# Get the Docker Image to run our model

* Open the command prompt/terminal/PowerShell
* Run the following command to get the TensorFlow image to serve models.

docker pull tensorflow/serving

# Get information about our model

* With our notebook server running from lab, go to another command prompt
* Find the name of the running TensorFlow Juypiter container

docker ps --all

* With the running container name or id connect to the container and start a bash session using the following command.

docker exec -i -t container\_name\_or\_id /bin/bash

* We can now run a command to get information about our model that we created in Lab9.

You may have to alter the path/model name if you did something different in the labs.

saved\_model\_cli show --dir notebooks/dow\_regression/1 --tag\_set serve --signature\_def serving\_default

* The command should return something like this that gives us information on the shape of our model:

The given SavedModel SignatureDef contains the following input(s):

inputs['dense\_input'] tensor\_info:

dtype: DT\_FLOAT

shape: (-1, 2)

name: serving\_default\_dense\_input:0

The given SavedModel SignatureDef contains the following output(s):

outputs['dense\_2'] tensor\_info:

dtype: DT\_FLOAT

shape: (-1, 1)

name: StatefulPartitionedCall:0

Method name is: tensorflow/serving/predict

* We can see that the model is expecting a single input, dense\_input, with a shape of (-1,2) and returns a single result with a shape of (-1, 1)
* We can use ctrl-p, ctrl-q to exit our Docker bash session.

# Mount the Model in a Docker Service

* If you recall when we set up the Juypiter notebook server, the notebook directory and by extension our model is actually being saved to a shared location on our machine instead of in the Docker Image. We can use this to share the model to our Docker image for hosting the service.
* Use the following Docker command to start the service:

docker run -t --rm -p 8501:8501 -v ~/notebooks/dow\_regression:/models/dow\_regression -e MODEL\_NAME=dow\_regression tensorflow/serving &

What this command is doing is running the tensorflow/serving image and mounting our local dow\_regression folder to /models/dow\_regression in the running container. We are also specifying the name of the model to load as dow\_regression..

The internal container service port of 8501 is being mapped to our host machine’s port 8501 so if we call localhost:8501 the request will be passed in to this container.

# Testing our Model Service

* There are several ways we can test our model service. The easiest is to use Curl at the command line (this may be slightly different on a Windows machine)

curl -d '{"inputs": [[2022.0, 107.0]]}' -X POST <http://localhost:8501/v1/models/dow_regression:predict>

Notice that our version 1 of the model becomes part of the URL, conceivably if we have multiple model versions we can control which one we are calling

* What we have done here is pass in an input in the shape that we puller from our model, a matrix tensor with one dimension. That dimension has two values, the year to get a prediction for and the year – 1950. In this case we are looking for the prediction on year 2022.
* I received the following result:

{

"outputs": [

[

23032.1406

]

]

}

* What if we wanted to call this in a .Net standard Website? To do this open up the Lab11 project in the Projects folder.
* In the Constants TensorFlowML fils set the value of the RegressionEndpoint variable.

<http://localhost:8501/v1/models/dow_regression:predict>

* Open the Controllers\HomeController file. Add the following to implement the “Predict” function

var returnValue = new IndexModel

{

YearToPredict = yearToPredict

};

var intYear = int.Parse(yearToPredict);

var data = $"{{\"inputs\": [[{intYear}.0, {intYear - 1915}.0]]}}";

var request = (HttpWebRequest)WebRequest.Create(TensorFlowML.RegressionEndpoint);

request.Method = "POST";

request.ContentType = "application/json";

request.ContentLength = data.Length;

using (var webStream = await request.GetRequestStreamAsync())

{

using (var requestWriter = new StreamWriter(webStream, System.Text.Encoding.ASCII))

{

await requestWriter.WriteAsync(data);

}

}

var webResponse = await request.GetResponseAsync();

using (Stream webStream = webResponse.GetResponseStream() ?? Stream.Null)

{

using (StreamReader responseReader = new StreamReader(webStream))

{

returnValue.Result = responseReader.ReadToEnd();

}

}

return returnValue;

* Run the project and try entering 1922.
* Try other years like 1915 or 2050

Notice that when we get to the start of our model years like 1915 don’t return the right value. In fact, the model has no concept of when the stock market began so it will return negative numbers. Our model isn’t perfect and for things that have a finite beginning, reasonableness checks may need to be added. Is this model accurate enough? Likely not for something like the Dow, so much more needs to go into such a prediction. Understanding the question of, “is this accurate enough” is a foundational question for machine learning.