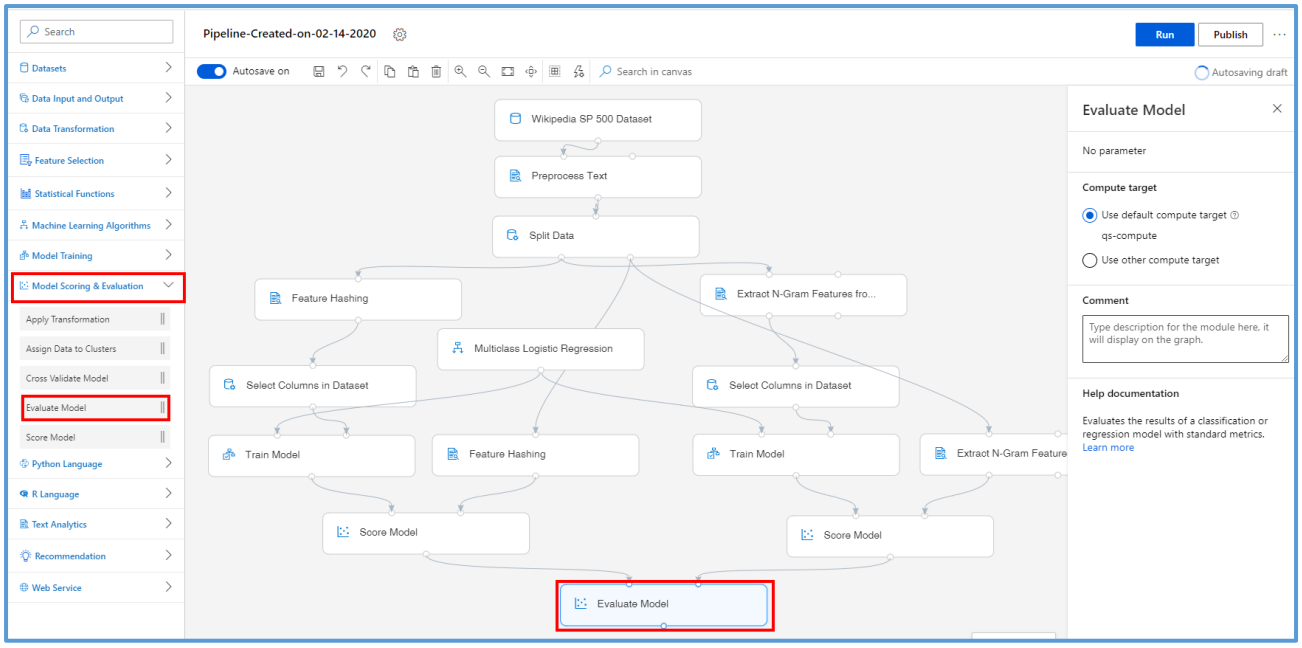
Task 14: Setup Evaluate Model Module

1. Select **Model Scoring & Evaluation** section in the left navigation. Follow the steps outlined below:

    1. Select the **Evaluate Model** prebuilt module

    2. Drag and drop the selected module on to the canvas

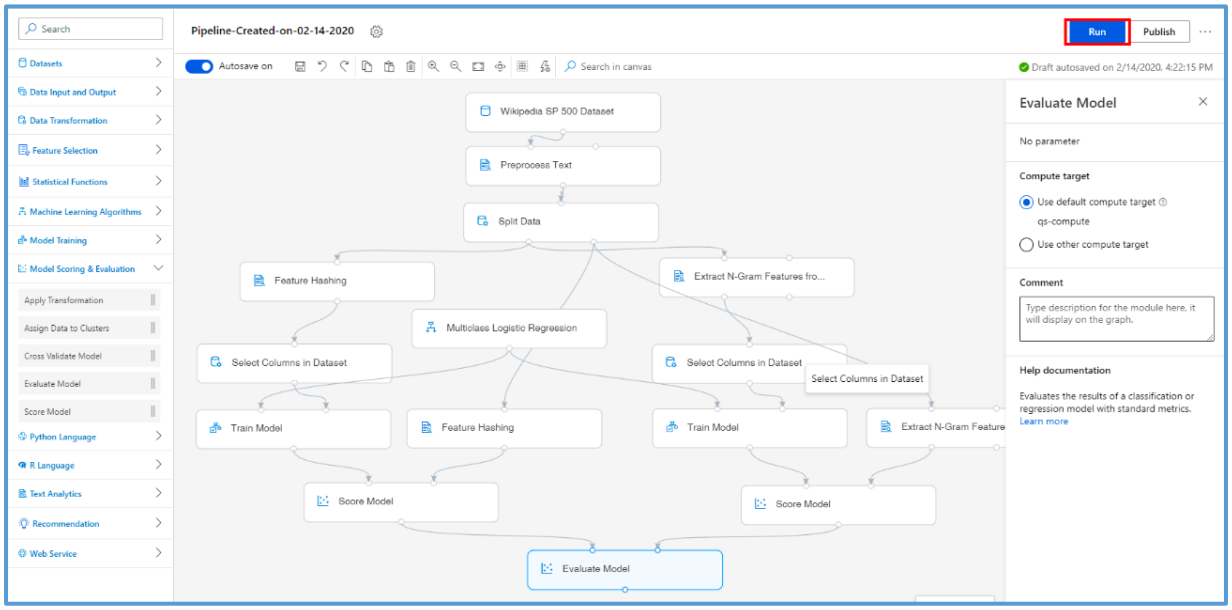
    3. Connect the two `Score Model` modules to the inputs of the `Evaluate Model` module



Exercise 2: Submit Training Pipeline

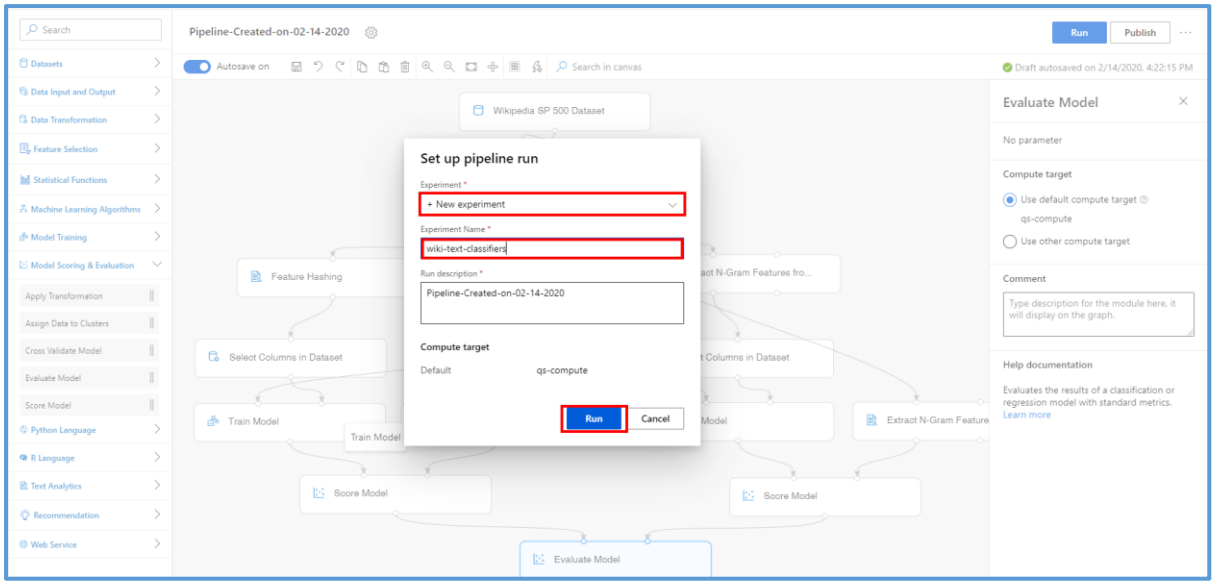
Task 1: Create Experiment and Submit Pipeline

1. Select **Submit** to open the Setup pipeline run editor.



Please note that the button name in the UI is changed from **Run** to **Submit**.

2. In the Setup pipeline run editor, select **Experiment, Create new** and provide New experiment name: **wiki-text-classifier**, and then select **Submit**.



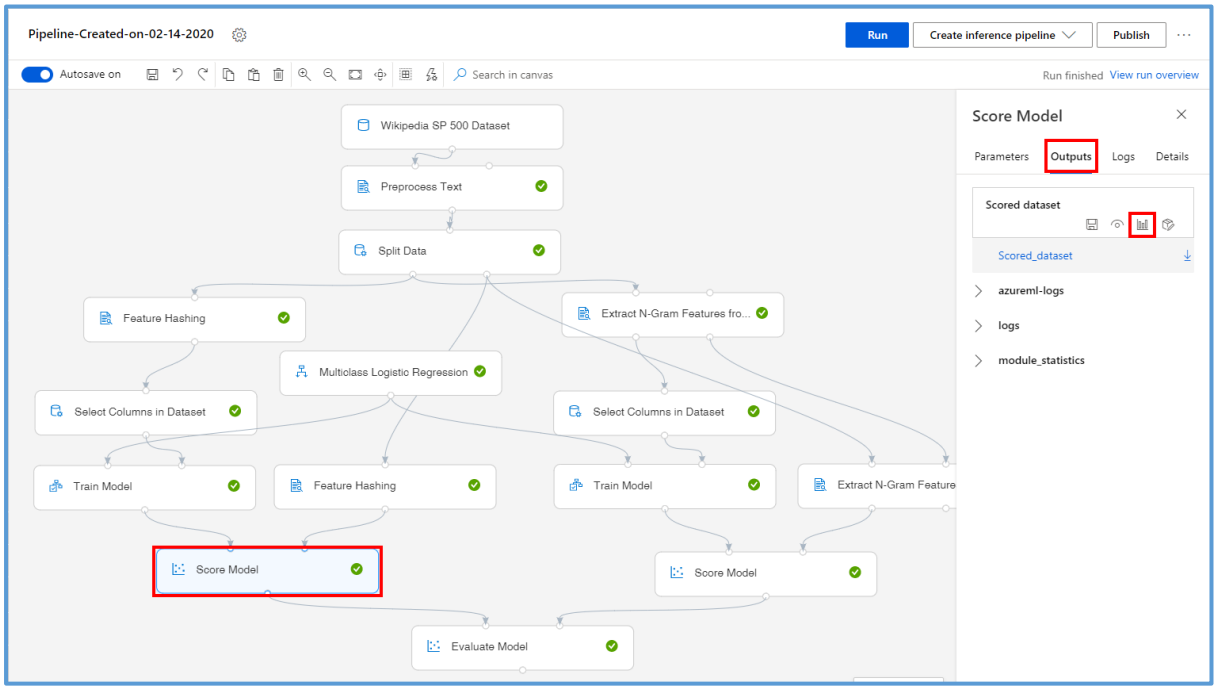
3. Wait for pipeline run to complete. It will take around **20 minutes** to complete the run.

4. While you wait for the model training to complete, you can learn more about the classification algorithm used in this lab by selecting Multiclass Logistic Regression module.

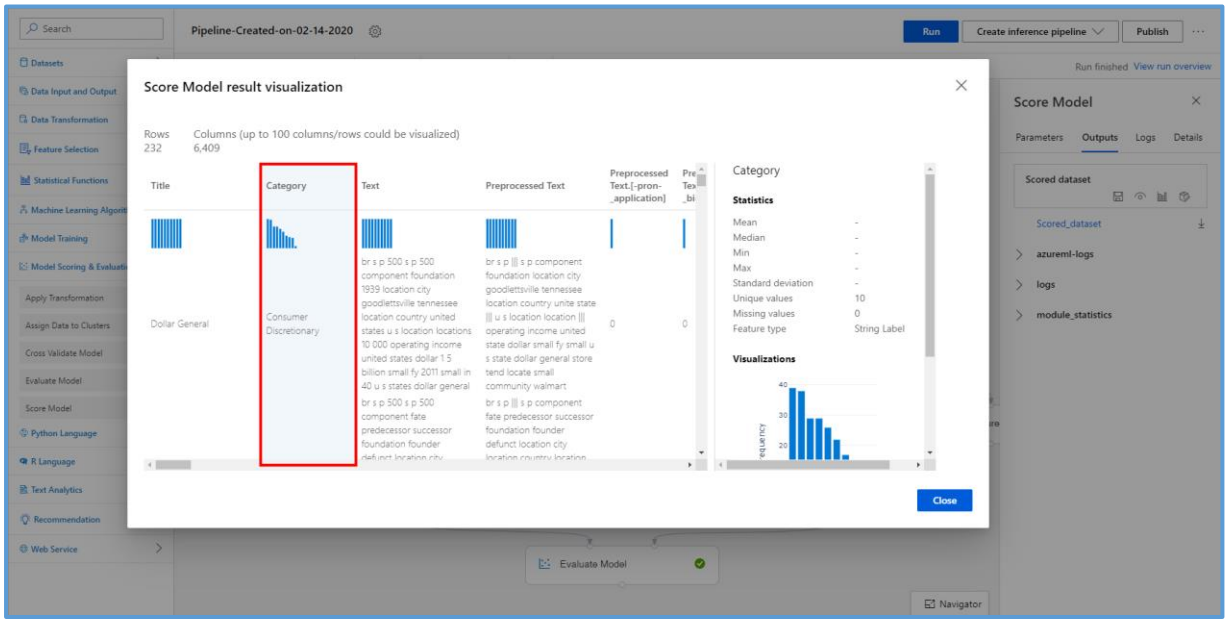
Exercise 3: Visualize Training Results

Task 1: Visualize the Model Predictions

1. Select **Score Model, Outputs, Visualize** to open the Score Model result visualization dialog or just simply right-click the Score Model module and select **Visualize Scored Dataset**.

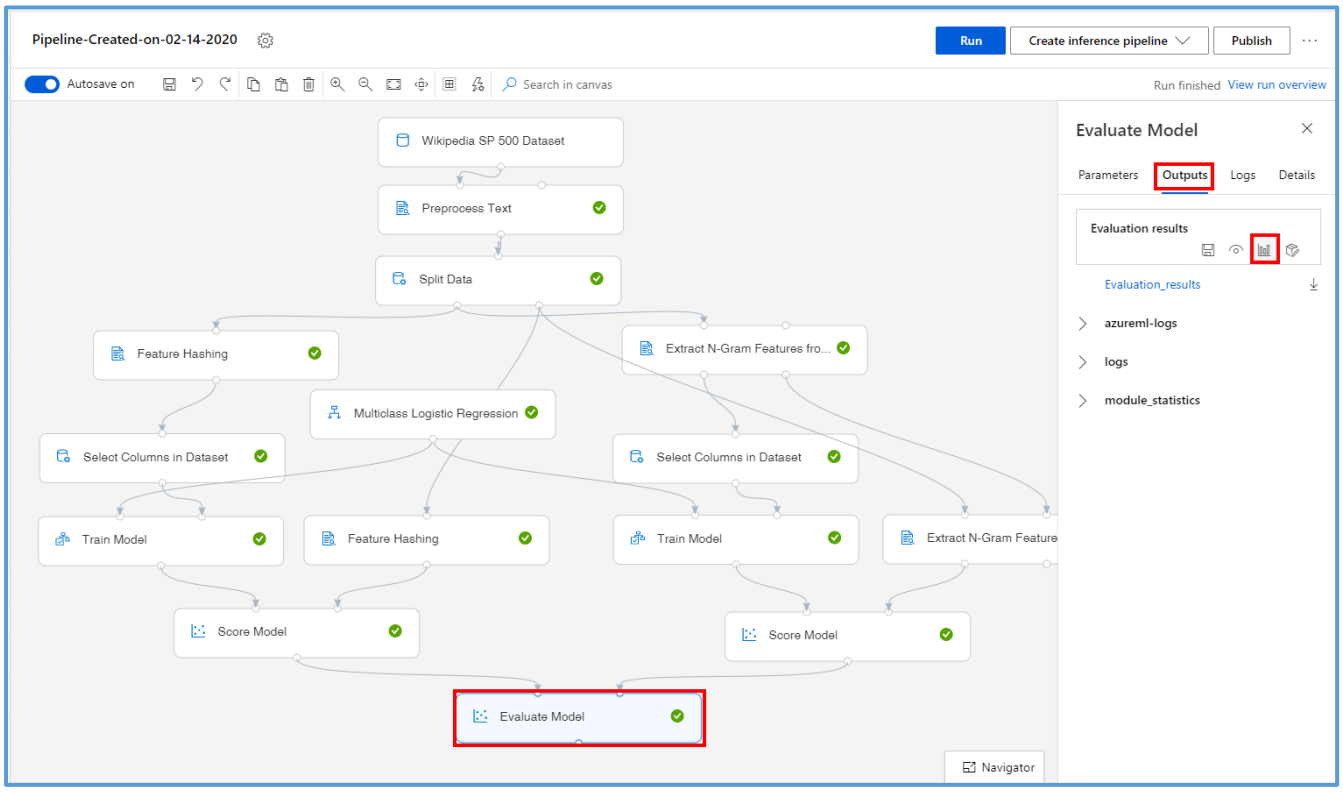


2. Observe the predicted values under the column **Category**.

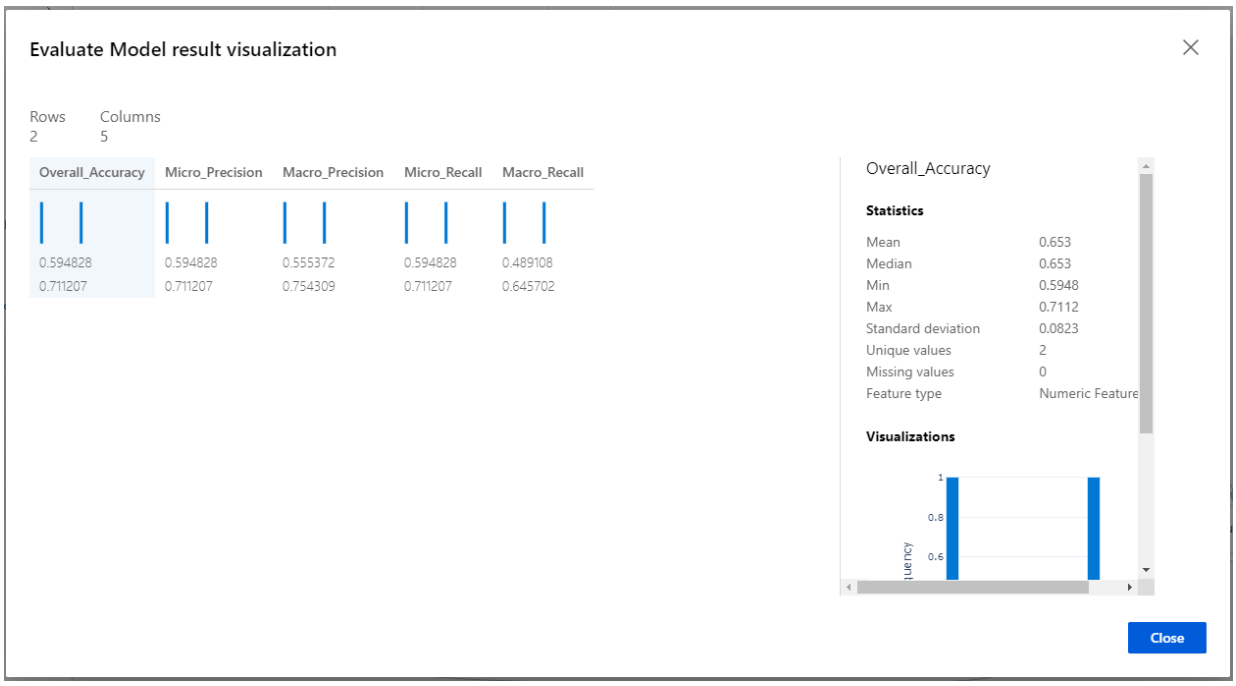


Task 2: Visualize the Evaluation Results

1. Select **Evaluate Model, Outputs, Visualize** to open the Evaluate Model result visualization dialog or just simply right-click the Evaluate Model module and select **Visualize Evaluation Results**.



2. Evaluate the model performance by reviewing the various evaluation metrics. Evaluate Model has two input ports, so that we could evaluate and compare scored datasets that are generated with different methods. In this sample, we compare the performance of the result generated with feature hashing method and n-gram method.



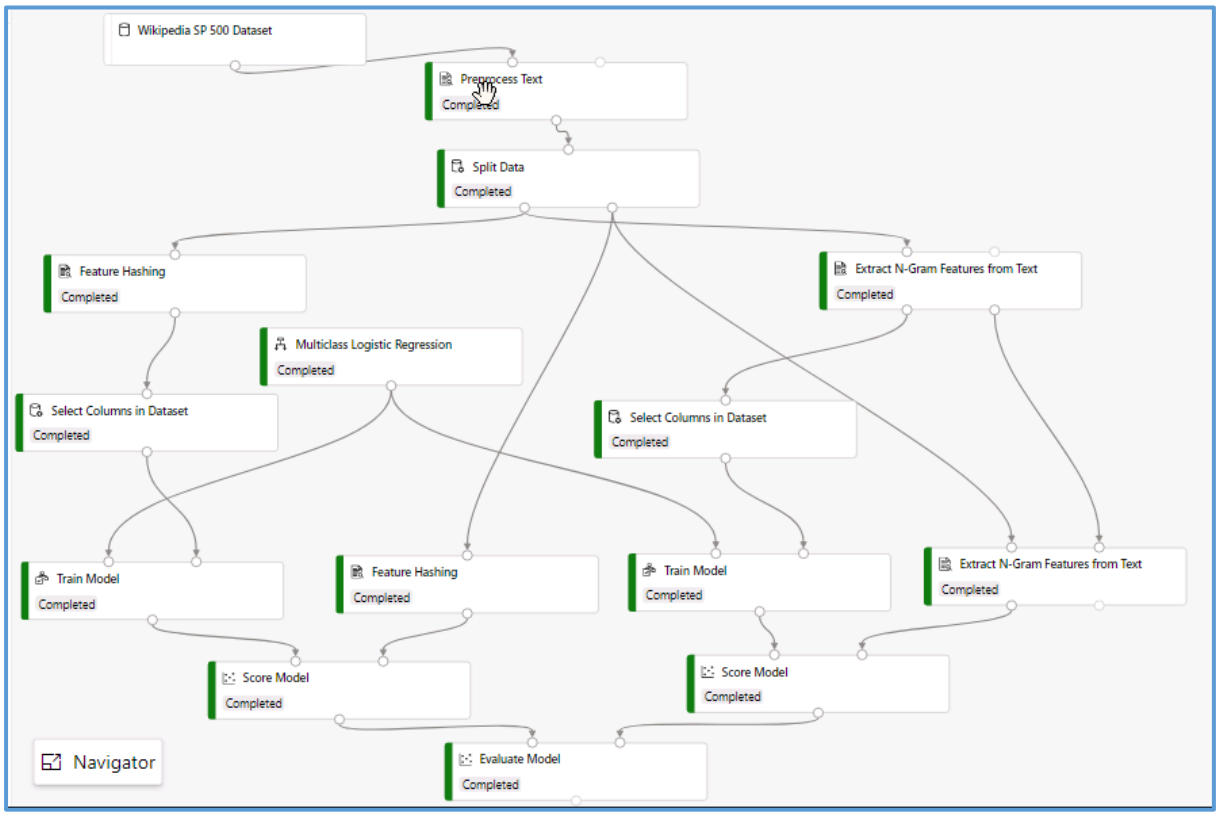
Next Steps

Congratulations! You have trained a simple text classifier and compared performance of the result generated with two different featurization modules. You can continue to experiment in the environment but are free to close the lab environment tab and return to the Udacity portal to continue with the lesson.

Chapter 17: Walkthrough: Train a Simple Text Classifier

- While setting up the pipeline, setup the default compute target.
- Choose the sample dataset [Wikipedia SP 500] for the training process
- First, we used the Preprocess text task, which was, configured to use English language & we have selected the “Text” column for cleaning process. Various pre-processing tasks were, chosen.
- Split data task was, used with “Split Rows” as the splitting task & 50% to be, considered in the first output dataset. [50% for Training and 50% for Test]
- “Feature Hashing” task was, used on Training data to convert the plain text of the article into integer value. “Preprocess text” column was the target column, which was, derived from the Preprocess Text task.
- Training output of the data has also been, used to extract N-gram features from the text, with “Vocabulary mode” being “Create” as one of the important setting. This step is mainly to create the vocabulary.
- We have then specified the names of the columns which we would like to use further, for this we have used “Select columns in Dataset” for both “Feature Hashing” and “Extract N-gram features from text”

- This concludes the data configuration task & now we would start the Data Training part.
- Multiclass logistic Regression algorithm was, used.
- Train the model on the data output from “Feature hashing” and also on the output from “Extract N-gram features from text”
- For testing purpose in one section, we connected the Test data output from Split Data to the input of New “Feature Hashing” task & in the other section, the Test data output from Split Data is, connected to “Extract N-gram features from text” & here Vocabulary mode has been, selected as “Read only”. Hence the second output from “Extract N-gram features from text” Is connected to the second input of “Extract N-gram features from text” in order to maintain consistency.
- Then we used “Score Model” task on both the sections. In one section, Score model is, connected with the Output from Train Model & Feature Hashing task. In the other section, Score model is connected to the output from Train model and “Extract N-gram features from text”
- Finally the output from Score model in both sections have added to the Evaluate model task

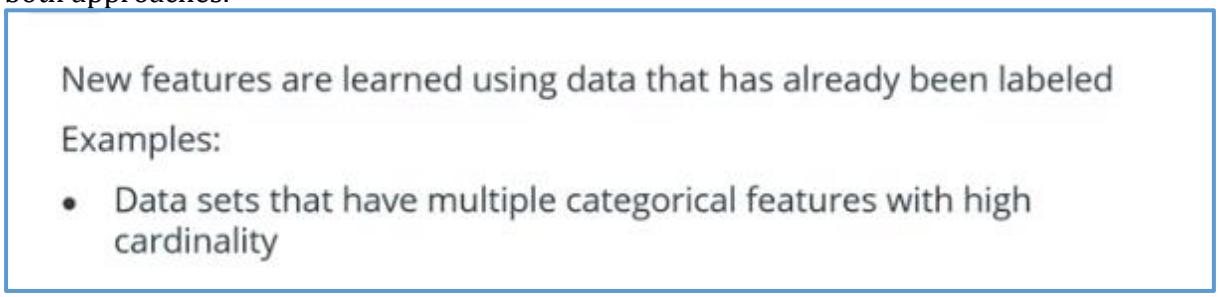


Chapter 18: Feature Learning

As we have discussed previously, *feature engineering* is one of the core techniques that can be, used to increase the chances of success in solving machine-learning problems. As a part of feature engineering, **feature learning** (also called **representation learning**) is a technique that you can use to derive new features in your dataset. Let us have a look at how this technique works.

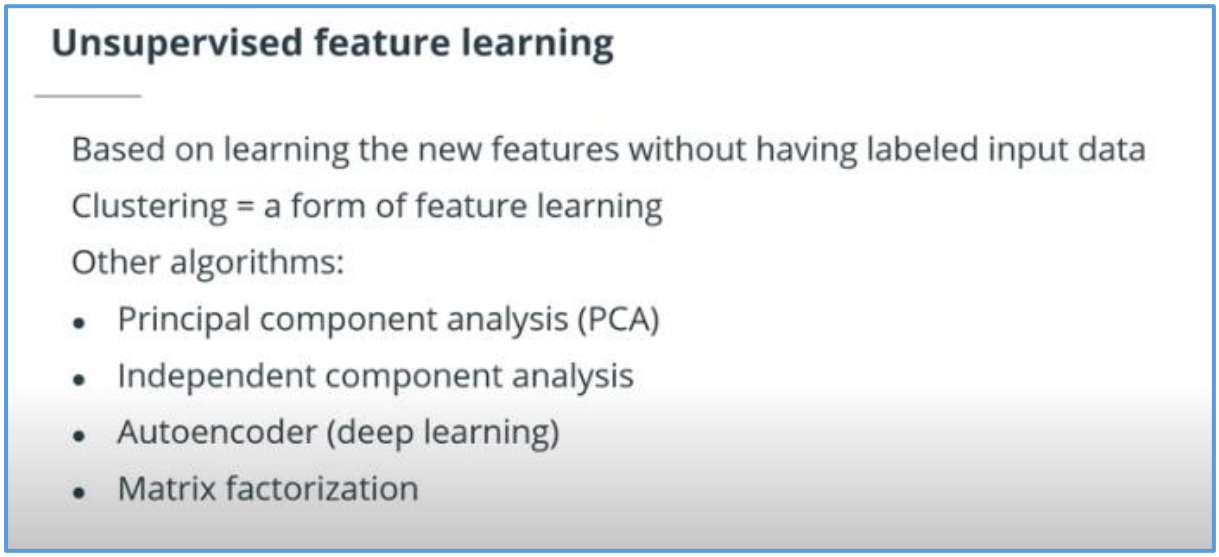
Supervised and Unsupervised Approaches

Earlier in this lesson, we pointed out that *feature learning* is one of the machine learning techniques that can be, done in both *supervised* and *unsupervised* ways. Let us have a look at both approaches.



One-hot encoding is a very different approach. In, one-hot encoding, we transform each categorical value into a column. If there are n categorical values, n new columns are, added. For example, the Colour property has three categorical values: Red, Green, and Blue, so three new columns Red, Green, and Blue are, added. If an item belongs to a category, the column representing that category gets the value 1, and all other columns get the value 0.

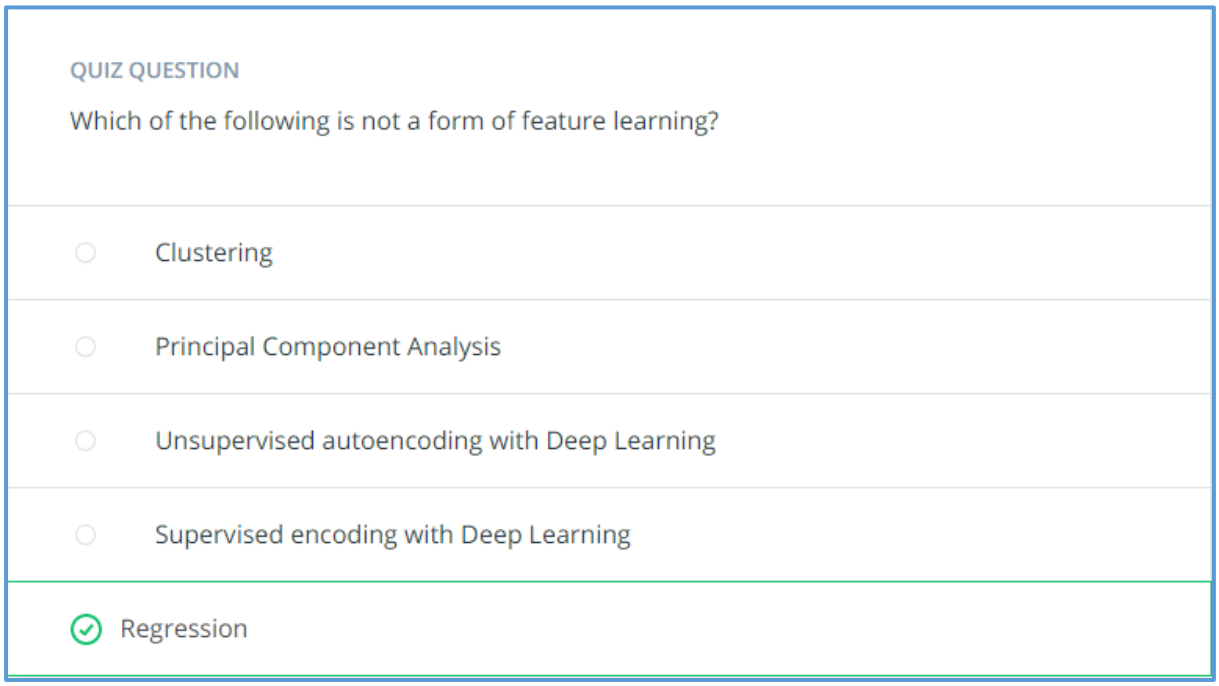
One drawback of, one-hot encoding is that it can potentially generate a very large number of columns.



In case of unsupervised learning, we do not need existing labels on input data; however, we can learn new features through some of these algorithms & then use those new features as inputs for our ML training process.

Remember that feature (or representation) learning is, used to transform sets of inputs into other inputs that are potentially more useful in solving a given problem. Regression is a supervised learning prediction problem, hence it is not, considered to be, a form of

feature learning.



Chapter 19: Applications of Feature Learning

As we just mentioned, some prominent applications of machine learning include *image classification* and *image search*. On the remainder of this page, we will take a closer look at some specific examples of these two approaches.

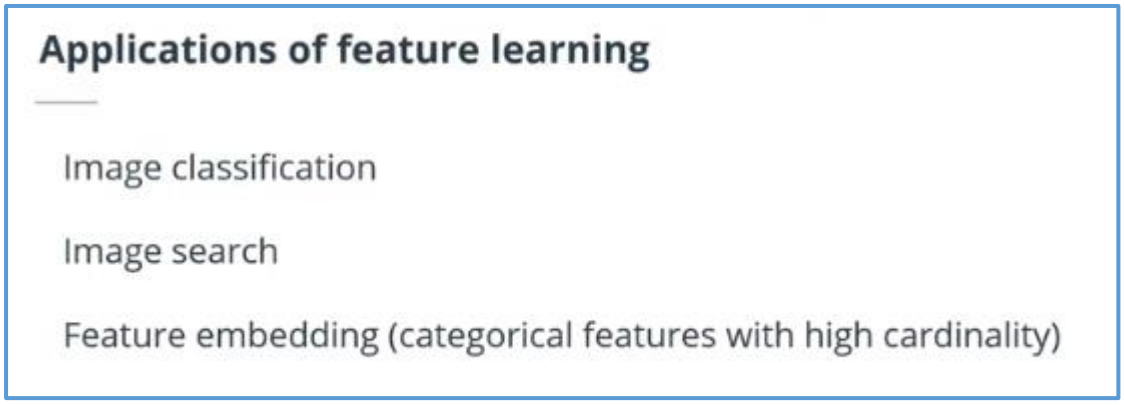


Image Classification with Convolutional Neural Networks (CNNs)

CNN is a special case of deep learning neural network that has a combination of hidden layers, which has very specific abilities in learning hidden features.

Important fact about image classification with CNN is that we can use the convolutional layers to learn some local patterns. We start to learn a bunch of local patterns. They are translation invariant which means once the model recognizes a certain type of local pattern somewhere in the image it can recognize it somewhere else in the image.

In case of densely connected Neural networks it learns global patterns, thus if it learns to apply the pattern at a different place it requires to be retrained.

CNN's have the capabilities of learning hierarchies of patterns.

Next is the Max –pooling layer which down samples the data & makes it translational invariant.

Finally, we densely connect all the outputs and learn the classification.

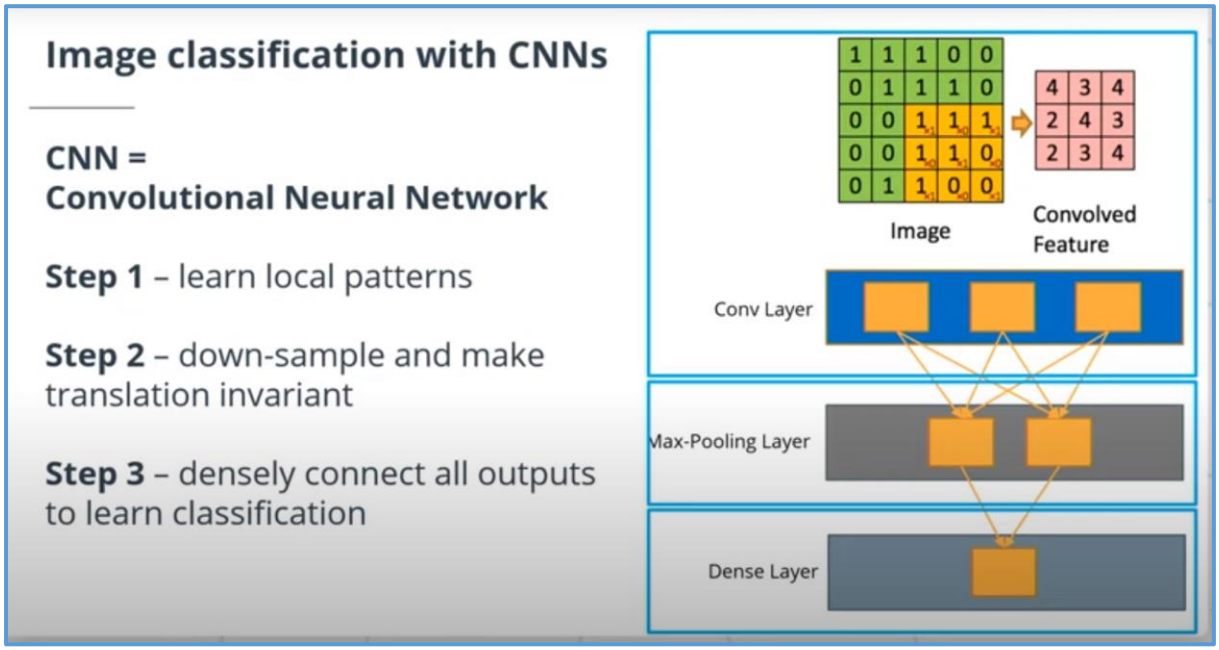


Image Search with Autoencoders

Autoencoder is a special type of neural network, which is, used for unsupervised learning.

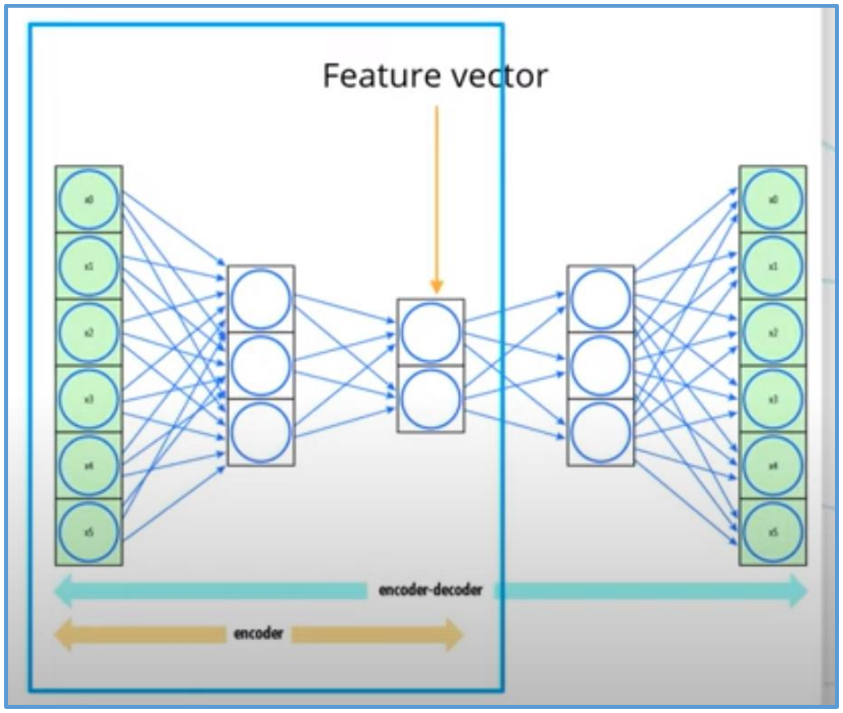
It is, trained to reproduce its own inputs as accurately as possible.

It typically starts with very large number of inputs & then each layer is progressively narrower than the previous one until the minimum layer width is, reached. Post which layers start to have increasingly larger width until we reach the output, which has the same number of outputs, as there are inputs in the first layer.

The middle layer has the smallest width & it is important for two reasons:

1. **It marks a border between the two internal components of the autoencoder which are the encoder left of the particular layer and the decoder which is right of the particular layer.**
2. **That layer produces the feature vector, which Is a compressed representation of the input. [Compressed means dimensionally reduced as opposed to reduce in size]. Autoencoders are not efficient in compressing the images, rather they are efficient in translating those images into n-dimensional vectors, where n is the width of the middle layer. Finally, those vectors can be, compared with each other using distance metric like Euclidean distance.**

Once the autoencoder is, trained then we can use the left part, which is the encoder, which is a trained DL ML model, which can be, used to encode inputs. Finally, we can apply that distance measure to find the input images.



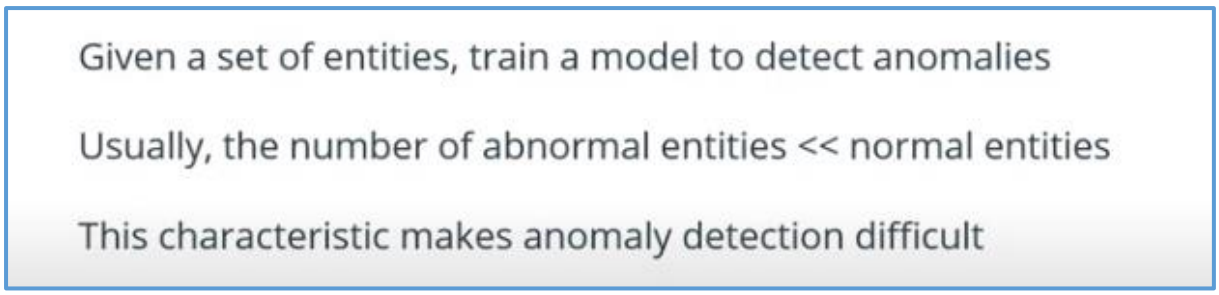
We have a set of images that contains atoms. We move those images as part of training through the autoencoder. Finally, we get an encoder, which is capable of encoding images, which has atoms in them.

Now when we have a new image we feed through the encoder to get the feature vector. Then we calculate the distance between that feature vector and all other feature vector used during the training & if the distance is below the threshold then there is high probability that the new image has image or multiple images of atoms.

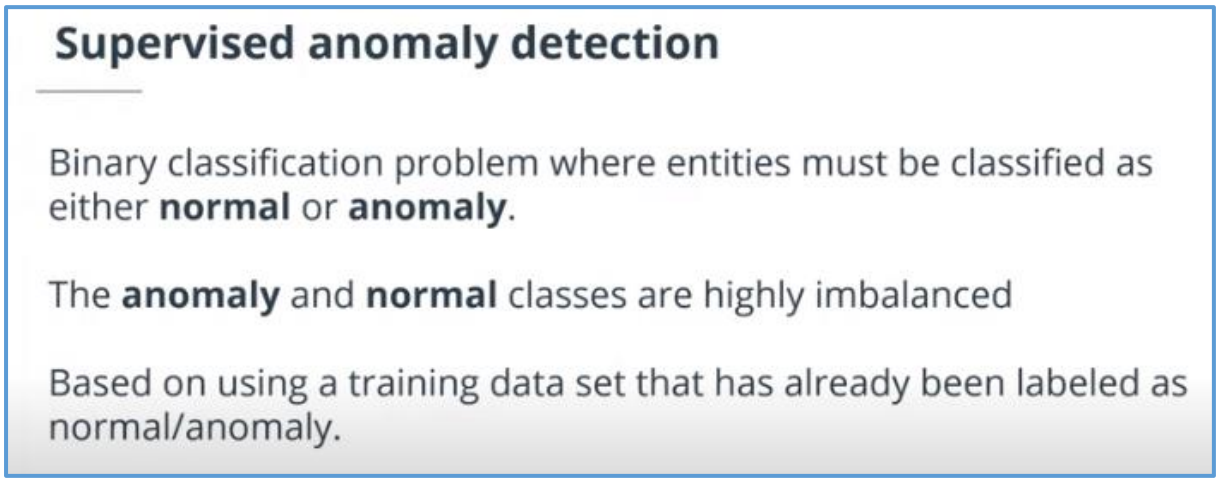
That is how a simple image searcher using autoencoder style DL ML models can actually be, implemented.

Chapter 20: Anomaly Detection

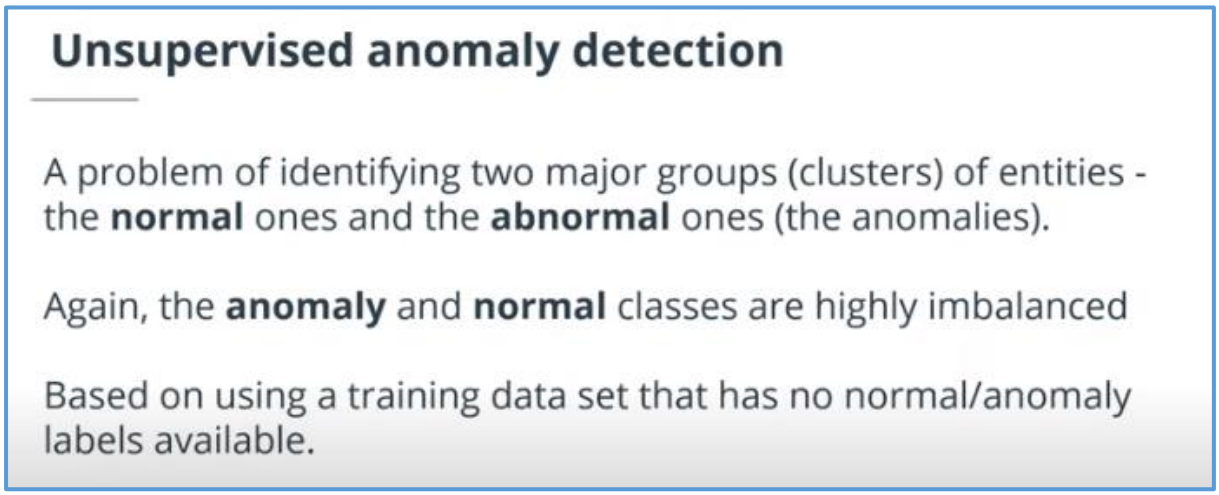
Datasets often contain a small number of items that deviate significantly from the norm. These *anomalies* can be of interest, since they may be the result of bad data, unusual behavior, or important exceptions to the typical trends. **Anomaly detection** is a machine learning technique concerned with finding these data points. It is a Classification problem.



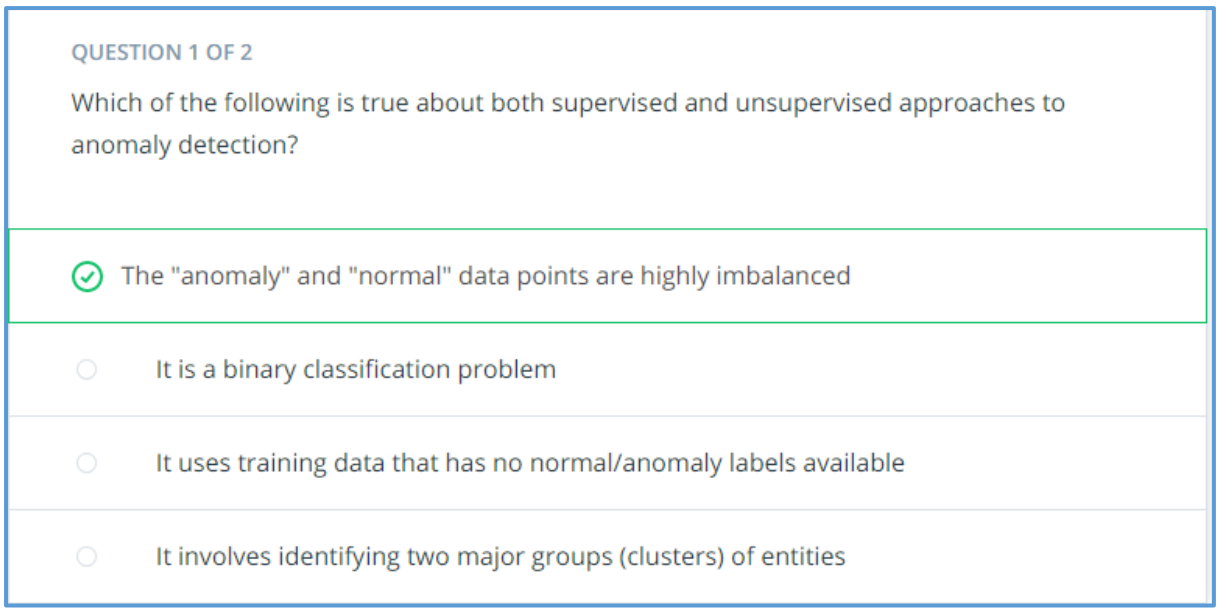
There is severe imbalance between the two classes, which makes anomaly detection very difficult. Thus, some of the classical ML approaches that are, used for binary classification cannot deliver the desired results.



We need a dataset with transactions that has already been, proven to be, fraudulent at earlier points of time.



Thus, we need to create a model that is capable of defining what either normal or abnormal means & then detect the patterns in new items.

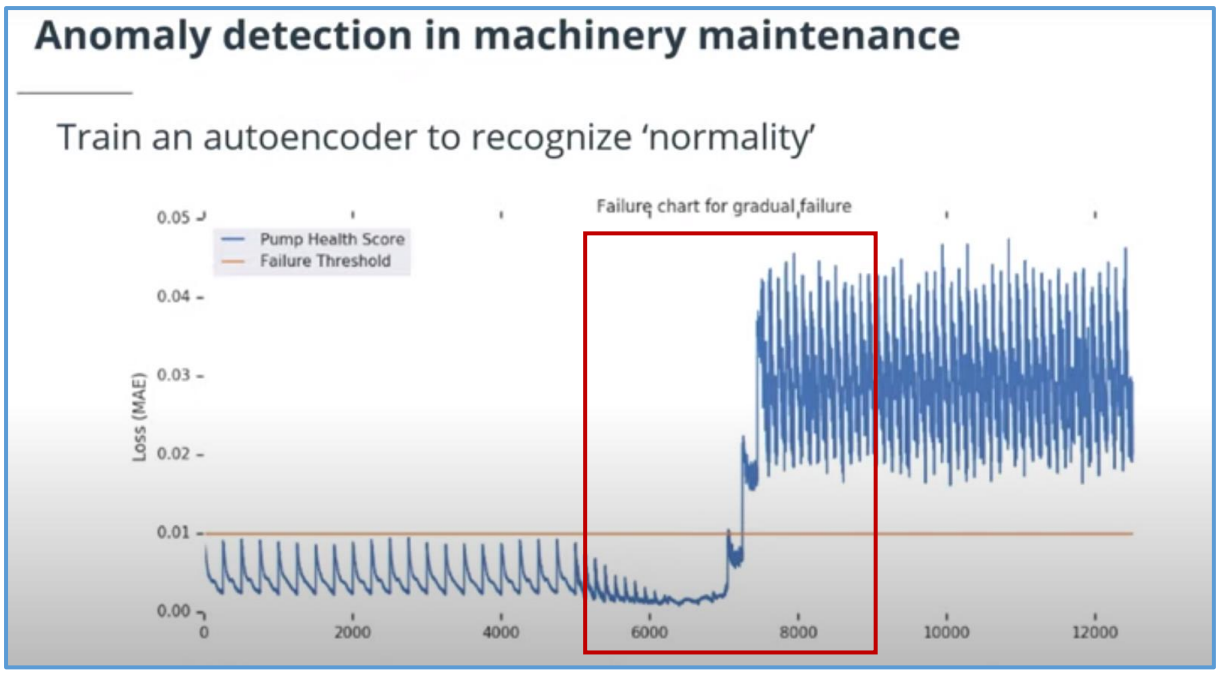


Applications of Anomaly Detection

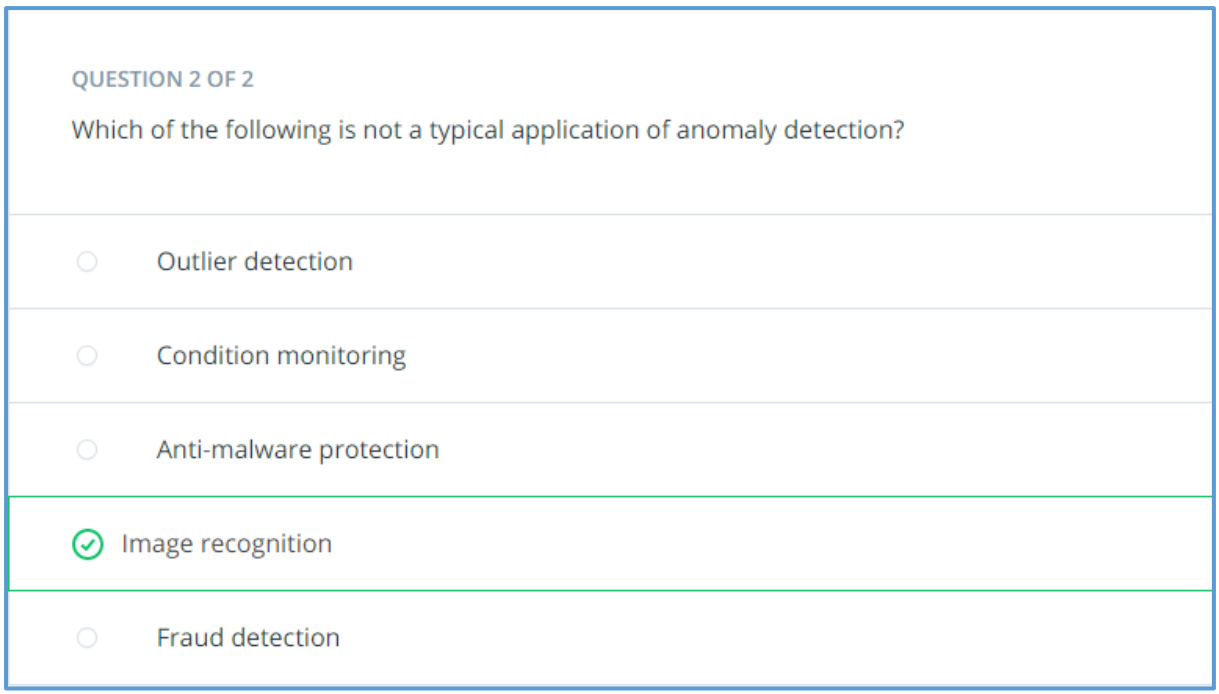
- **Monitoring of the condition for various components of machinery used in industries. Typical pattern is in Industrial maintenance.**
- **Fraud detection**
- **Intrusion detection – Pattern of network communication that is specific to a vector of attack that is classified as an intrusion is an abnormal type of activity in network**
- **Anti – Virus and Anti – Malware protection**
- **Data preparation – Outlier detection**

Anomaly detection in machinery maintenance –

- **Train a specific class of deep learning model, an autoencoder and we will train that autoencoder to recognize normality and then obviously, if the autoencoder is trained to recognize normality, once we feed an entity that falls under the abnormal order the failure class then that particular ML model will be able to identify it.**



- **We receive inputs from various sensors, which we then pass through the autoencoder. That task of the autoencoder is to produce as output values that are as close to its inputs.**
- **Once we have set of readings that corresponds to normal set of operations, we use to train the autoencoder.**
- **Post training we apply the autoencoder to real stream of data, and then if failure occurs the autoencoder would be able to identify the failures.**
- **Threshold value is one of the metric used while training the autoencoder (Mean average error).**
- **When we train autoencoder on normal data which is coming under normal conditions of operations, the autoencoder would have certain level of this metric.**
- **We take the highest value of the metric, which is then set as the threshold of your normality.**
- **Then we move the new data through autoencoder & we calculate the value of MAE & if that particular value is above the threshold, which indicates, that we are in the region of failure.**
- **This helps in identifying the part where the system is failing but not completely failed, this in turn helps in taking early actions.**



Chapter 21: Prelaunch Lab

Prelaunch your lab environment.

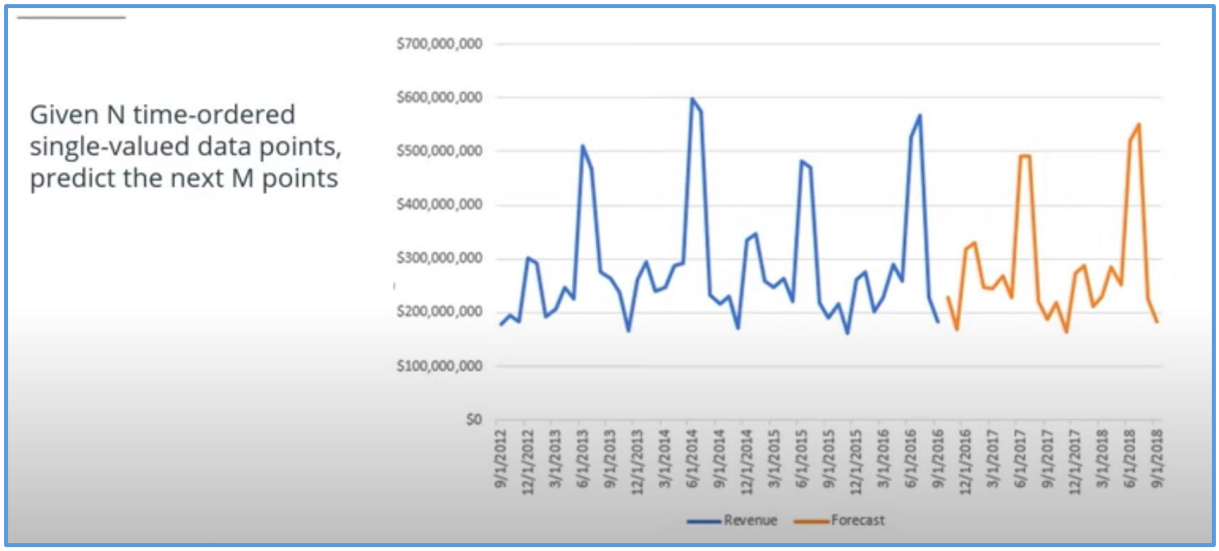
NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 22: Forecasting

Now let us have a look at the problem of **forecasting**. A typical example of a forecasting problem would be given a set of ordered data points (such as sales data over a series of dates), we want to predict the next data points in the series (such as what sales will look like next week).

Remember, forecasting is a class of problems that deals with predictions in the context of orderable datasets. These orderable datasets can be time-series datasets, but they do not have to be—forecasting can be, applied to other types of orderable sets as well.



Types of Forecasting Algorithms

ARIMA – AutoRegressive Integrated Moving Average

ARIMA is an evolution of ARMA, which was initially, designed to provide some kind of description for a random time series based processing.

Multi-variate regression

Forecasting can also be, modelled as a Regression problem where the task is to predict a numerical value for the next iteration of time series data.

We can take into account other time series based parameters which we can use to improve the accuracy of the model.

Prophet

Prophet was, developed by Facebook. This works best with Time series that has strong seasonal effect.

ForecastTCN

TCN stands for Temporal Convolutional Network. ForecastTCN was developed by, Microsoft based on the deep learning approach of TCN. It is a One-dimensional Convolutional network, since the approach is, based on time series.

RNNs (Recurrent Neural Networks)

Refers to class of network that has connections with additional nodes. Classical Neural network is often called feed forward neural network where everything flows in one direction. [Input to Output]. Now if we add backward facing connections then we create

cycles within the network. This gives rise to RNN's. They can effectively learn time based patterns.

Chapter 23: Lab Forecasting

Train a time-series forecasting model using Automated Machine Learning
Lab Overview

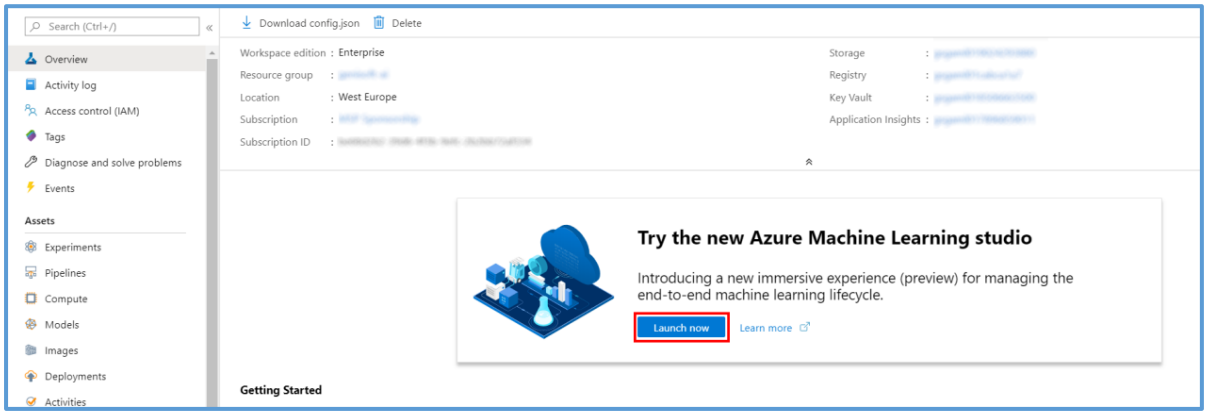
In this lab you will learn how the Automated Machine Learning capability in Azure Machine Learning (AML) can be used for the life cycle management of the manufactured vehicles and how AML helps in creation of better vehicle maintenance plans. To accomplish this, you will train a Linear Regression model to predict the number of days until battery failure using Automated Machine Learning available in AML studio.

Exercise 1: Creating a model using automated machine learning

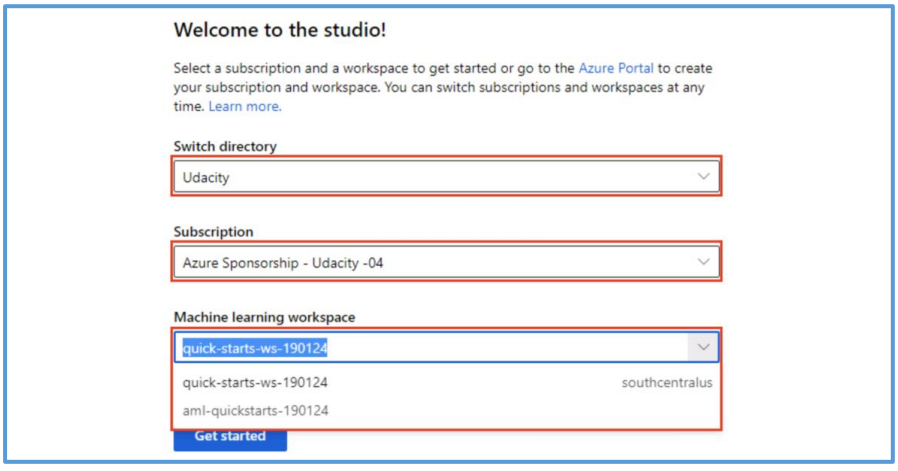
Task 1: Create an automated machine learning experiment using the Portal

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

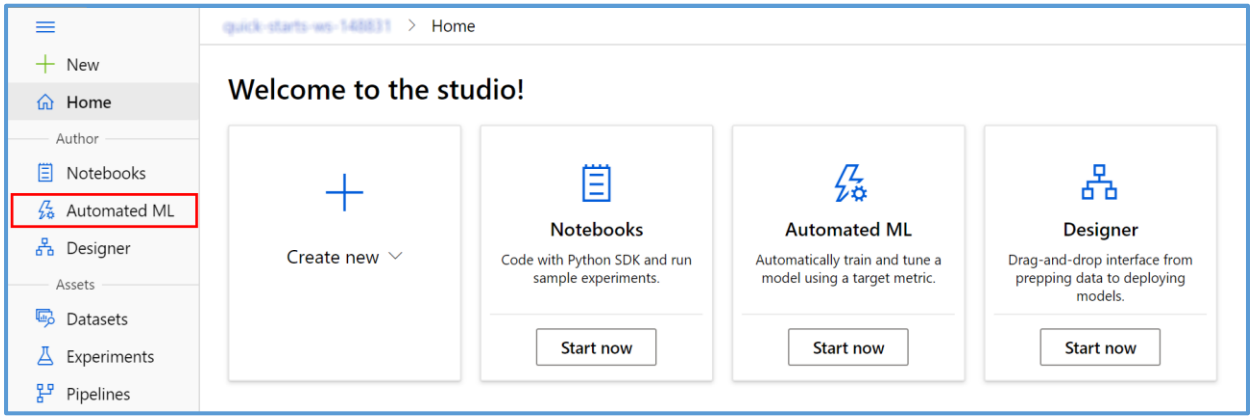


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:

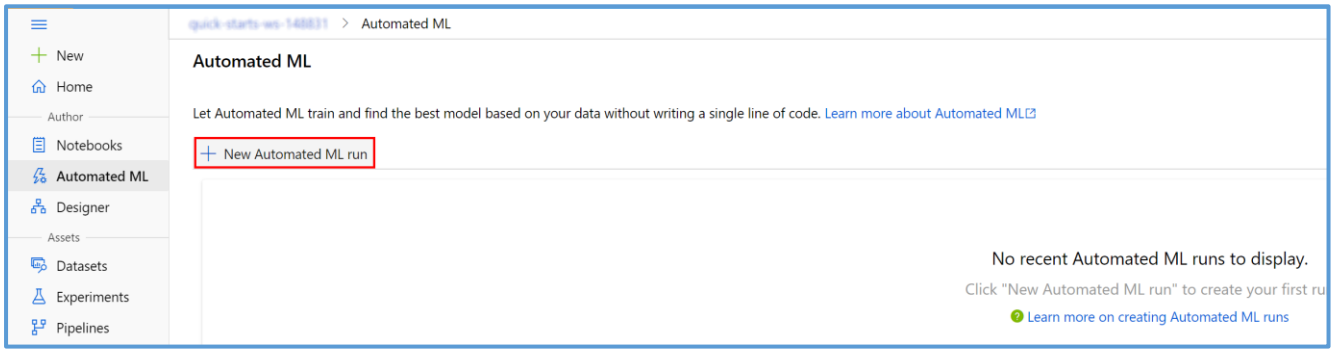


For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it doesn't matter which) and then click **Get started**.

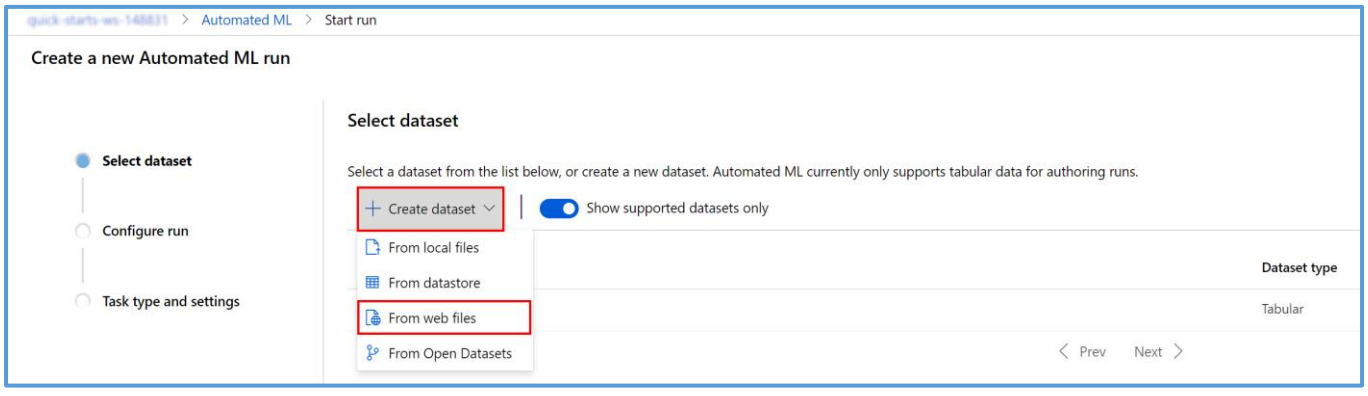
4. Select **Automated ML** in the left navigation bar.



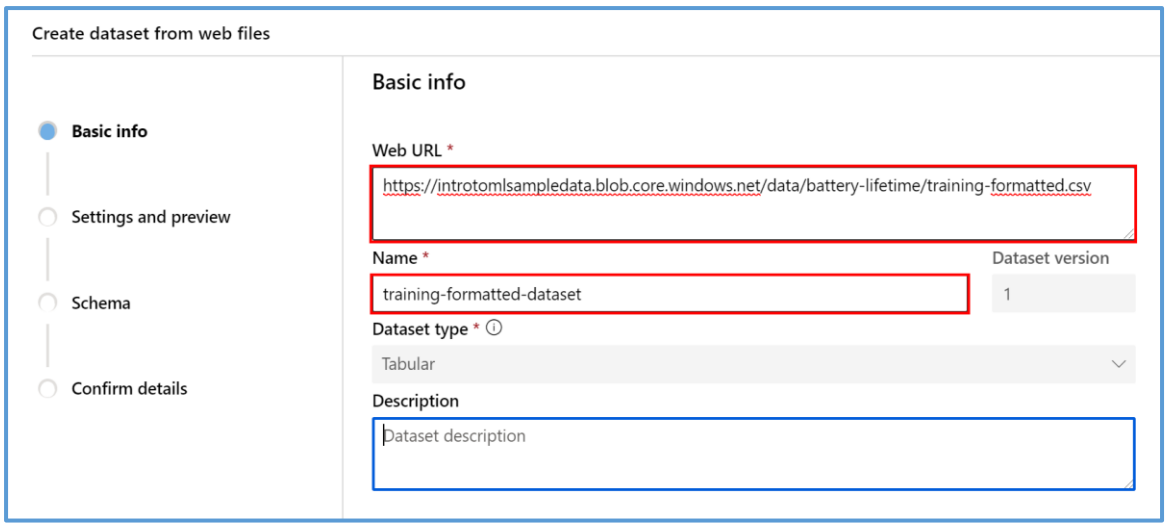
5. Select **New automated ML run** to start creating a new experiment.



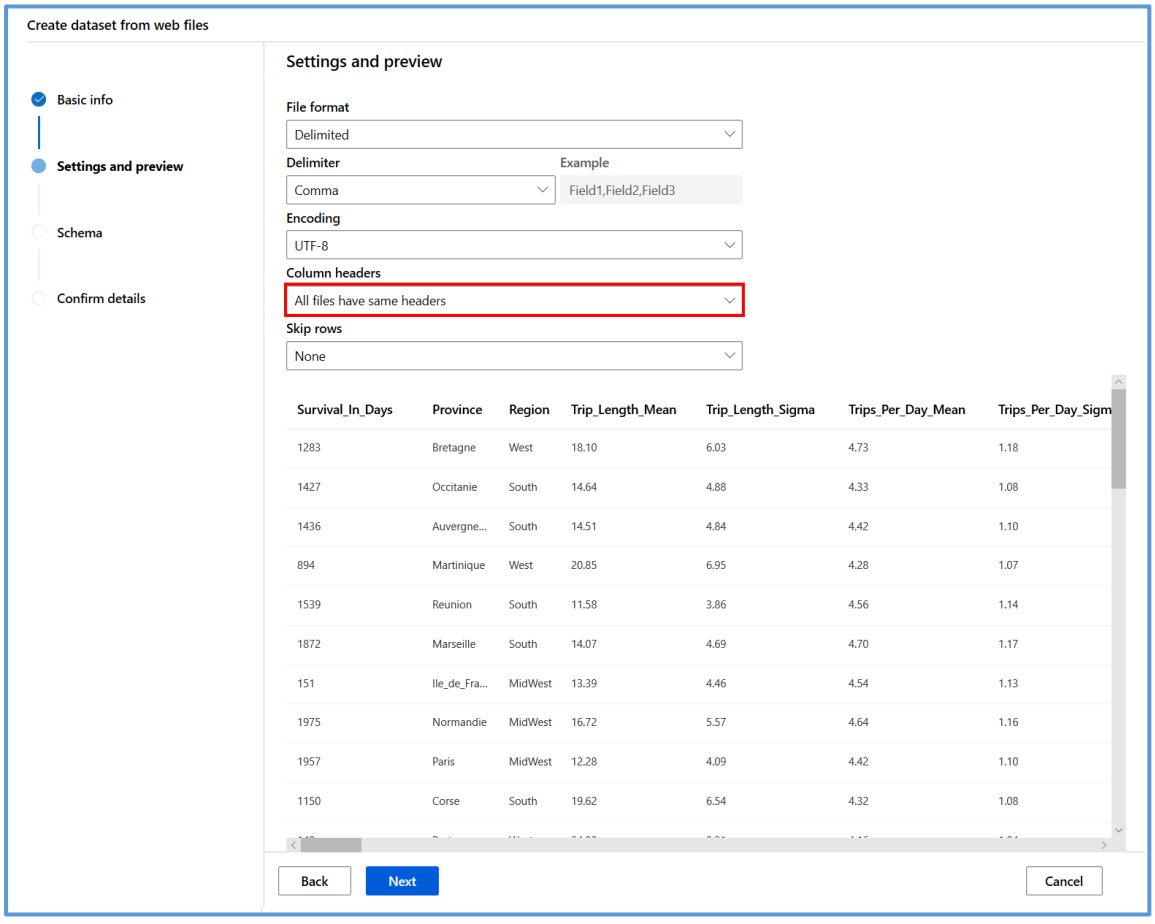
6. Select **Create dataset** and choose the **From web files** option from the drop-down.



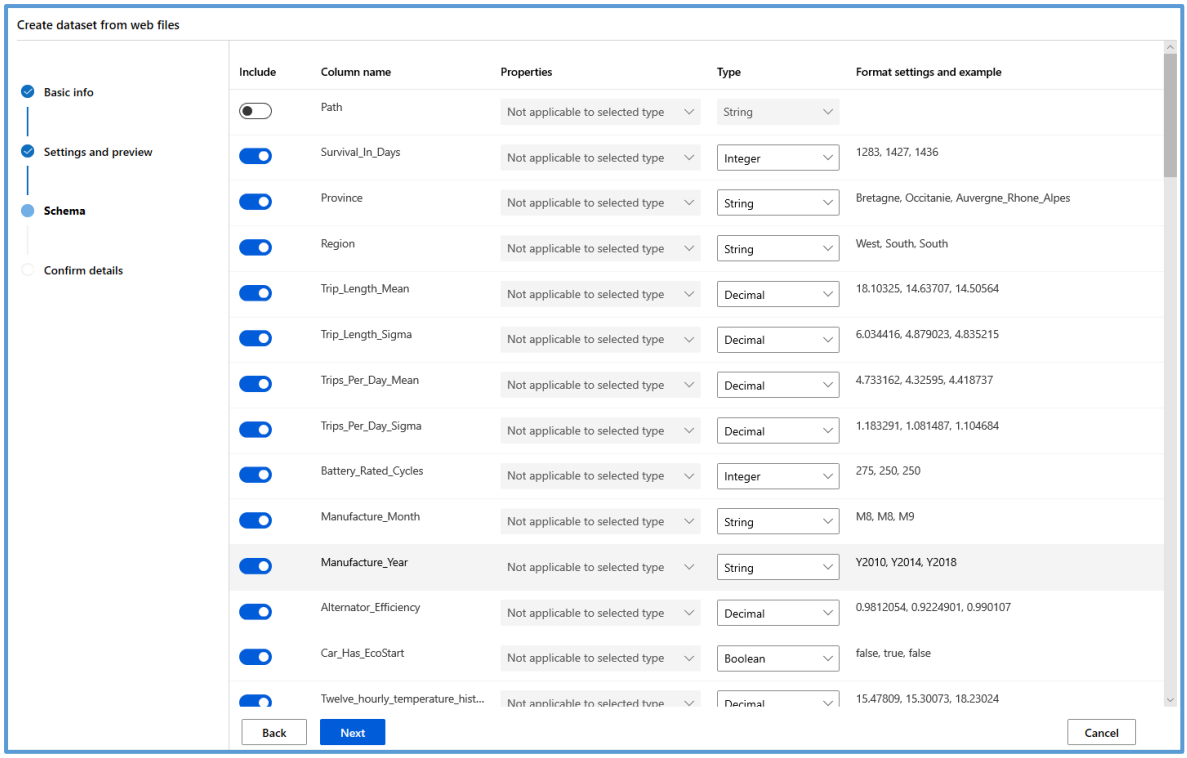
7. Fill in the training data URL in the `Web URL` field: `https://introtomlsampledata.blob.core.windows.net/data/battery-lifetime/training-formatted.csv`, make sure the name is set to `training-formatted-dataset`, and select **Next** to load a preview of the parsed training data.



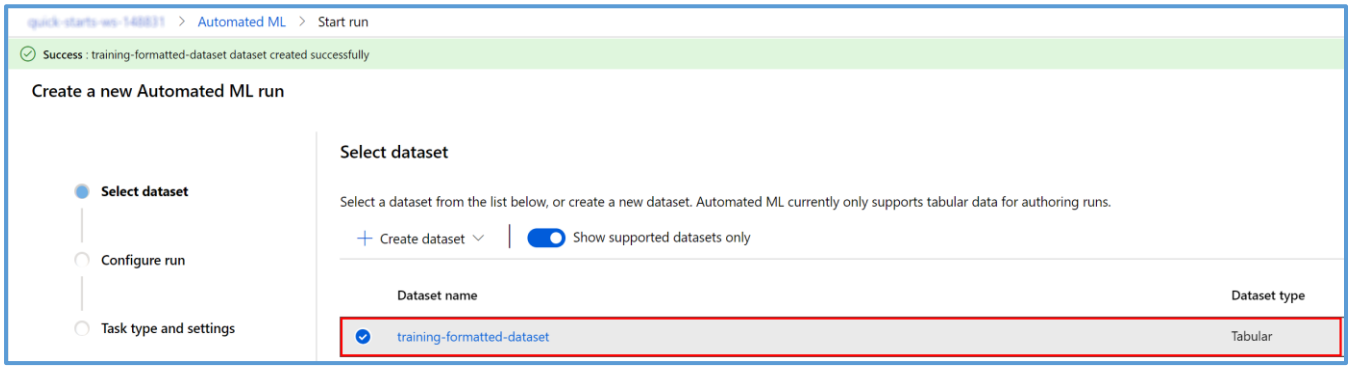
8. In the Settings and preview page, for the Column headers field, select All files have same headers. Scroll to the right to observe all of the columns in the data.



9. Select **Next** to check the schema and then confirm the dataset details by selecting **Next** and then **Create** on the confirmation page.

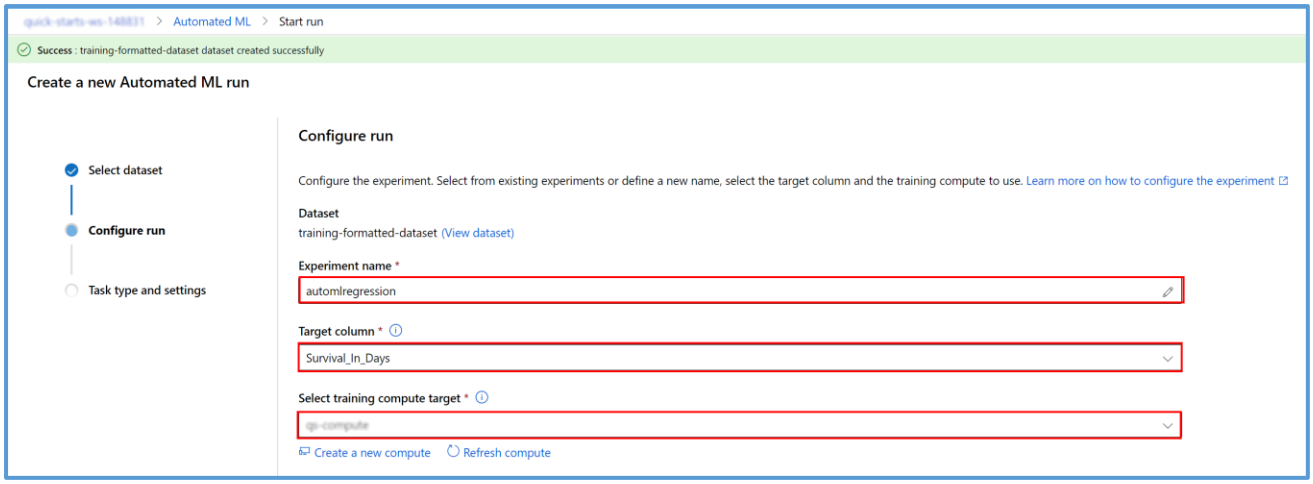


10. Now you should be able to select the newly created dataset for your experiment. Select the `training-formatted-dataset` dataset and select **Next** to move to the experiment run details page.

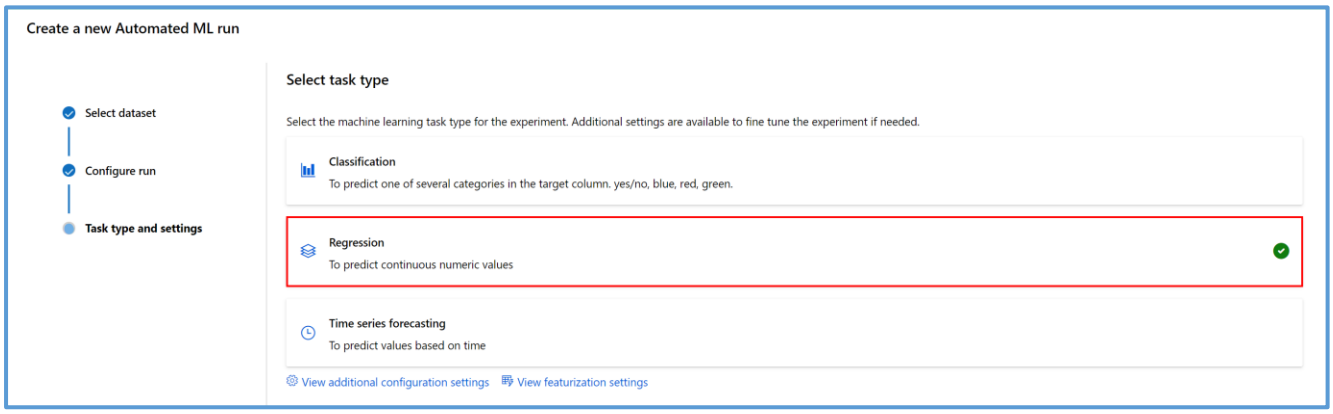


11. You will now configure the Auto ML run basic settings by providing the following values for the experiment name, target column and training compute:

    - Experiment name: **automlregression**
    - Target column: select **Survival_In_Days**
    - Select training compute target: : select **qs-compute**

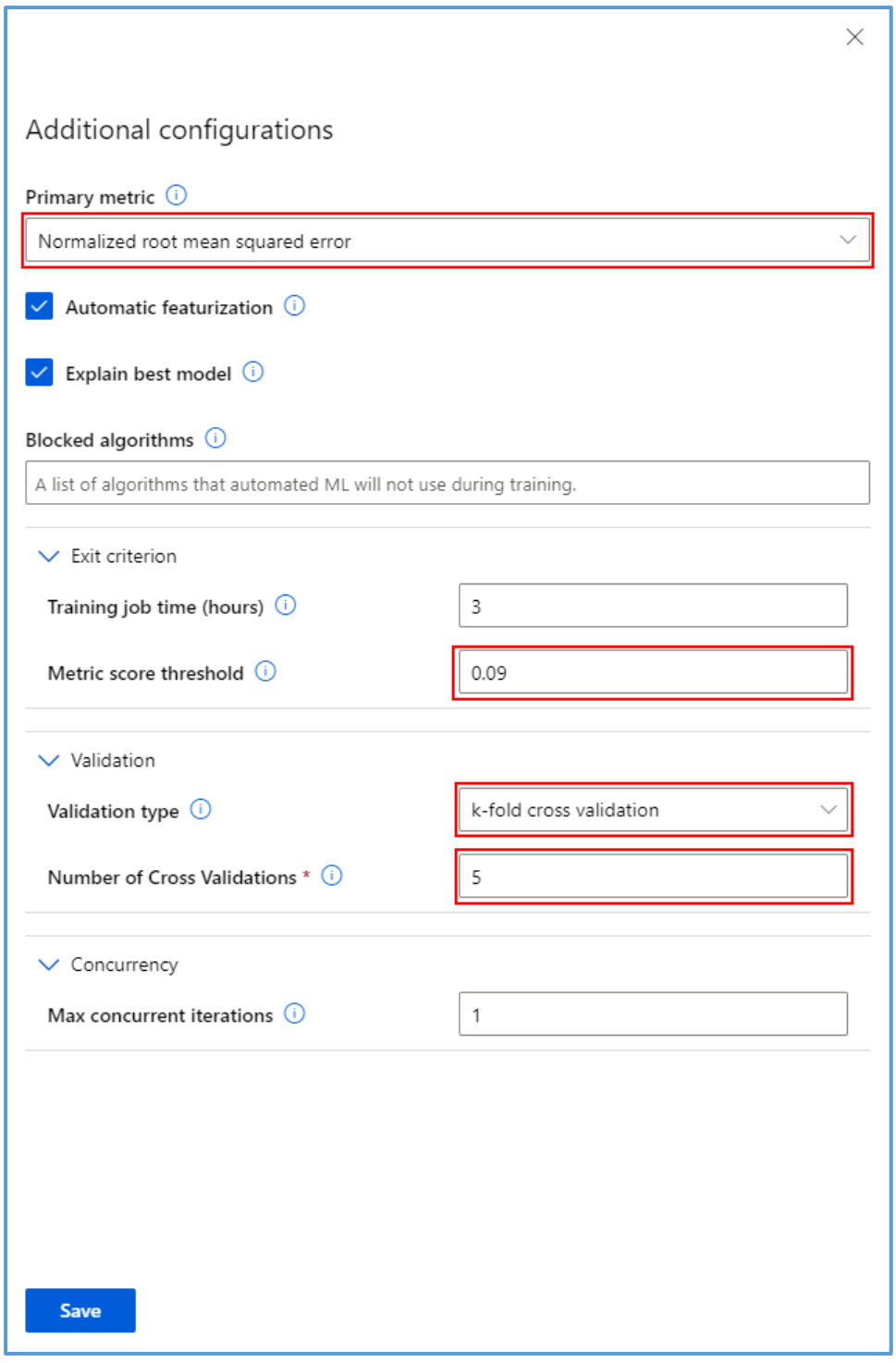


12. Select **Next** and select **Regression** in the Task type and settings page.

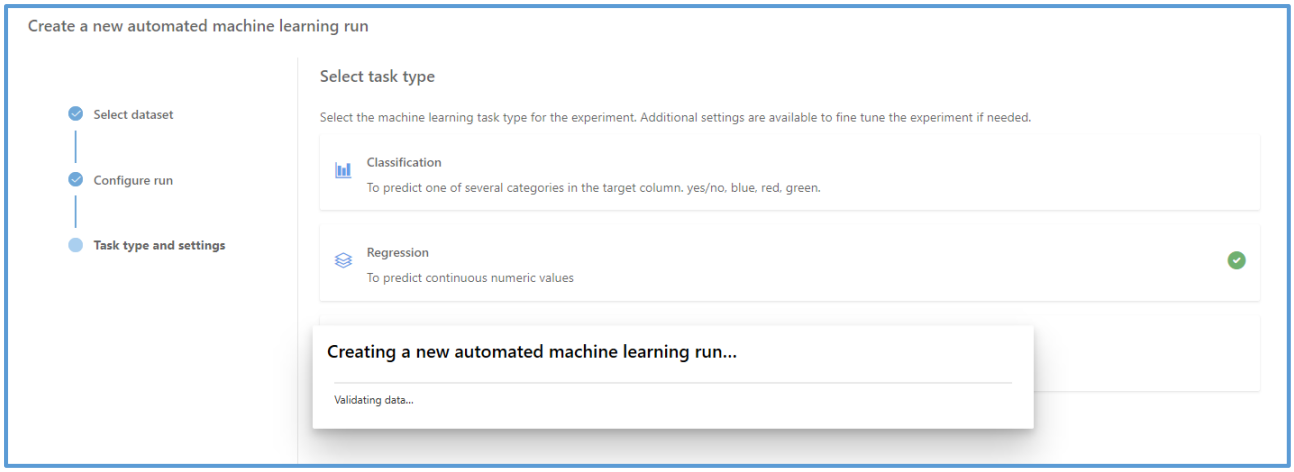


13. Select **View additional configuration settings** to open the advanced settings section. Provide the following settings:

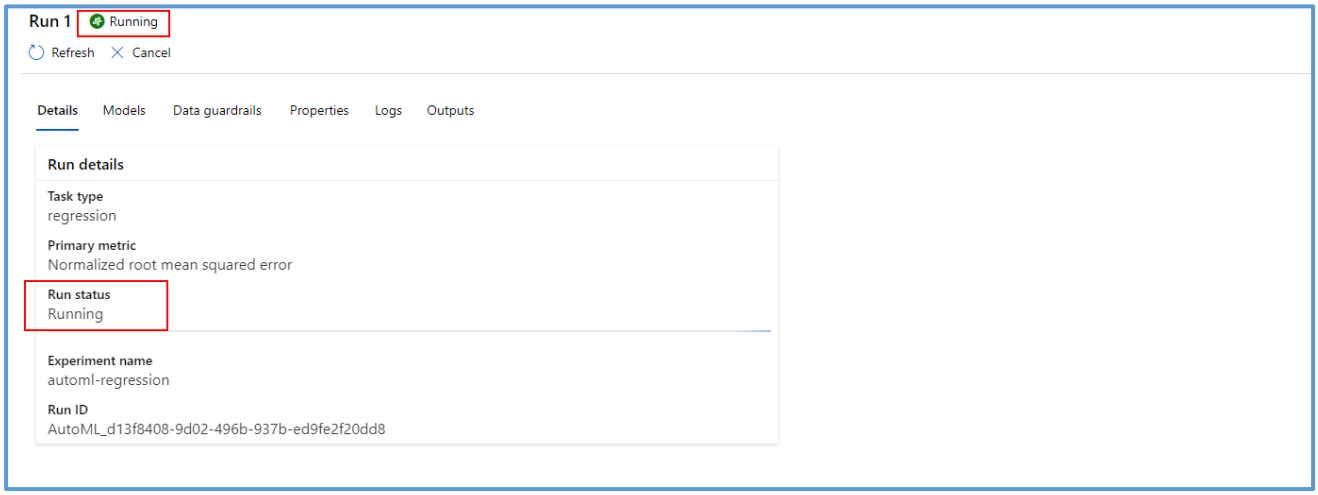
    - Primary metric: **Normalized root mean squared error**
    - Exit criterion > Metric score threshold: **0.09**
    - Validation > Validation type: **k-fold cross validation**
    - Validation > Number of Cross Validations: **5**
    - Concurrency > Max concurrent iterations: **1**



14. Select **Save** and then **Finish** to begin the automated machine learning process.

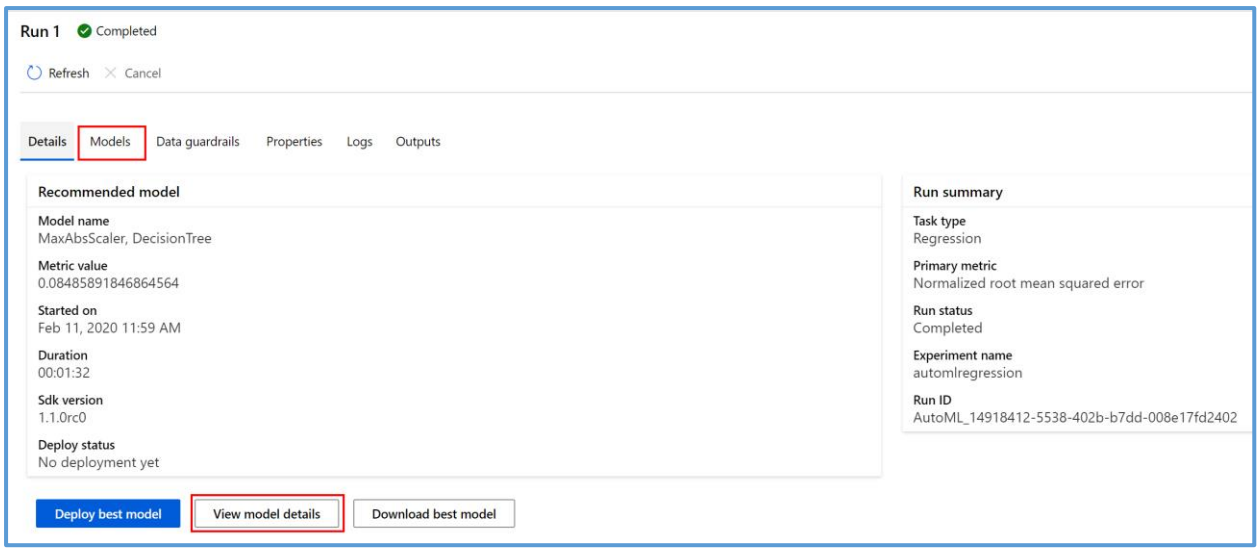


15. Wait until the Run status becomes **Running** in the Run Detail page.

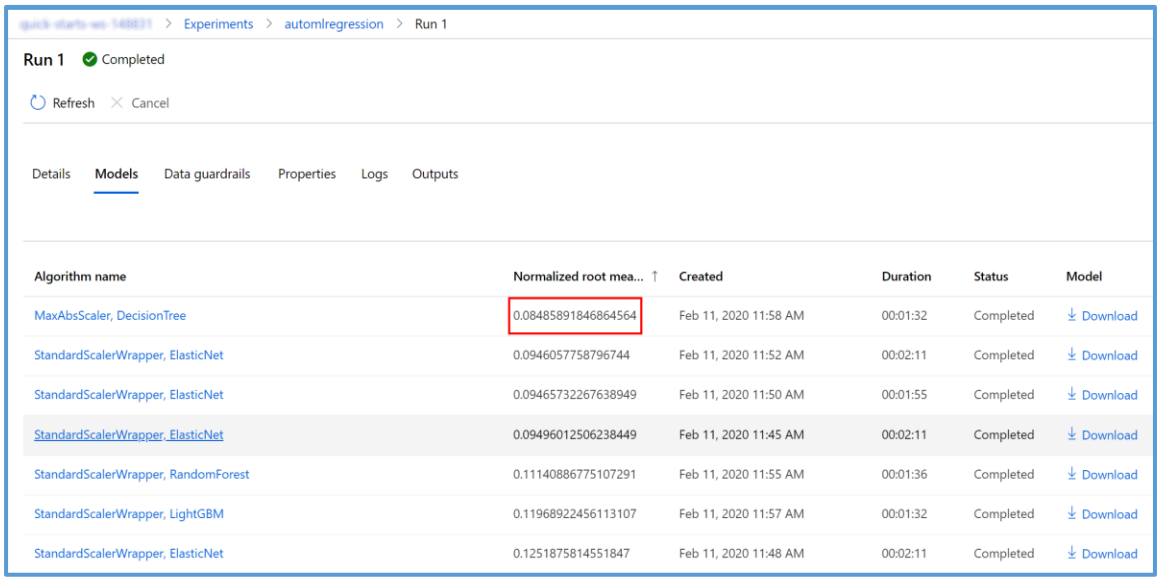


Task 2: Review the experiment run results

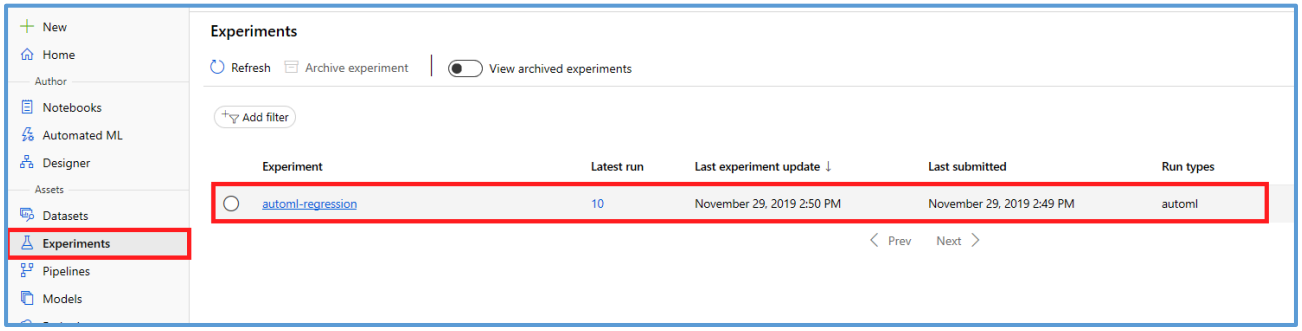
1. The experiment will run for about *15 minutes*. While it runs and once it completes, you should check the Models tab on the Run Detail page to observe the model performance for the primary metric for different runs.



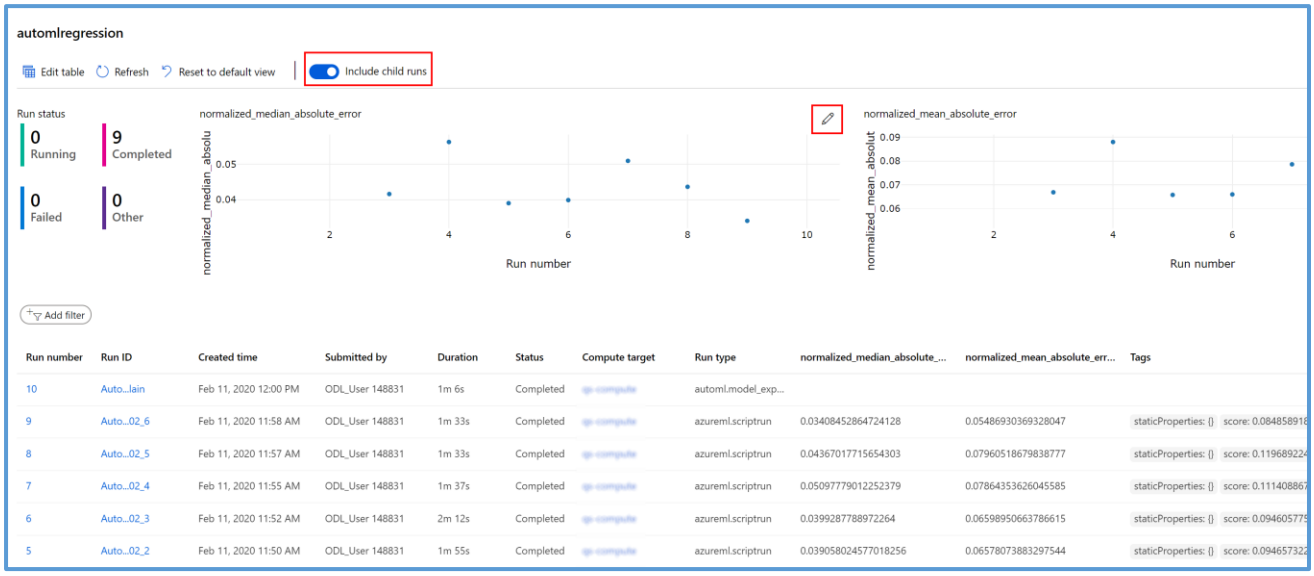
2. In the models list, notice at the top the iteration with the best **normalized root mean square error** score. Note that the normalized root mean square error measures the error between the predicted value and actual value. In this case, the model with the lowest normalized root mean square error is the best model.



3. Select **Experiments** on the left navigation pane and select the experiment automlregression to see the list of available runs.



4. Select the option to **Include child runs** to be able to examine model performance for the primary metric of different runs. By default, the left chart describes the `normalized_median_absolute_error` value for each run. Select the pen icon on the right corner of the `normalized_median_absolute_error` chart to configure the `normalized_root_mean_square_error` metric representation.



Next Steps

Congratulations! You have trained a simple time-series forecasting model using automated machine learning in the visual interface. You can continue to experiment in the environment but are free to close the lab environment tab and return to the Udacity portal to continue with the lesson.

Chapter 24: Walkthrough Forecasting

- Regression type of task using AutoML.
- Threshold was set to 0.09. Metric value needs to be lower the better. If we lower the threshold then the AutoML would test various other algorithms for longer period of time in order to get the best model.

Chapter 25: Lesson Summary

In this lesson, you have learned the fundamentals of **deep learning**, including:

- The differences between classical machine learning and deep learning
- The benefits and applications of Deep Learning

- How to train your first neural network model

  Next, you learned about some of the most important **specialized cases of model training**, including:

- *Similarity learning* and the basic features of a recommendation engine
- *Text classification* and the fundamentals of processing text in machine learning
- *Feature learning*, an essential task in feature engineering
- Anomaly detection
- Time-series forecasting.
  Along the way, you got practice with several hands-on labs, using the Designer in Azure Machine Learning Studio to train a simple neural network, a recommendation engine, a text classifier, and a time-series forecasting model.