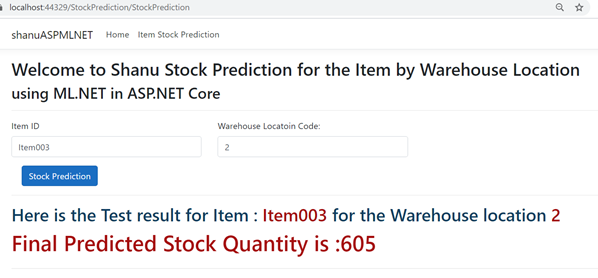
Getting Started with ML.NET Model Builder for Using in ASP.NET Core

[Try Free File Format APIs for Word/Excel/PDF](https://www.c-sharpcorner.com/AdRedirector.aspx?AdId=133&target=https://www.aspose.com/)

[**SourceCode.zip**](javascript:void(0);)



## **Introduction**

ML.NET Model builder makes our work easier while developing an ML.NET application. It helps build, train and deploy the models more easily for all kinds of ML .NET supporting models, as well us helps us develop the custom models. If you are new to ML.NET, please read my previous articles related to ML.NET. The latest version of ML.NET is also established to develop easy custom ML using AutoML and with ML.NET Model builders.

Before getting started with the ML.NET, let's understand the basic concept of ML.NET to develop our machine learning applications.

**Load Data**

For the perfect prediction of results, we need to give a lot of data to train the model. In ML.NET, we can give the data for both training and testing with a Text (CSV/TSV, Relational Database.

**Train**

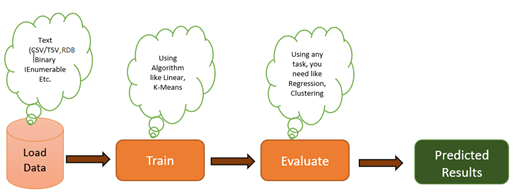
We need to select the right algorithm to train the model. Depending on our need, we need to pick the correct algorithm to train and predict the results.

**Evaluate**

Select the Machine learning type for our model training and prediction. If you need to work with a segment, then you can select the Clustering model. If you need to find the price of stock prediction, you can select Regression, and if you need to find the sentiment analysis, then can select the Classification model.

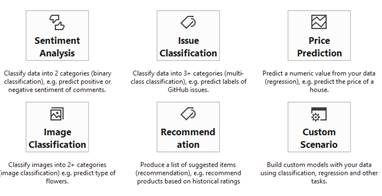
**Predicted Results**

Based on the train and test data with the trained model, the final prediction will be displayed using the ML.NET application. The trained model will be saved as the binary format which can also be integrated with our other .NET applications.



While using ML.NET model builder, it's easy to load the data either from CSV, Excel or from the Database. Select and train our preferred available ML.NET models, or go with custom model training. Evaluate the model and finally predict and display the trained model result to any of our .NET applications. Now in this article, let's see trained and predicted results. We will be displaying results in our ASP.NET Core application.

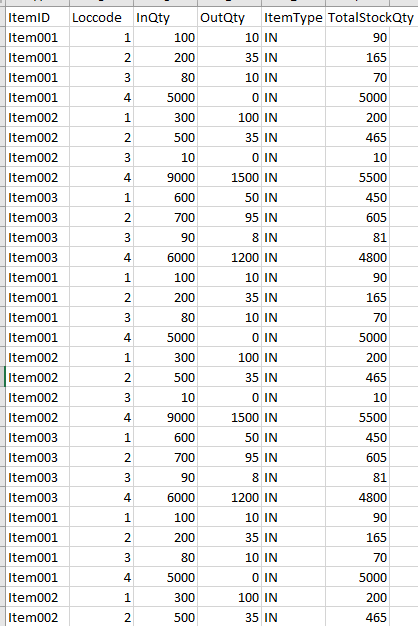
In ML.NET Mode builder we can see the following scenario. (Note: Microsoft can add more scenario, as it might be different from each version)



In this sample program, we will be using ML.NET Model builder to predict the Item Stock using the Custom Scenario.

In my [previous article](https://www.codeproject.com/Articles/1256360/Getting-Started-with-Machine-Learning-DotNet-ML-NE) that I wrote on April 2018, I explained the same Item Stock Prediction using the console application. I used the same Item Stock Data as CSV file and used the data to train the model and predict the result. Finally, I bound the result to the ASP.NET Core application.

The item stock data will look like this. Here, I have the columns Item Code, Location Code, Warehouse in Quantity, Warehouse Out Quantity, Item Type and finally Remaining Item total Stock quantity of each item in each location. We will use the below dataset to train, evaluate, and predict the result using the ML.NET Model builder.



**Prerequisites**

[Visual Studio 2017 15.9.12 or later](https://visualstudio.microsoft.com/vs/)

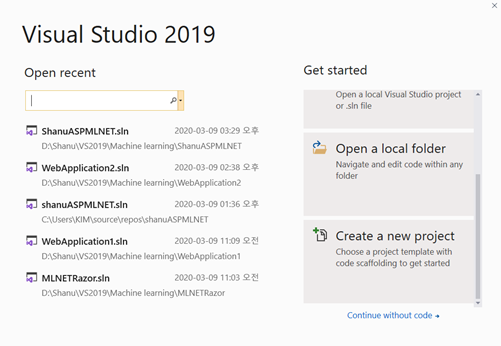
[ML.NET Model Builder](https://marketplace.visualstudio.com/items?itemName=MLNET.07)

**Code Part**

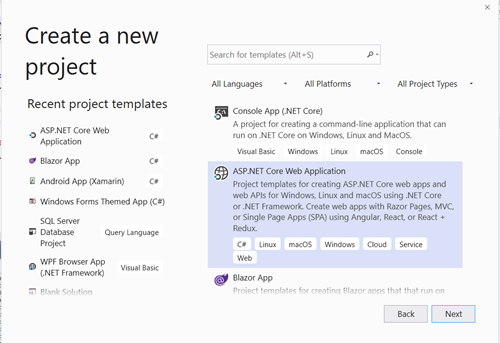
**Step 1 - Create an ASP.NET Core Application**

(Note: I used Visual Studio 2019)

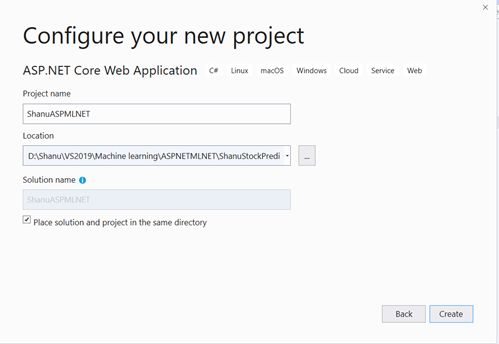
After installing the prerequisites, click Start >> Programs >> Visual Studio 2019 >> Visual Studio 2019 on your desktop. >> Click Create a Project. Click continue.



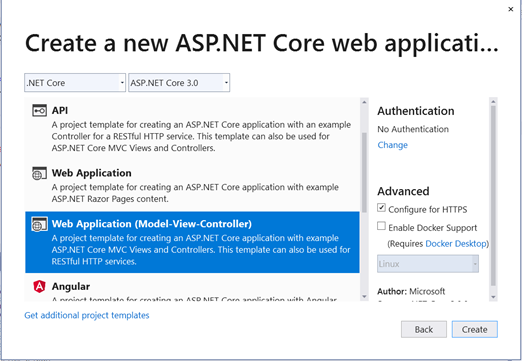
Select ASP.NET Core Web Application and click Next.



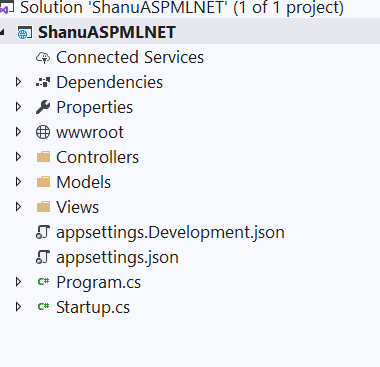
Select the Project Folder and check the Place solutions in the same directory and on the create button.



Now select Web Application (Model-View-Controller) and click on the Create button.

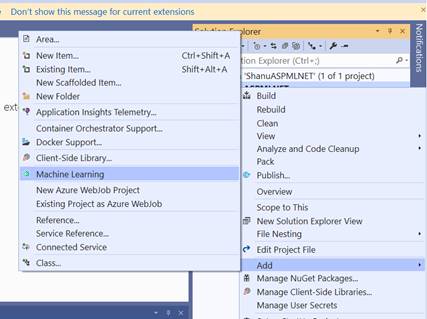


We can see our ASP.NET Project has been created with Controllers/Models and Views.

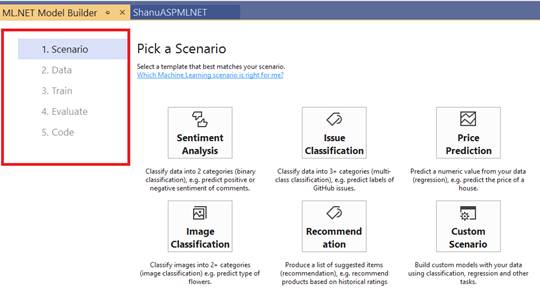


**Step 2 - Working with ML.NET Model Builder**

Right click on the project, select Add and click on Machine Learning.



We can see the ML.NET Model Builder



In ML.NET Model Builder, on the  left side we can see the menu, which displays Scenario, Data, Train, Evaluate and Code

**Scenario**

From the Scenario, you can see the available ML.NET Models for the development example like Binary classification, Image Classification, etc. Also, you can see the Custom Scenario in order to build your own custom models. In our demo, we will be using the custom model.

**Data**

You can load the data for training from the file or from the Database.

**Train**

Select the model to be trained

**Evaluate**

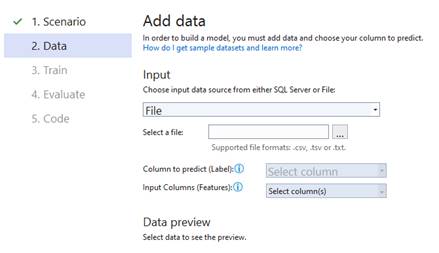
The evaluation will be performed after the model training is completed. While evaluating, the trained model will be compared with the test data to compare and predict the final result to be produced.

**Code**

Finally, the code will be used to create the trained model and use it in any application. We will be using the final trained and evaluated model code for our ASP.NET Core application to predict and display the result for Total Item stock Quantity available per location.

**Custom Scenario**

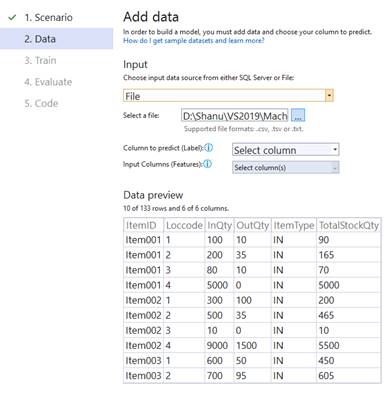
Here, we have selected the Custom scenario of our demo application in order to predict the Stock quantity.



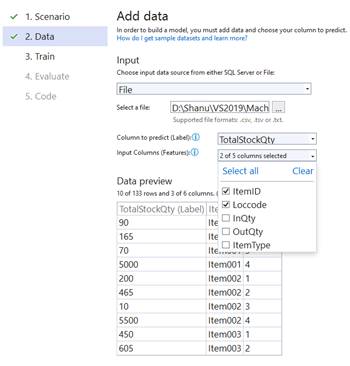
**Data**

After selecting our Scenario, we can see the next menu will be displayed. Here, we have selected the Custom Scenario. Next, the Data menu has been displayed. From here, we have the option to load the data to be trained from the file or the Database. Here we will be using the file to load the data.

Click on the Select a file and load the file. We can see here that we have loaded the file to be trained.



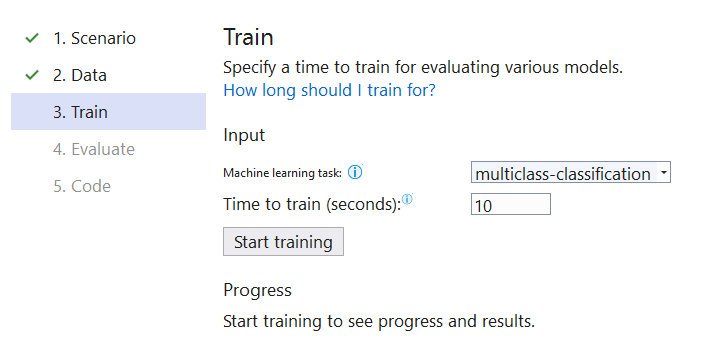
Next, we need to select the column to predict (In Machine Learning, the prediction column is called the Label). In our demo, we are going to predict the TotalStockQty. Also, we need to select the Input column given by us to predict the result. For our demo, we need to predict the Stock Qty based on the Item and by the Location. Therefore we have selected the Input (Feature) columns as ItemID and Loccode.



**Train**

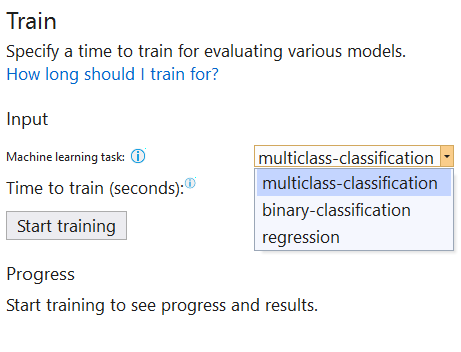
After that, click on the train button to train the model with the loaded data.

In the train screen, we can see the Machine Learning tasks to be selected for training our model. We also can see the time to train the model in seconds to be selected.



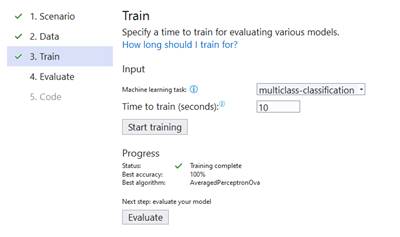
**Machine Learning Task**

We can see that in the custom scenario, the task is available for ML.NET is multiclass-classification, binary-classification, and regression. Here for our demo, we selected multiclass-classification.



Our file size is less than 10mb data size, so we can select approximately 10 seconds to train the model.

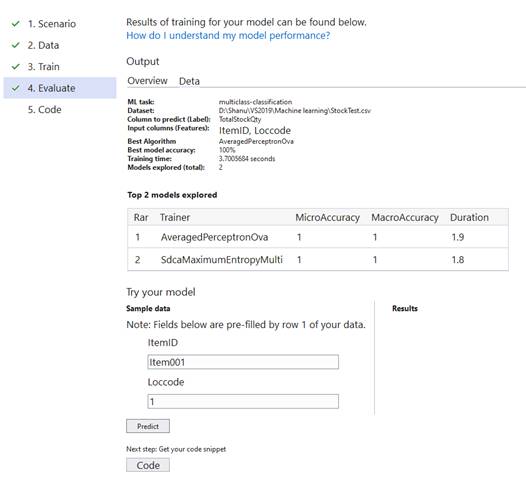
After selecting the Machine learning task and training time, click on the Start training button and wait for few seconds to complete the training.



Now we can see the training is completed and the best algorithm has been used. Next, we need to click on the Evaluate button to evaluate our training model.

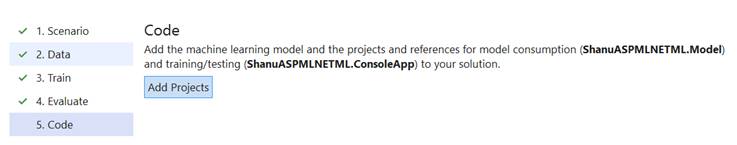
**Evaluate**

In the evaluation screen, we can evaluate the trained model and finally click on the code button to get the code.

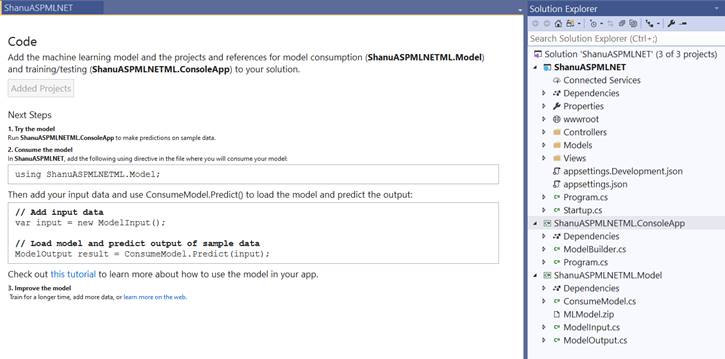


**Code**

We can see that with the ML.NET Model, training and testing project can be added to our solutions. We will add this model to our ASP.NET Code project solution. For that, we need to click on the Add to Projects button.

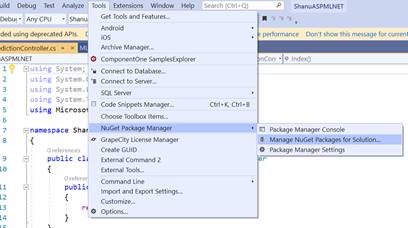


We can see both Model and the console project have been added to our ASP.NET Core solution. Now let’s see how to import the model to our ASP.NET core and display the predicted result to the users.

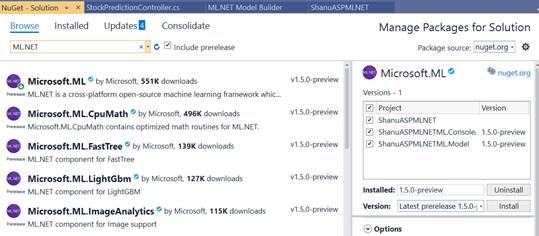


**Step 3 - Add the ML.NET to ASP.Net Core project for prediction**

Select Tools from the menu, and select NuGet Package Manager, then click on Manage NuGet Packages for the Solution.



Select Browse and enter ML.NET in search. Select Microsoft.ML. Also, on the right side, we can see our ML.NET model projects displayed. Select the project and click on Install.



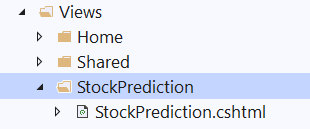
**Step 4 - Create new ASP.NET Core Controller**

Right click on the Controller Class and click New Empty control. Give the controller name StockPrediction and add the controller to the controller folder.



**Step 5 - Create new ASP.NET Core View**

Next, create a new folder inside the View folder as StockPrediction. Also, add a view with the name StockPrediction.cshtml



**Controller Code**

In the StockPredictionController Controller add the below code:

In the import part, add the model

1. **using** ShanuASPMLNETML.Model;

After importing that, add the below code. Here, we change the default view with our new View name. Also, we have created one more post Action result and returned the predicted result to the view to bind the results.

1. **public** **class** StockPredictionController : Controller
2. {
3. [HttpGet]
4. **public** IActionResult StockPrediction()
5. {
6. **return** View();
7. }
9. [HttpPost]
10. **public** ActionResult StockPrediction(ModelInput input)
11. {
12. ViewBag.Result = "";
13. var stockPredictions = ConsumeModel.Predict(input);
14. ViewBag.Result = stockPredictions;
16. ViewData["ItemID"] = input.ItemID;
17. ViewData["Loccode"] = input.Loccode;
18. **return** View();
20. }
21. }

**View Code**

In the StockPrediction View, add the below code. Here, we added 2 textboxes, one for the item Code input and one for the Location code input. We have set the same name as our model name so that the data will be passed to our Trained model and return the predicted results. Finally, we bind the predicted results to the users.

1. @model ShanuASPMLNETML.Model.ModelInput
2. @{
3. ViewData["Title"] = "item Stock Predicti0n";
4. Layout = "~/Views/Shared/\_Layout.cshtml";
5. }
7. <h2>Welcome to Shanu Stock Prediction **for** the Item by Warehouse Location</h2>
8. <h3>using ML.NET **in** ASP.NET Core</h3>
9. <hr />

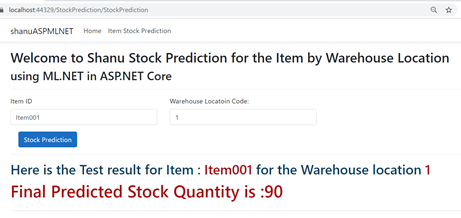
12. <div **class**="row">
13. <div **class**="col-md-12">
14. <form asp-action="StockPrediction">
15. <div **class**="row">
16. <div **class**="form-group col-md-4">
17. <label asp-**for**="ItemID" **class**="control-label">Item ID</label>
18. <input asp-**for**="ItemID" **class**="form-control" />
19. </div>
20. <div **class**="form-group col-md-4">
21. <label asp-**for**="Loccode" **class**="control-label">Warehouse Locatoin Code:</label>
22. <input asp-**for**="Loccode" **class**="form-control" />
23. </div>
25. </div>
26. <div **class**="form-group col-md-4 text-left">
27. <label **class**="control-label"></label>
28. <input type="submit" value="Stock Prediction" **class**="btn btn-primary" />
29. </div>
30. </form>
31. <hr />
32. </div>
34. </div>

37. @**if** (ViewBag.Result != **null**)
38. {
39. <div **class**="row">
40. <div **class**="col-md-12">
41. <h2 style="color:#103f62">Here is the Test result **for** Item : <span style="color:#a50f0f"> @ViewData["ItemID"]  </span> **for** the Warehouse location <span style="color:#a50f0f">@ViewData["Loccode"] </span>  </h2>
42. <h1 style="color:#a50f0f">Final Predicted Stock Quantity is :@ViewBag.Result.Prediction  </h1>
43. </div>
44. </div>
45. <hr />
46. }

**Build and run the project**

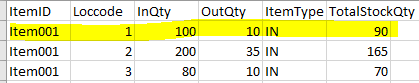
We can enter the Item code and Location code to be predicted and display the Stock Quantity.

Here, we have enter the Item code as Item001 and Location code as 1.



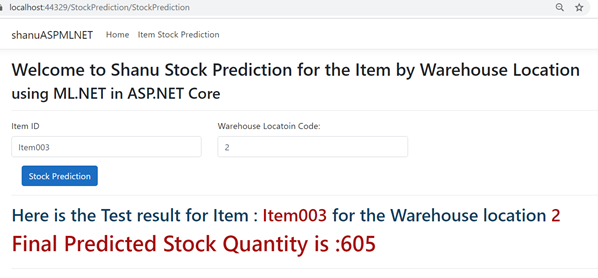
We can see the Predicted result as :90

In the Excel file, we can also cross check the result or our original data.

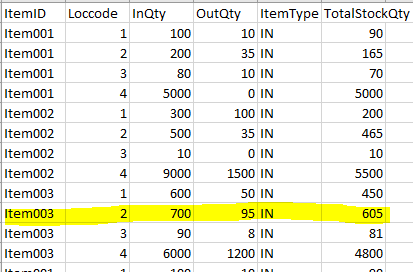


Same as above, we test for one more data for prediction.

Here, we enter the Item code as Item003 and Location code as 2.



The Excel file data for the Item003 is added in the file as shown below.



From both results, we can find that the prediction is completed 100%, and the result is as expected.

## **Conclusion**

ML.NET (Machine Learning DotNet) is a great framework for all the .NET lovers who are all looking to work with machine learning. Now, only the preview version of ML.NET is available and I can’t wait until the release of the public version. If you are .NET lovers, unaware of Machine Learning and are looking forward to working with machine learning, then ML.Net is for you all. It's a great framework to get started with ML.NET. Hopefully, you all enjoyed reading this article and see you all soon with another post1