There are four kinds of compute resource you can create:

- **Compute Instances**: Development workstations that data scientists can use to work with data and models. –

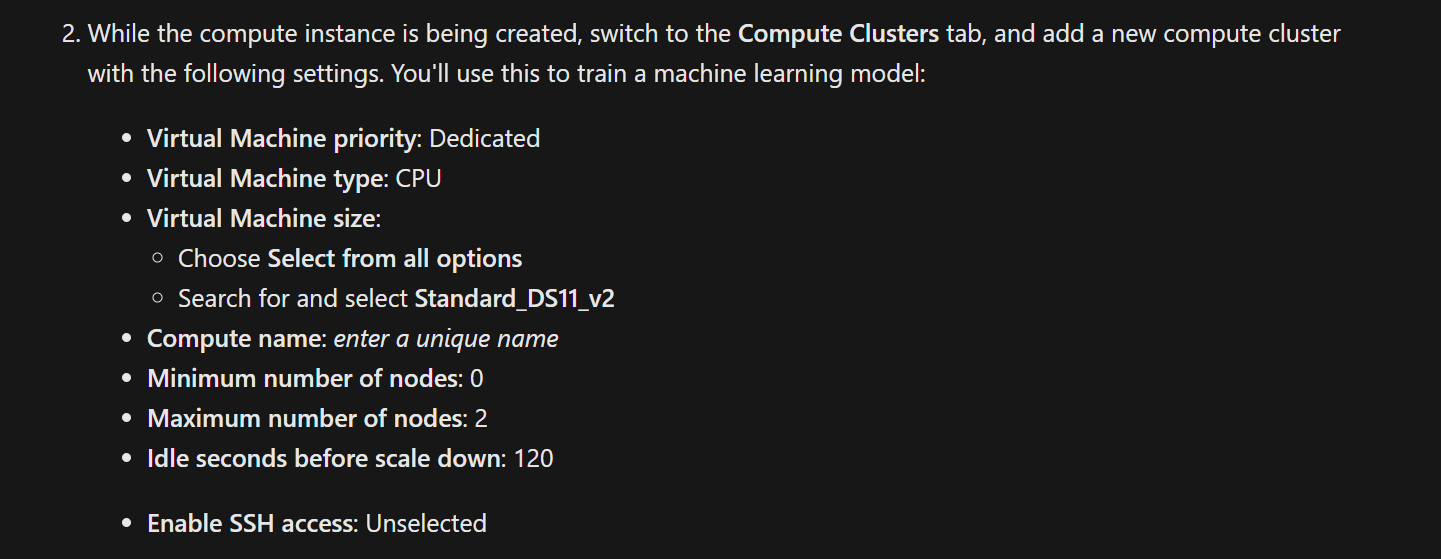
**Compute Clusters**: Scalable clusters of virtual machines for on-demand processing of experiment code.

- **Inference Clusters**: Deployment targets for predictive services that use your trained models. - **Attached Compute**: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.

Compute Instance used:

* **Virtual Machine type**: CPU
* **Virtual Machine size**:
  + Choose **Select from all options**
  + Search for and select **Standard\_DS11\_v2**
* **Compute name**: *enter a unique name*
* **Enable SSH access**: Unselected

Created a cluster



**Train a machine learning model**

Completed100 XP

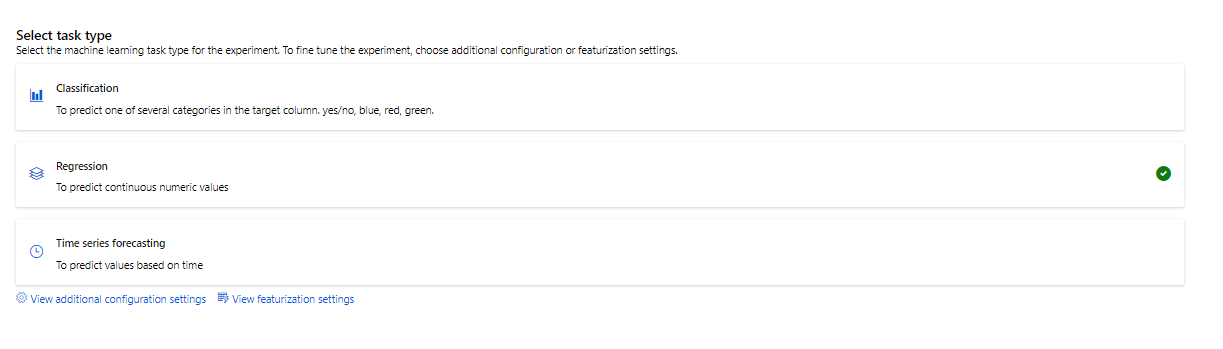
* 15 minutes

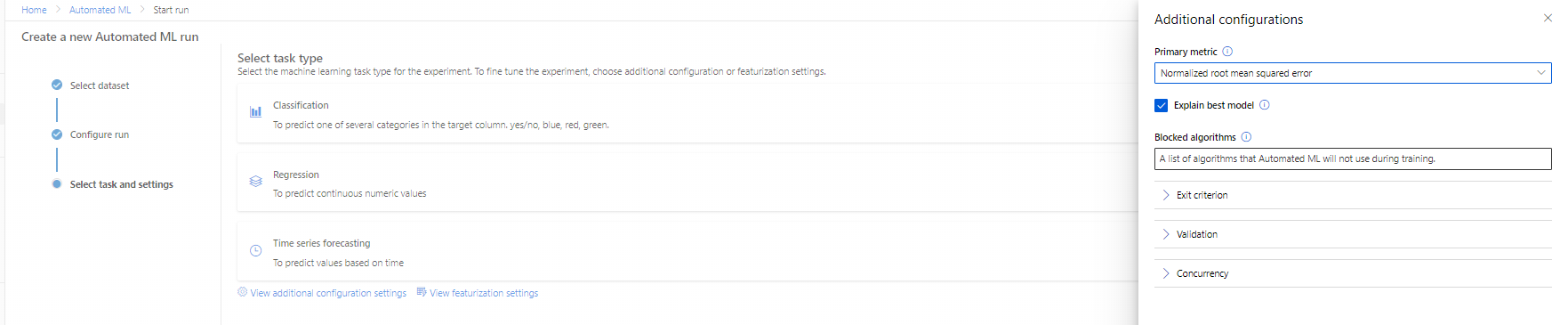
Azure Machine Learning includes an *automated machine learning* capability that leverages the scalability of cloud compute to automatically try multiple pre-processing techniques and model-training algorithms in parallel to find the best performing supervised machine learning model for your data.

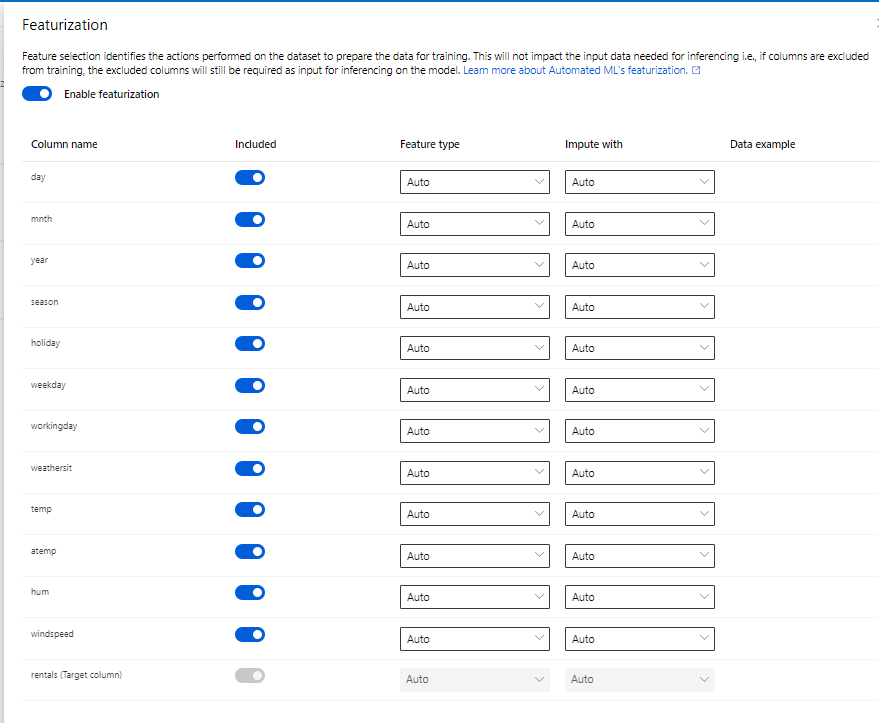
**Note**

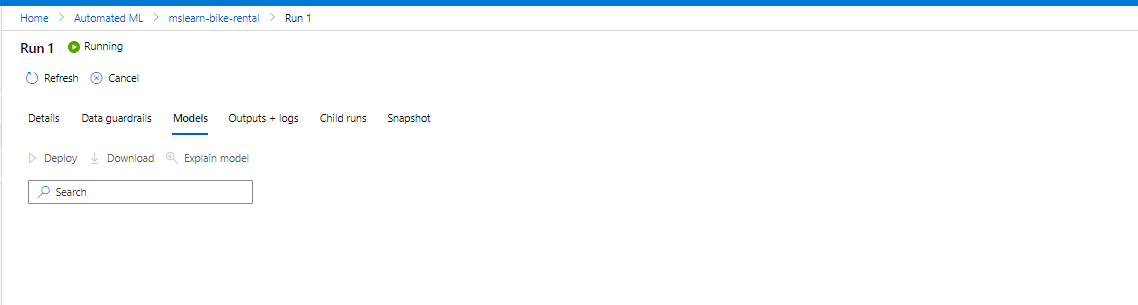
The automated machine learning capability in Azure Machine Learning supports *supervised* machine learning models - in other words, models for which the training data includes known label values. You can use automated machine learning to train models for:

* **Classification** (predicting categories or *classes*)
* **Regression** (predicting numeric values)
* **Time series forecasting** (regression with a time-series element, enabling you to predict numeric values at a future point in time)









## Review the best model

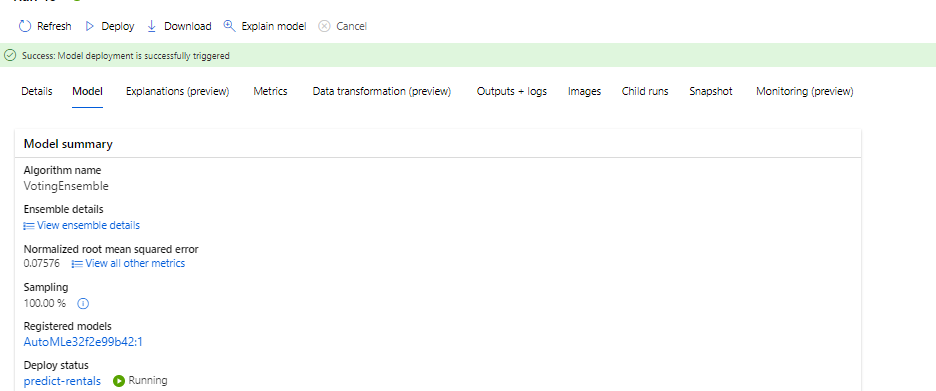
After the experiment has finished; you can review the best performing model that was generated (note that in this case, we used exit criteria to stop the experiment - so the "best" model found by the experiment may not be the best possible model, just the best one found within the time allowed for this exercise!).

## Deploy a predictive service

In Azure Machine Learning, you can deploy a service as an Azure Container Instances (ACI) or to an Azure Kubernetes Service (AKS) cluster. For production scenarios, an AKS deployment is recommended, for which you must create an inference cluster compute target.

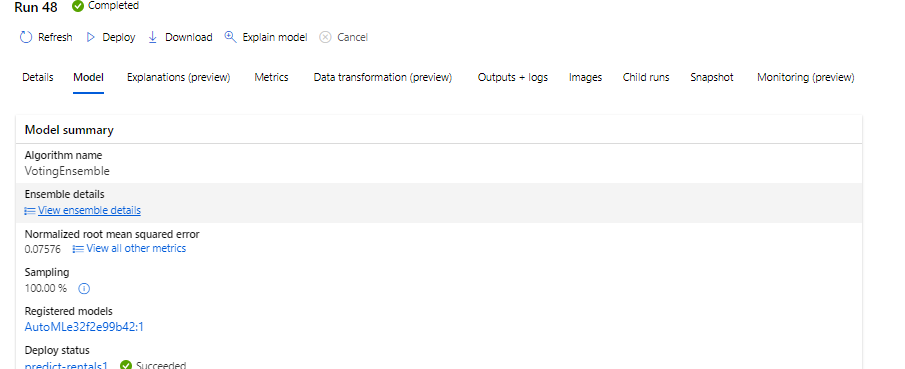
Select the algorithm name for the best model. Then, on the **Model** tab, use the **Deploy** button to deploy the model with the following settings:

* **Name**: predict-rentals
* **Description**: Predict cycle rentals
* **Compute type**: Azure Container Instance
* **Enable authentication**: Sele





1. Wait for the deployment to start - this may take a few seconds. Then, in the **Model summary** section, observe the **Deploy status** for the **predict-**
2. **rentals** service, which should be **Running**. Wait for this status to change to **Successful**. You may need to select **↻ Refresh** periodically.





## Test the deployed service

Now that you've deployed a service, you can test it using some simple code.

1. With the **Consume** page for the **predict-rentals** service page open in your browser, open a new browser tab and open a second instance of [Azure Machine Learning studio](https://ml.azure.com/). Then in the new tab, view the **Notebooks** page (under **Author**).
2. In the **Notebooks** page, under **My files**, use the **🗋** button to create a new file with the following settings:
   * **File location**: Users/your user name
   * **File name**: Test-Bikes.ipynb
   * **File type**: Notebook
   * **Overwrite if already exists**: Selected
3. When the new notebook has been created, ensure that the compute instance you created previously is selected in the **Compute** box, and that it has a status of **Running**.
4. Use the **≪** button to collapse the file explorer pane and give you more room to focus on the **Test-Bikes.ipynb** notebook tab.
5. In the rectangular cell that has been created in the notebook, paste the following code:

PythonCopy

endpoint = 'YOUR\_ENDPOINT' #Replace with your endpoint

key = 'YOUR\_KEY' #Replace with your key

import json

import requests

#An array of features based on five-day weather forecast

x = [[1,1,2022,1,0,6,0,2,0.344167,0.363625,0.805833,0.160446],

[2,1,2022,1,0,0,0,2,0.363478,0.353739,0.696087,0.248539],

[3,1,2022,1,0,1,1,1,0.196364,0.189405,0.437273,0.248309],

[4,1,2022,1,0,2,1,1,0.2,0.212122,0.590435,0.160296],

[5,1,2022,1,0,3,1,1,0.226957,0.22927,0.436957,0.1869]]

#Convert the array to JSON format

input\_json = json.dumps({"data": x})

#Set the content type and authentication for the request

headers = {"Content-Type":"application/json",

"Authorization":"Bearer " + key}

#Send the request

response = requests.post(endpoint, input\_json, headers=headers)

#If we got a valid response, display the predictions

if response.status\_code == 200:

y = json.loads(response.json())

print("Predictions:")

for i in range(len(x)):

print (" Day: {}. Predicted rentals: {}".format(i+1, max(0, round(y["result"][i]))))

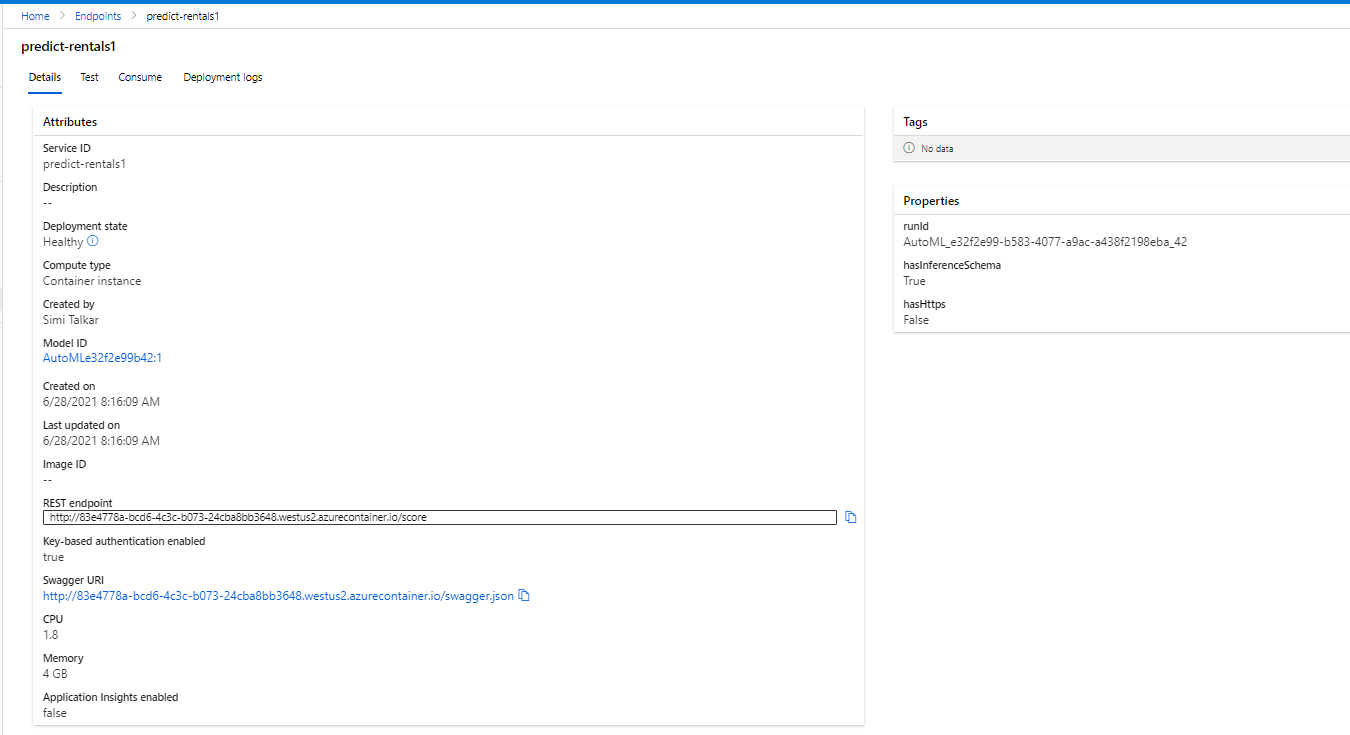
else:

print(response)

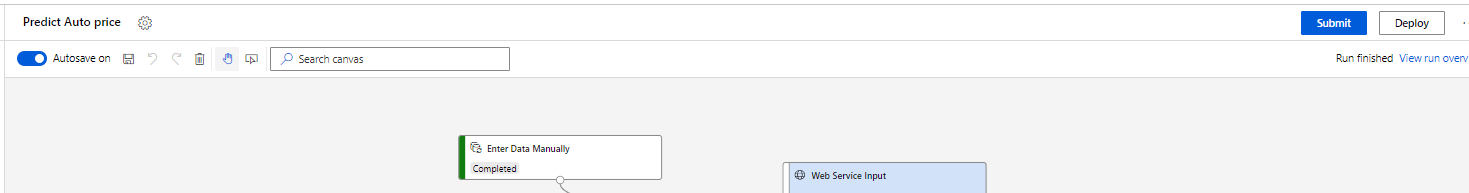
**Note**

Don't worry too much about the details of the code. It just defines features for a five day period using hypothetical weather forecast data, and uses the **predict-rentals** service you created to predict cycle rentals for those five days.

1. Switch to the browser tab containing the **Consume** page for the **predict-rentals** service, and copy the REST endpoint for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_ENDPOINT.
2. Switch to the browser tab containing the **Consume** page for the **predict-rentals** service, and copy the Primary Key for your service. The switch back to the tab containing the notebook and paste the key into the code, replacing YOUR\_KEY.
3. Save the notebook, Then use the **▷** button next to the cell to run the code.
4. Verify that predicted number of rentals for each day in the five day period are returned.







## Clean-up

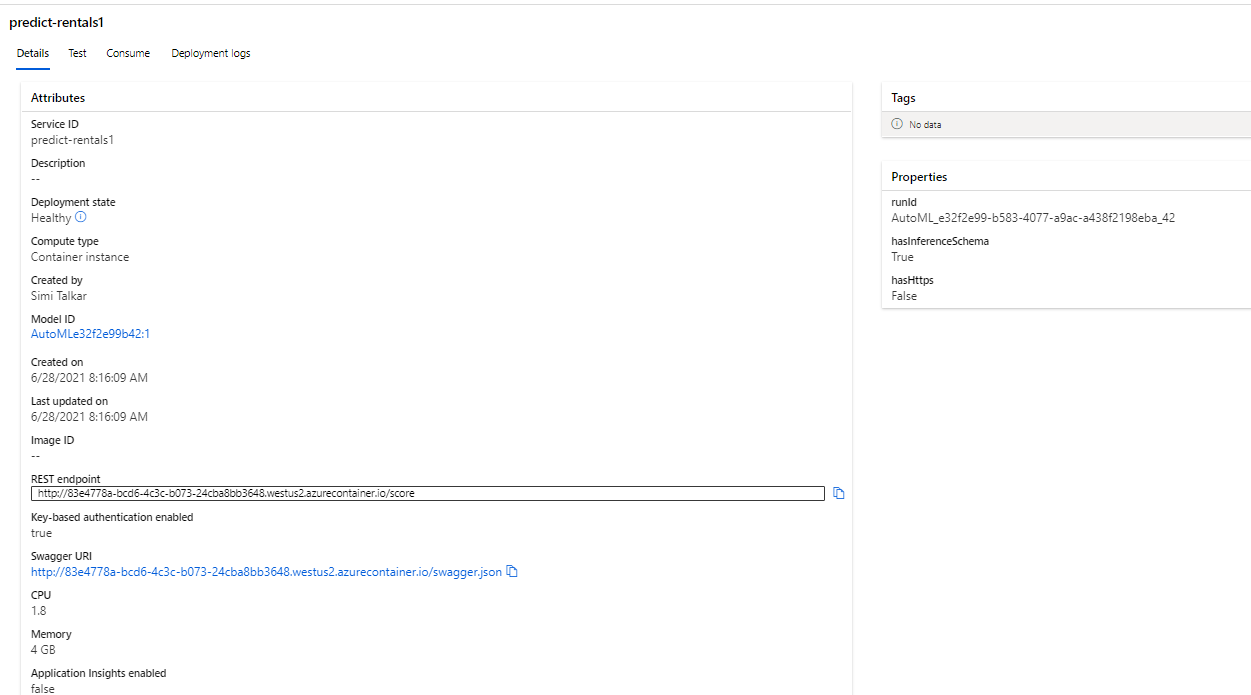
The web service you created is hosted in an Azure Container Instance. If you don't intend to experiment with it further, you should delete the endpoint to avoid accruing unnecessary Azure usage. You should also stop the training cluster and compute instance resources until you need them again.

1. In [Azure Machine Learning studio](https://ml.azure.com/), on the **Endpoints** tab, select the **predict-rentals** endpoint. Then select **Delete** (🗑) and confirm that you want to delete the endpoint.
2. On the **Compute** page, on the **Compute Instances** tab, select your compute instance and then select **Stop**.

Stopping your compute ensures your subscription won't be charged for compute resources. You will however be charged a small amount for data storage as long as the Azure Machine Learning workspace exists in your subscription. If you have finished exploring Azure Machine Learning, you can delete the Azure Machine Learning workspace and associated resources. However, if you plan to complete any other labs in this series, you will need to recreate it.

To delete your workspace:

1. In the [**Azure portal**](https://portal.azure.com/), in the **Resource groups** page, open the resource group you specified when creating your Azure Machine Learning workspace.
2. Click **Delete resource group**, type the resource group name to confirm you want to delete it, and select **Delete**.



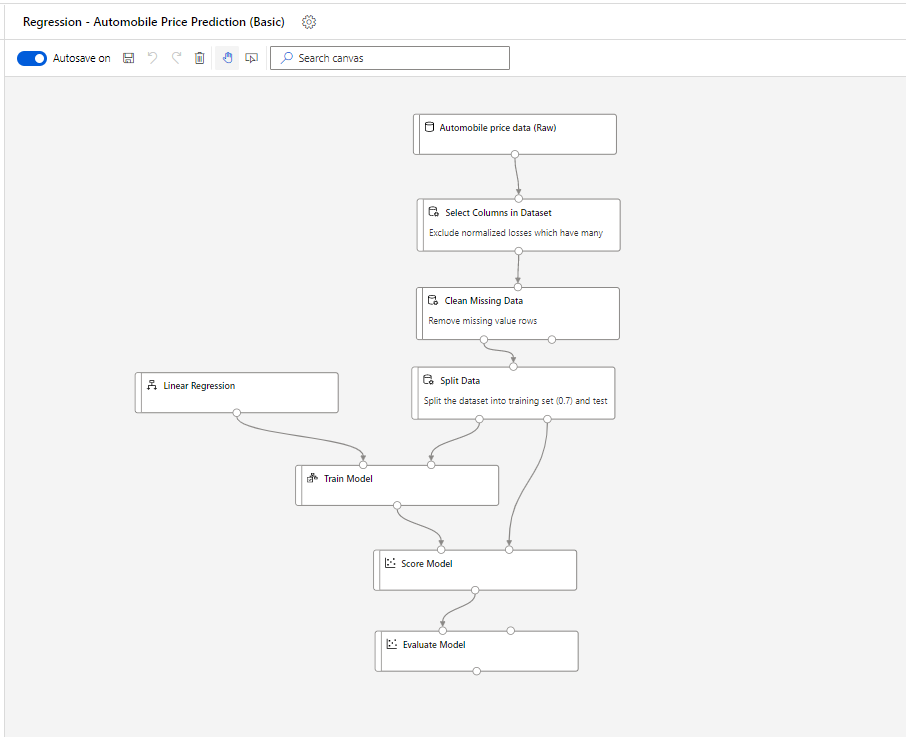
You can use Microsoft Azure Machine Learning designer to create regression models by using a drag and drop visual interface, without needing to write any code.

In this module, you'll learn how to:

* Use Azure Machine Learning designer to train a regression model.
* Use a regression model for inferencing.
* Deploy a regression model as a service.
* **Compute Instances**: Development workstations that data scientists can use to work with data and models.
* **Compute Clusters**: Scalable clusters of virtual machines for on-demand processing of experiment code.
* **Inference Clusters**: Deployment targets for predictive services that use your trained models.

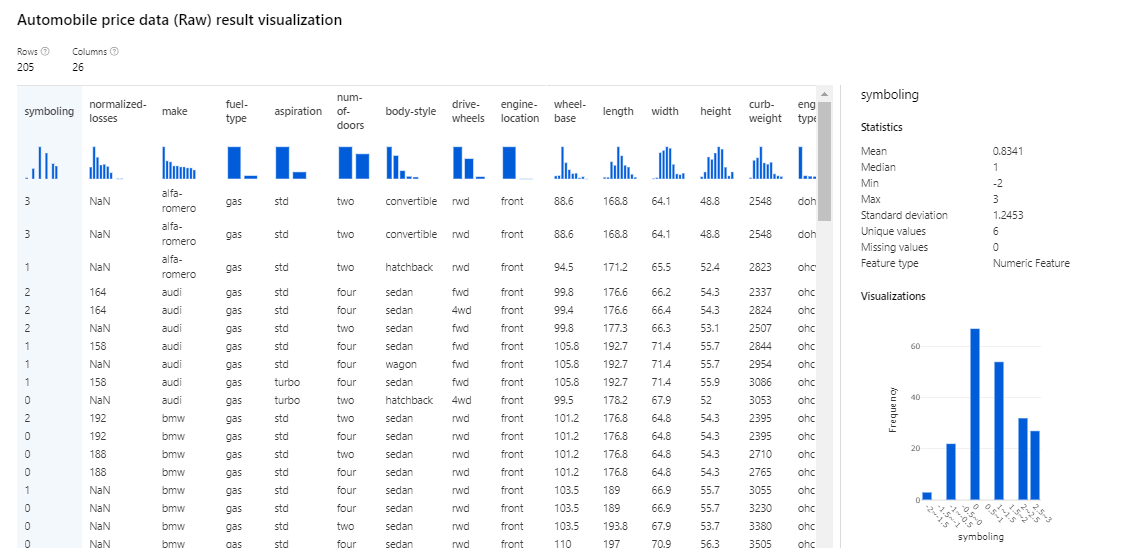


* **Attached Compute**: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.

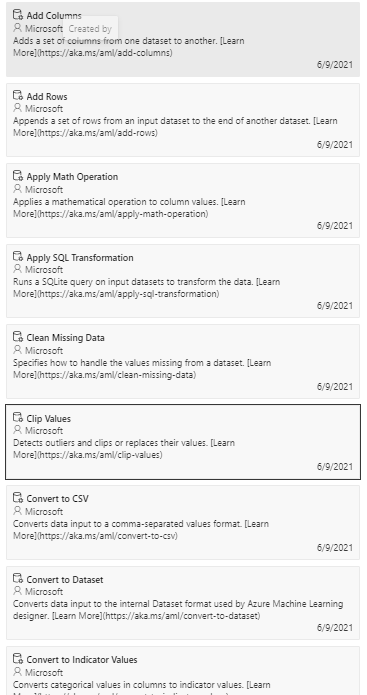


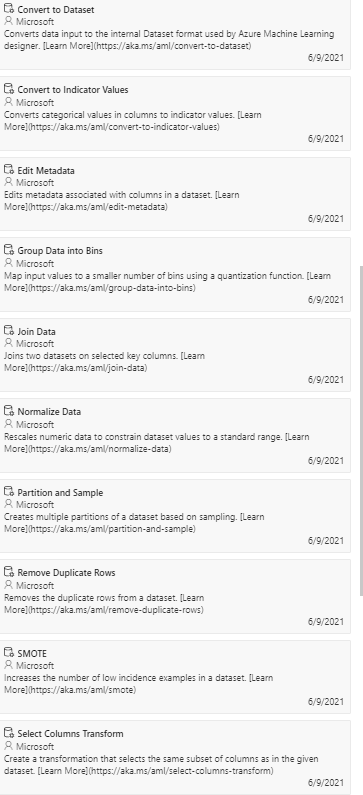
1. In Designer

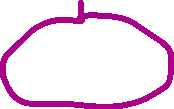
I created a new pipeline – by double clicking the name you can change the name of the pipeline

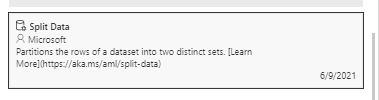
1. I selected a dataset and dropped it into the canvas and then visualized it by right clicking
2. View the statistics for the **bore**, **stroke**, and **horsepower** columns, noting the number of missing values. These columns have significantly fewer missing values than **normalized-losses**, so they may still be useful in predicting **price** if you exclude the rows where the values are missing from training.
3. Compare the values in the **stroke**, **peak-rpm**, and **city-mpg** columns. These are all measured in different scales, and its possible that the larger values for **peak-rpm** might bias the training algorithm and create an over-dependency on this column compared to columns with lower values, such as **stroke**. Typically, data scientists mitigate this possible bias by *normalizing* the numeric columns so they're on the similar scales.

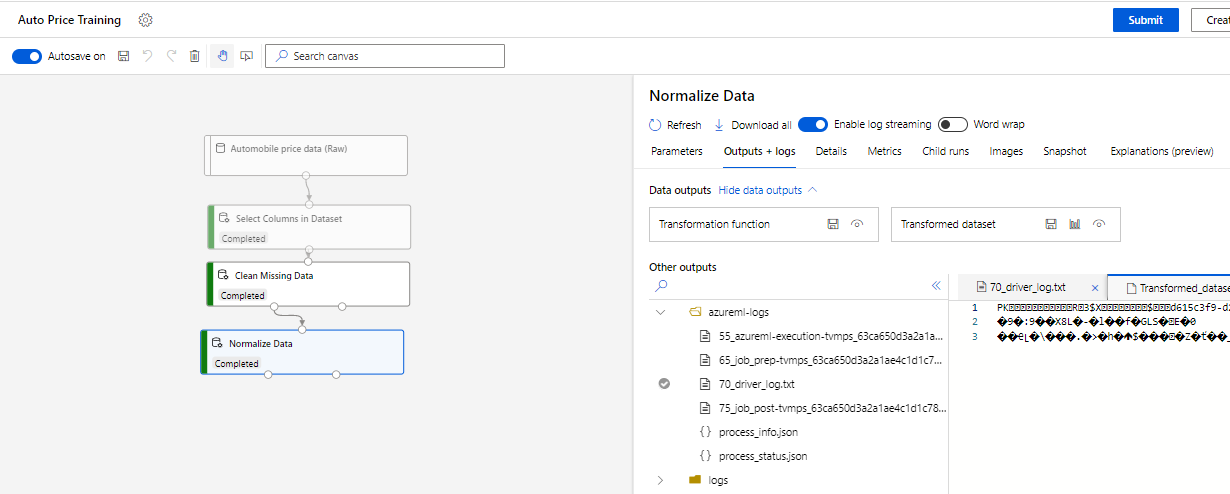
AVAILABLE DATA TRANSFORMATIONS



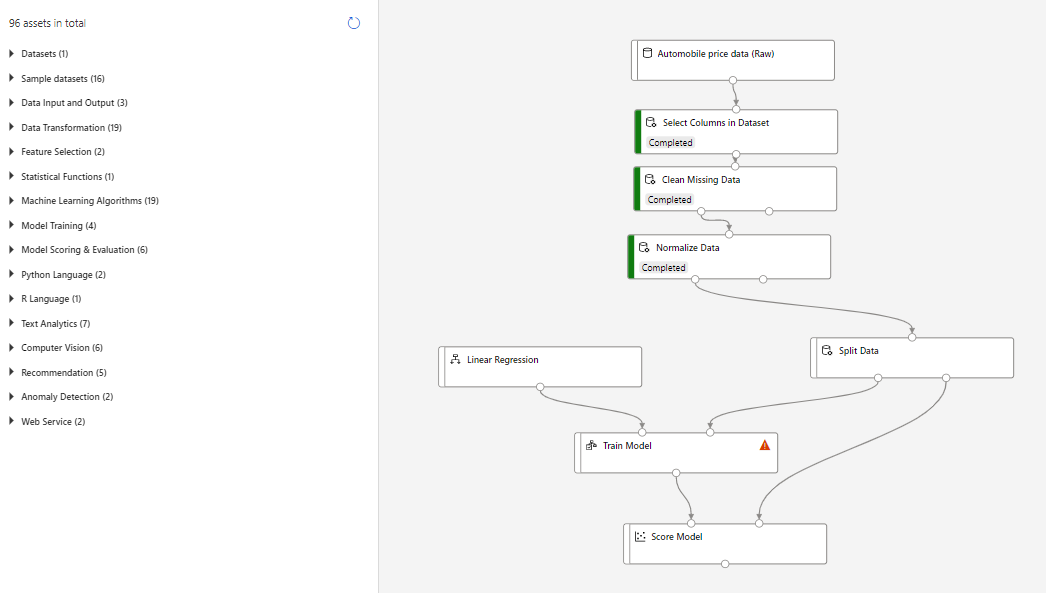


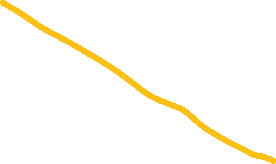
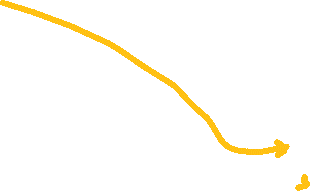
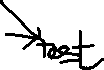












RUN THIS PIPELINE BEFORE ADDING EVALUATE

## Add an Evaluate Model module

1. Open the **Auto Price Training** pipeline you created in the previous unit if it's not already open.
2. In the pane on the left, in the **Model Scoring & Evaluation** section, drag an **Evaluate Model** module to the canvas, under the **Score Model** module, and connect the output of the **Score Model** module to the **Scored dataset** (left) input of the **Evaluate Model** module.
3. Ensure your pipeline looks like this:

When the experiment run has completed, select the **Evaluate Model** module and in the settings pane, on the **Outputs + logs** tab, under **Data outputs** in the **Evaluation results** section, use the **Visualize** icon to view the results. These include the following regression performance metrics:



* **Mean Absolute Error (MAE)**: The average difference between predicted values and true values. This value is based on the same units as the label, in this case dollars. The lower this value is, the better the model is predicting.
* **Root Mean Squared Error (RMSE)**: The square root of the mean squared difference between predicted and true values. The result is a metric based on the same unit as the label (dollars). When compared to the MAE (above), a larger difference indicates greater variance in the individual errors (for example, with some errors being very small, while others are large).



* **Relative Squared Error (RSE)**: A relative metric between 0 and 1 based on the square of the differences between predicted and true values. The closer to 0 this metric is, the better the model is performing. Because this metric is relative, it can be used to compare models where the labels are in different units.



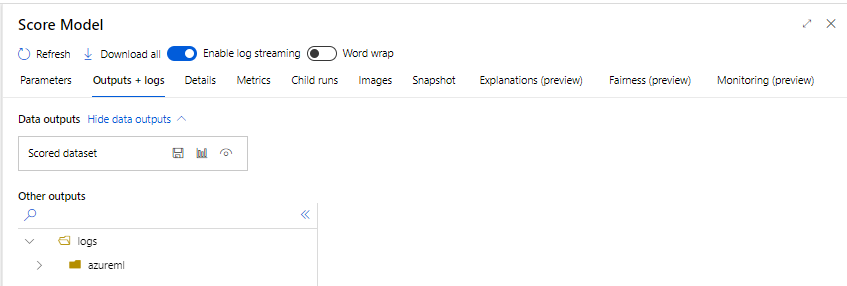
* **Relative Absolute Error (RAE)**: A relative metric between 0 and 1 based on the absolute differences between predicted and true values. The closer to 0 this metric is, the better the model is performing. Like RSE, this metric can be used to compare models where the labels are in different units.



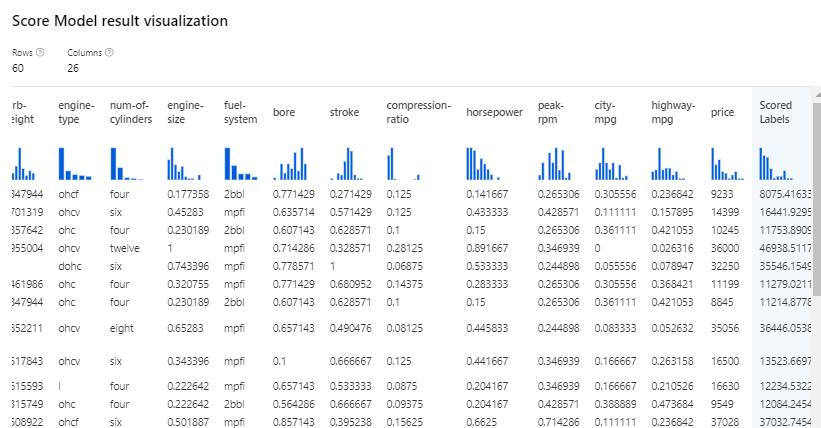
* **Coefficient of Determination (R2)**: This metric is more commonly referred to as *R-Squared*, and summarizes how much of the variance between predicted and true values is explained by the model. The closer to 1 this value is, the better the model is performing.

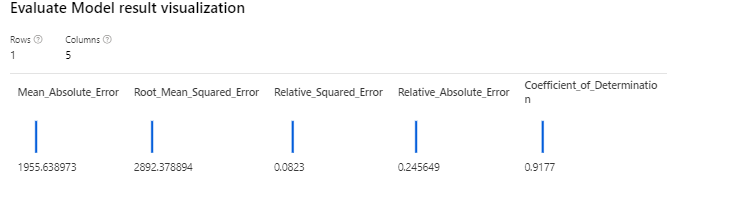


You can try a different regression algorithm and compare the results by connecting the same outputs from the **Split Data** module to a second **Train model** module (with a different algorithm) and a second **Score Model** module; and then connecting the outputs of both **Score Model** modules to the same **Evaluate Model** module for a side-by-side comparison.







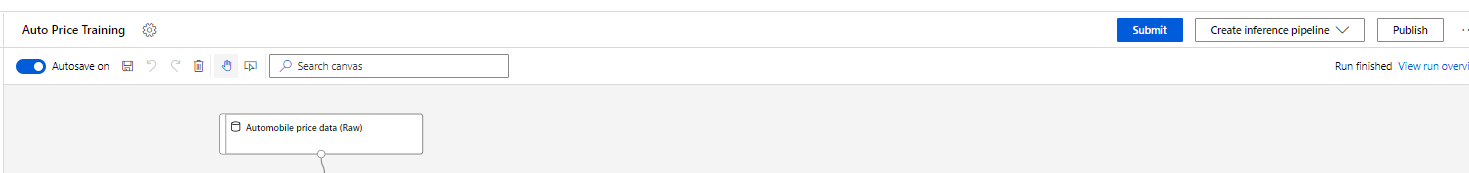


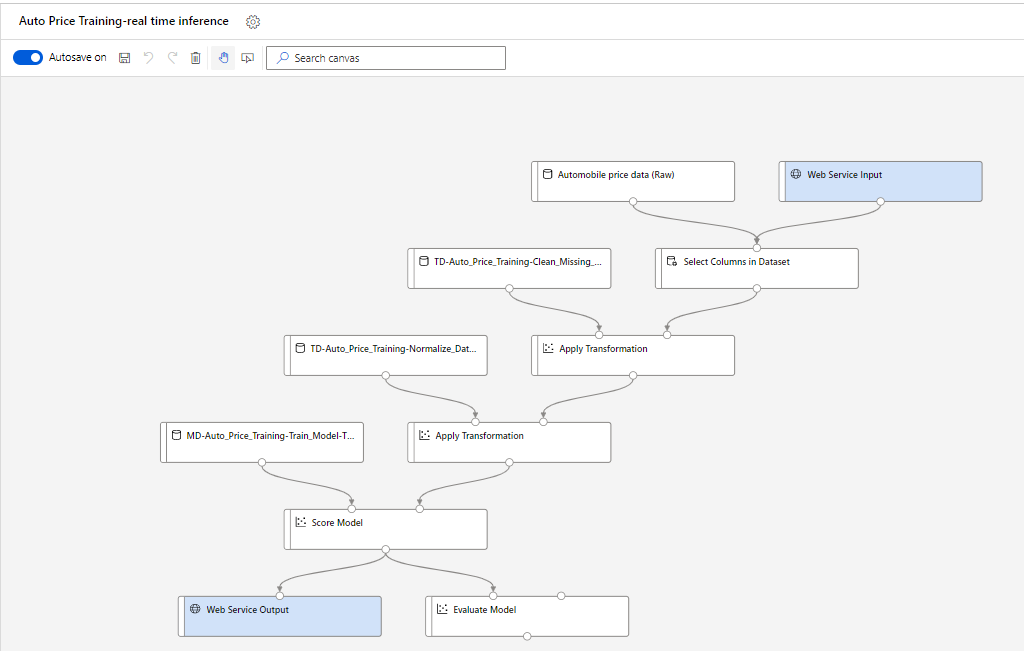
# Create an inference pipeline

Completed100 XP

* 8 minutes

After creating and running a pipeline to train the model, you need a second pipeline that performs the same data transformations for new data, and then uses the trained model to inference (in other words, predict) label values based on its features. This will form the basis for a predictive service that you can publish for applications to use.





## Test the service

Now you can test your deployed service from a client application - in this case, you'll use the code in the cell below to simulate a client application.

1. On the **Endpoints** page, open the **predict-auto-price** real-time endpoint.
2. When the **predict-auto-price** endpoint opens, view the **Consume** tab and note the following information there. You need this to connect to your deployed service from a client application.
   * The REST endpoint for your service
   * The Primary Key for your service



