## Getting Started with the Tabular Object Model (TOM)

The purpose of this hands-on tutorial is to provide campers with a guide how to get started using the Tabular Object Model (TOM) to program against data models created using Power BI Desktop. This lab will get you started with Visual Studio Code and the latest version of .NET known as .NET 5 that was released in November of 2020.

There lab exercises were derived from an awesome set of blog posts by Phil Seamark ([post1](https://dax.tips/2020/07/09/using-visual-studio-code-with-power-bi/), [post2](https://dax.tips/2020/07/30/automatically-create-measures-in-power-bi-using-vs-code/), [post3](https://dax.tips/2020/08/24/using-visual-studio-code-to-query-power-bi/) and [post4](https://dax.tips/2020/10/19/use-vs-code-to-create-power-bi-external-tool/)).

### Setup: Install Visual Studio Code and the .NET 5 SDK

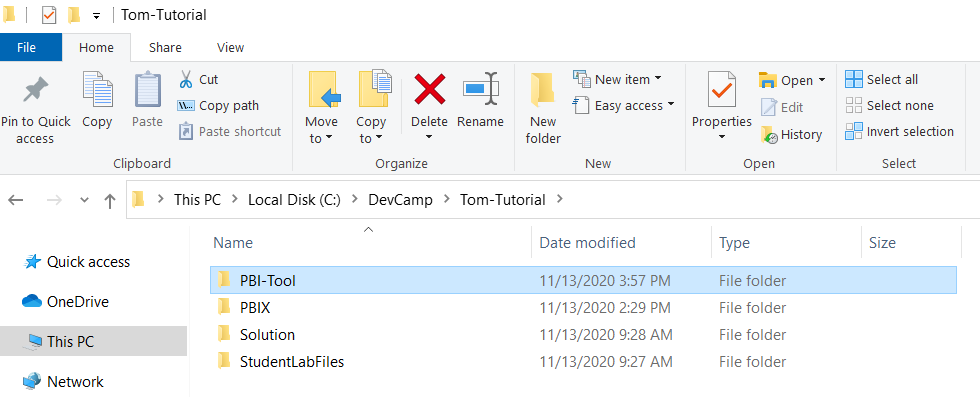
In this initial setup, you will download install the .NET 5 SDK and Visual Studio Code if you haven't done so already.

1. Install the .NET 5 SDK.
   1. Download and run the installer: <https://dotnet.microsoft.com/download/dotnet/thank-you/sdk-5.0.100-windows-x64-installer>
   2. Here is more info on .NET 5 downloads if you need it: <https://dotnet.microsoft.com/download>
2. Install Visual Studio Code
   1. Open a browser and navigate to [code.visualstudio.com](https://code.visualstudio.com/)
   2. download and run the installer for the current version for Windows.
   3. Use default settings when prompted during the install.

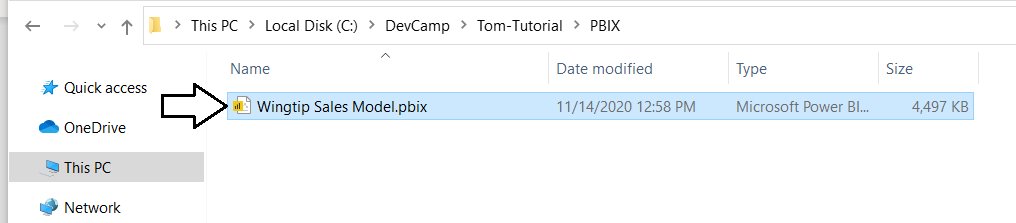
### Exercise 1: Connect to Power BI Desktop using the Tabular Object Model

In this exercise, you will create a new .NET console application and connect to a data model running in Power BI Desktop.

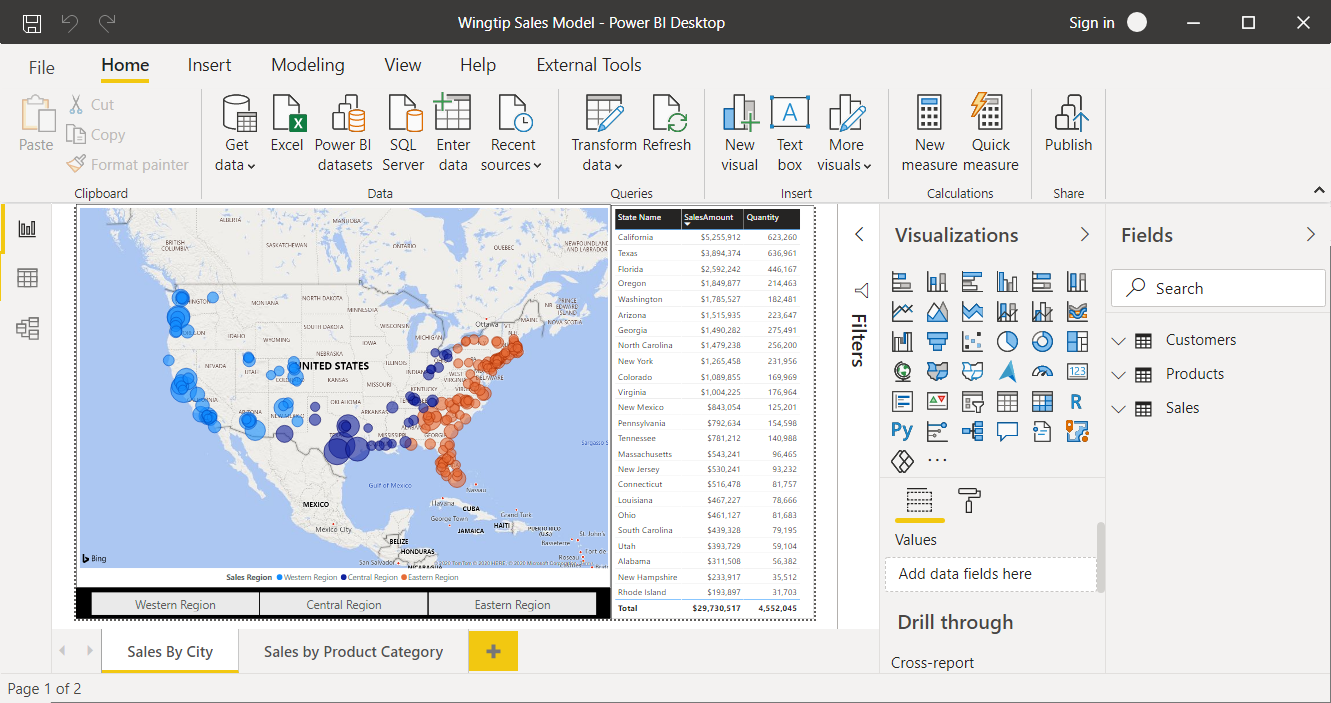
1. Create a new folder for the tutorial.
   1. Create a new folder on your local hard drive named **Tom-Tutorial**.
   2. Copy student files [use different screnshot].



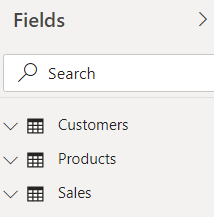
1. Launch Power BI Desktop project and open the project named **Wingtip Sales.pbix**.
   1. Inside the **PBIX** folder, locate the project file named **Wingtip Sales.pbix** and open it in Power BI Desktop



* 1. You should now see report and data model for **Wingtip Sales Model.pbix** loaded in Power BI Desktop.

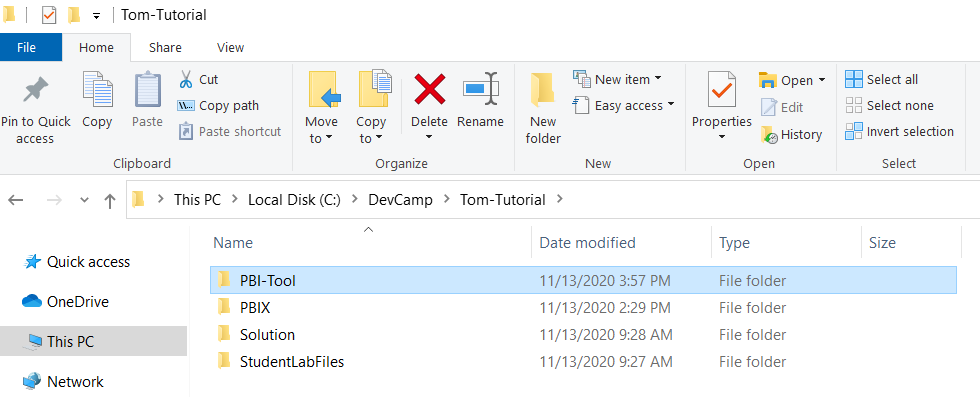


* 1. Quickly review the four tables named **Customers, Products** and **Sales** that are visible in the **Fields** list

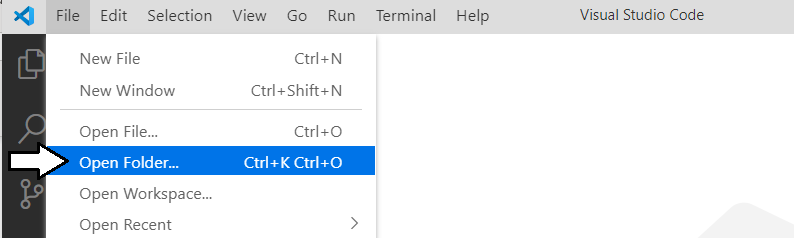


Leave this project open in Power BI desktop as you move on to the next step. Later in this exercises, you will connect to the data model of the **Wingtip Sales** **Model** project using the Tabular Object Model by connecting through a localhost address.

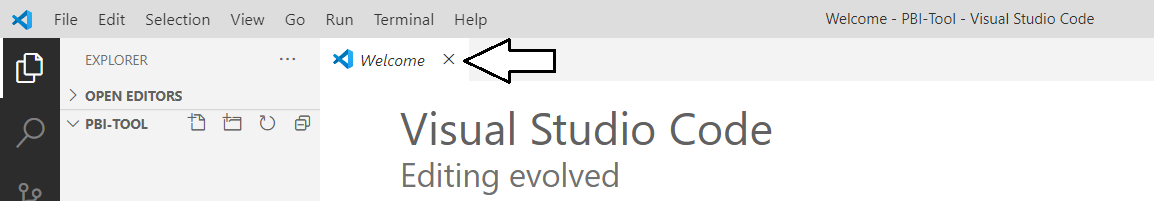
1. Create a new folder for your new project and name it **PBI-Tool**.
   1. Create a child inside the **Tom-Tutorial** folder named **PBI-Tool**.



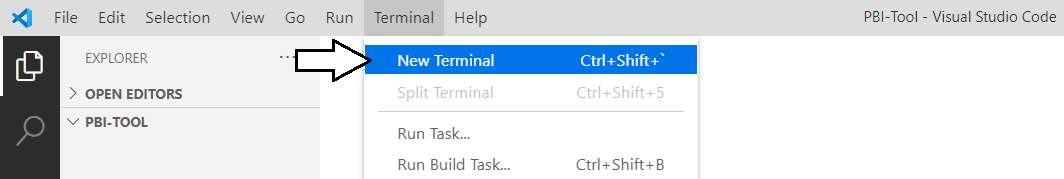
1. Launch Visual Studio Code and open the **PBI-Tool** folder.
   1. Launch Visual Studio Code and use the **Open Folder…** command to open the **PBI-Tool** folder created in the previous step.



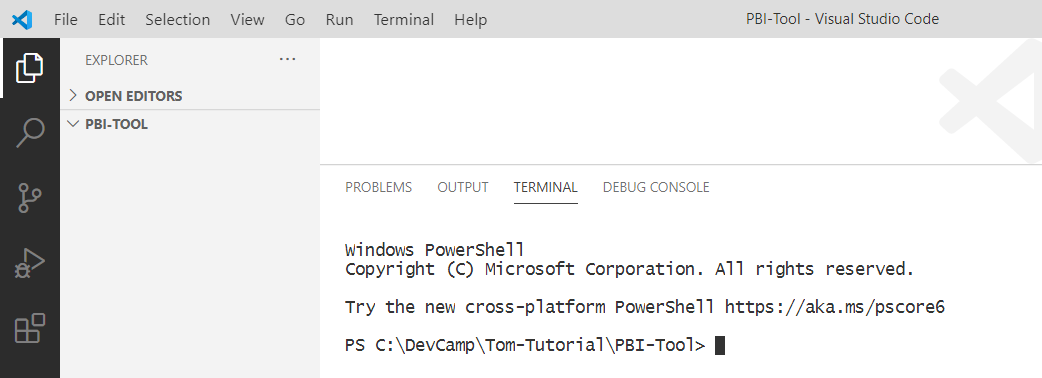
* 1. Once you have open the **PBI-Tool** folder, close the Welcome page.



1. Use the Terminal console to verify the current version of .NET
   1. Use the Terminal > New Terminal command of the [**Ctrl+Shift+`**] command to open the Terminal console..



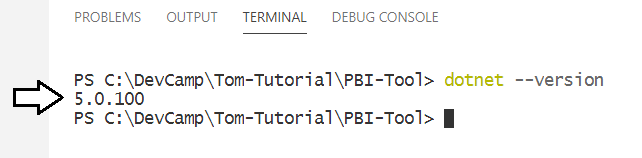
* 1. You should now see a Terminal console with a cursor where you can type in and execute command-line instructions.



* 1. Type the following **dotnet** command-line instruction into the console and press **Enter** to execute it.

dotnet –version

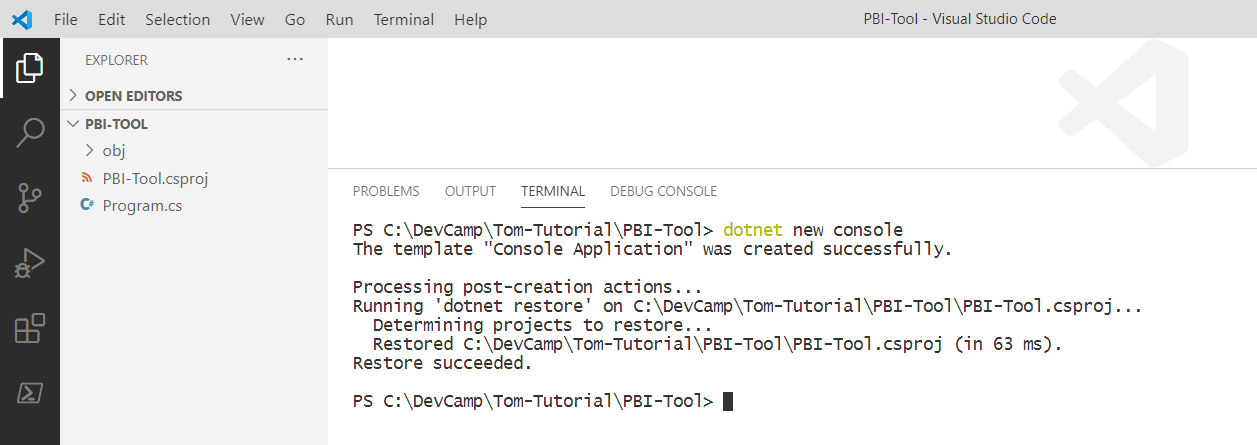
* 1. When you run the command, the **dotnet** CLI should respond by display the .NET version number.



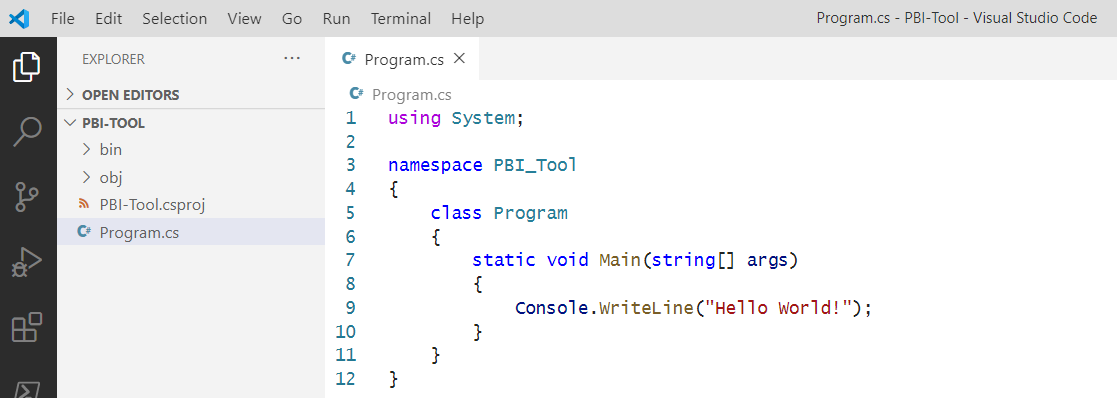
1. Use the dotnet CLI to create a new .NET console application in the **PBI-Tool** folder..
   1. In the Terminal console, type and execute the following command to generate a new .NET console application.

dotnet new console

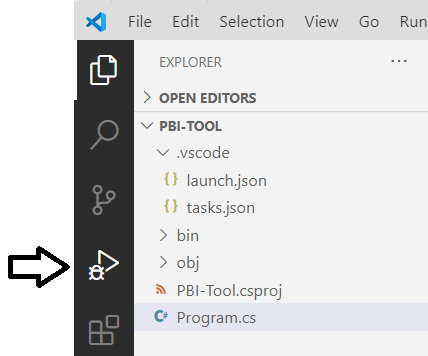
* 1. After running the **dotnet newt console** command, you should see new files have been added to the project.



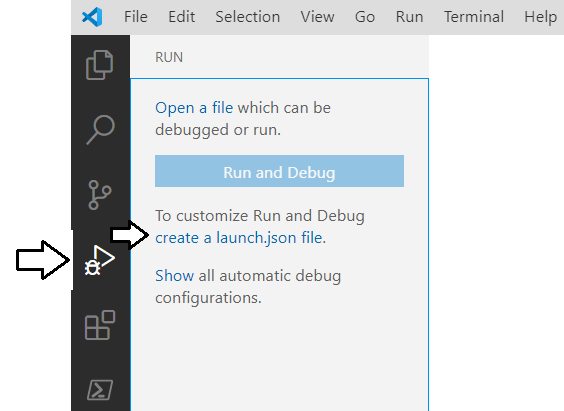
* 1. Open the **Program.cs** file and inspect the C# starter code inside which displays the tradition **Hello World!** greeting.



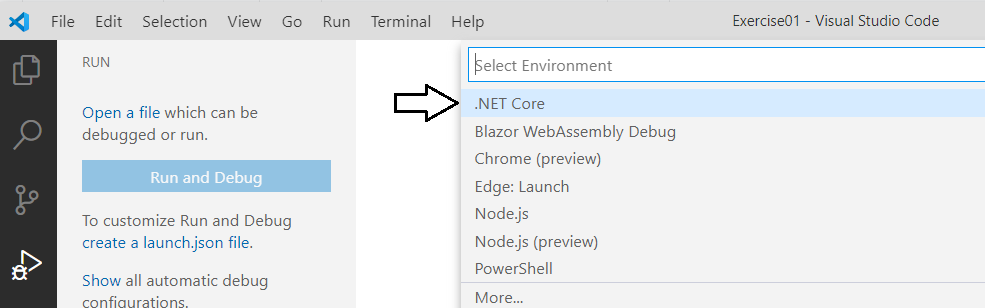
1. Add support for running the new console project in the .NET debugger.
   1. Using the left navigation in Visual Studio Code, move the Run/Debug view.



* 1. Click the **create a launch.json file** link.

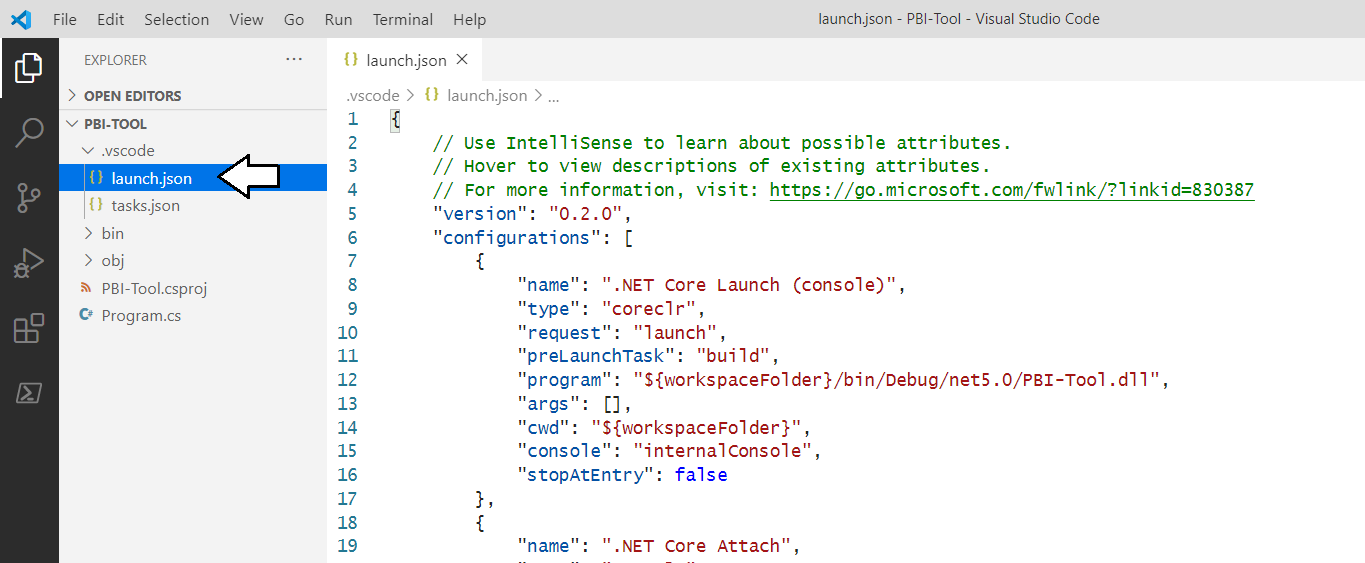


* 1. When prompted to **Select Environment**, select **.NET Core**.



When you select an environment, Visual Studio Code responds by generating two files named **launch.json** and **tasks.json**.

* 1. Examine the contents of the **launch.json** file.



* 1. Inside **launch.json**, locate the following line of code.

"stopAtEntry": false

* 1. Add a comma at the end of this line and then add another line of code below as shown in the following listing.

"stopAtEntry": false ,

"logging": { "moduleLoad": false }

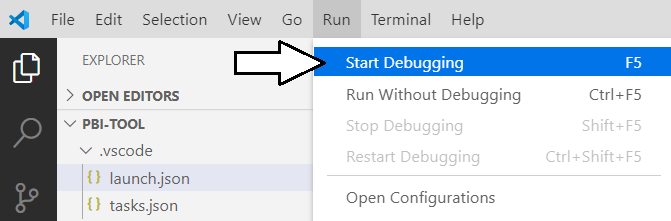
* 1. The **launch.json** file in your project should now match the following screenshot.



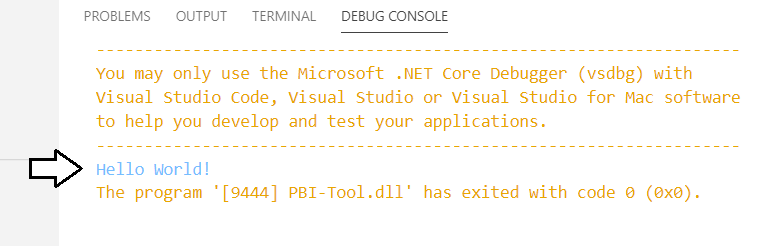
* 1. Save your changes and close **launch.json**.

The reason for disabling **moduleLoad** is that it will eliminate unnecessary messages shown in the console during debugging sessions.

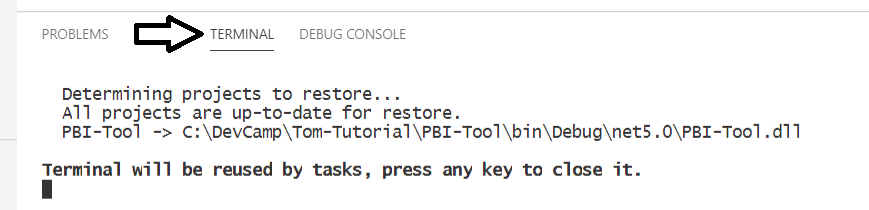
1. Run a debugging session to test the application and see the **Hello World!** Message.
   1. Run a debugging session by running the **Run > Start Debugging** command or pressing the **{F5}** key.



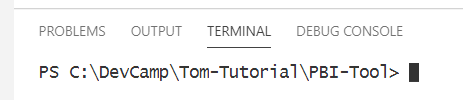
* 1. The application should run and display **Hello World!** In the debug console window as shown in the following screenshot.



* 1. Navigate to the **TERMINAL** console and press **ENTER** to stop the debugger.



* 1. You have now run and terminated a simple debugging session.



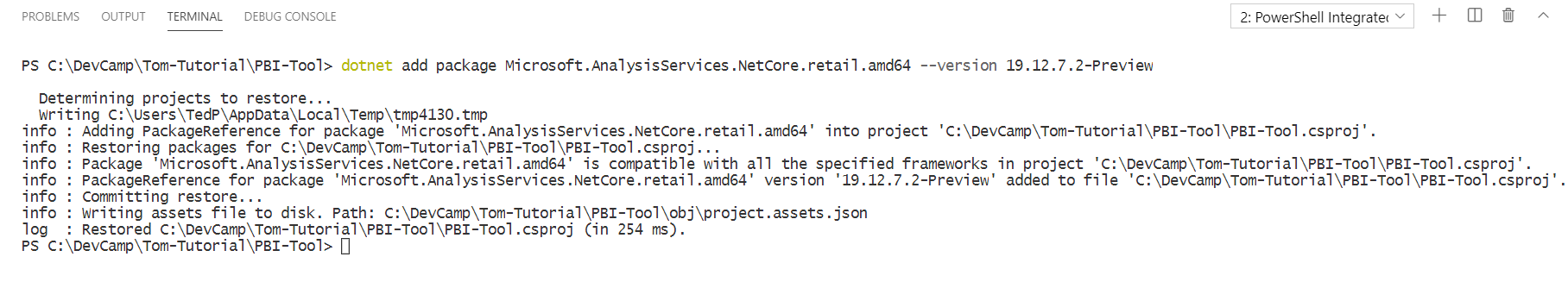
OK, you've now gotten 'hello world' out of the way. Remember that a journey of a thousand miles begins with a single step!

1. Add a **AMO** NuGet package to your project using the so you can program against the Tabular Object Model.
   1. Return to the Terminal console.
   2. Type and execute the following **dotnet add package** command to add the **AMO** NuGet package

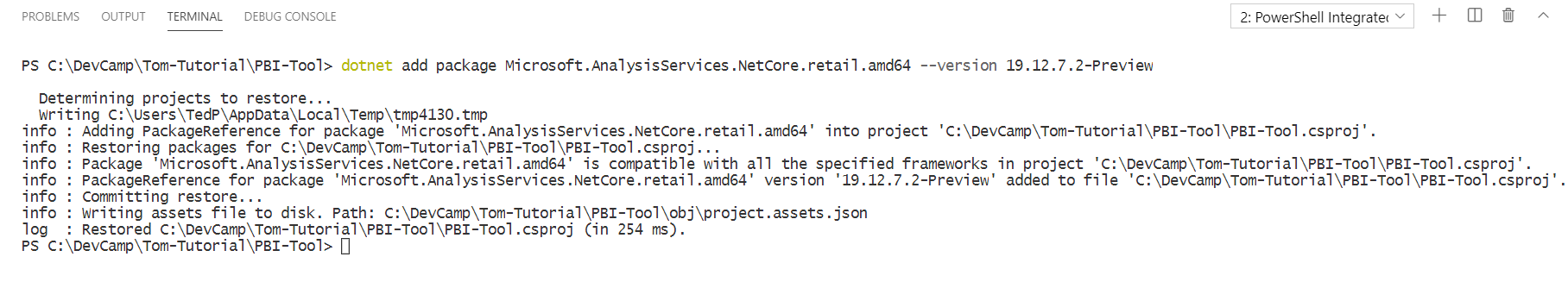
dotnet add package Microsoft.AnalysisServices.NetCore.retail.amd64 --version 19.12.7.2-Preview

At the time that lab exercise was written, the latest version of this package was **19.12.7.2-Preview**. You can check out the information at the following URL to determine the latest release: <https://www.nuget.org/packages/Microsoft.AnalysisServices.NetCore.retail.amd64>

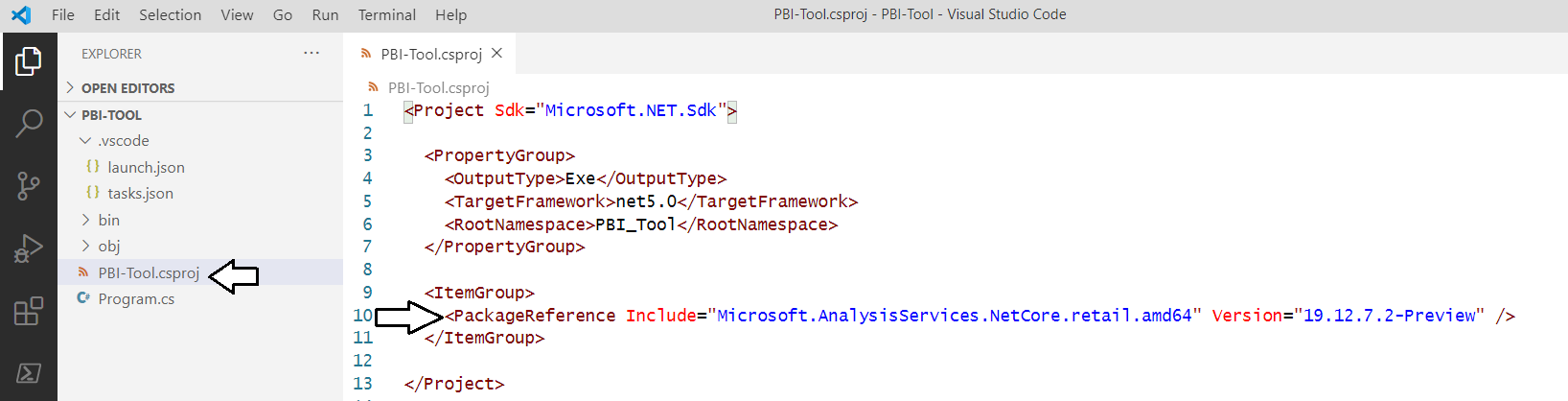
* 1. Once you have typed the following command into the **Terminal** console, press **Enter** to execute it.



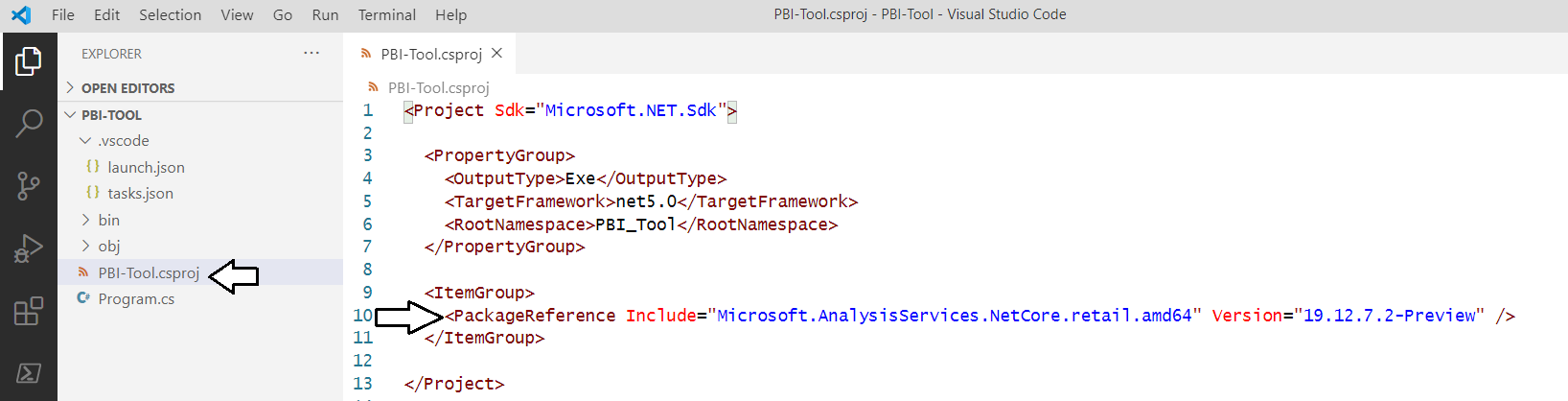
* 1. The **dotnet add package** command should run without any errors.



* 1. After the **dotnet add package** command has completed, open the project file named **PBI-Tool.csproj**.



* 1. You should see that the **PBI-Tool.csproj** file now contains a **PackageRefence** element to track the newly installed package.



* 1. Close **PBI-Tool.csproj** without saving any changes.

Now that you have added this NuGet package, you can program against the AMO library which includes the Tabular Object Model.

1. Modify the C# code in **Program.cs**.
   1. Open **Program.cs**.
   2. Remove all the existing code in **Program.cs** and replace with the following code.

using System;

using Microsoft.AnalysisServices.Tabular;

class Program {

const string connectString = "localhost:50000"; // update with port number on your machine

static void Main(string[] args) {

Server server = new Server();

server.Connect(connectString);

Model model = server.Databases[0].Model;

foreach(Table table in model.Tables) {

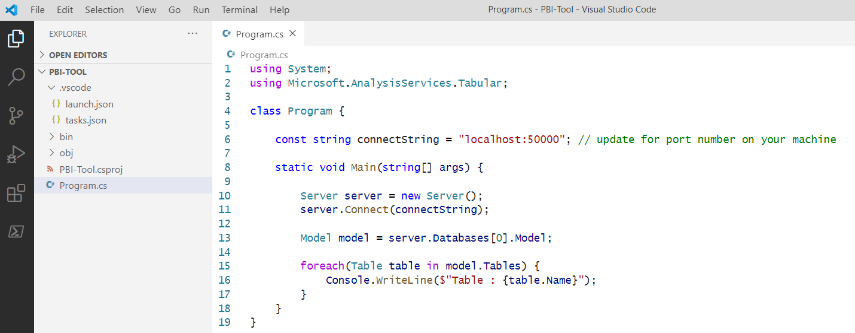
Console.WriteLine($"Table : {table.Name}");

}

}

}

* 1. The **Program.cs** file in your project should now match the following screenshot.

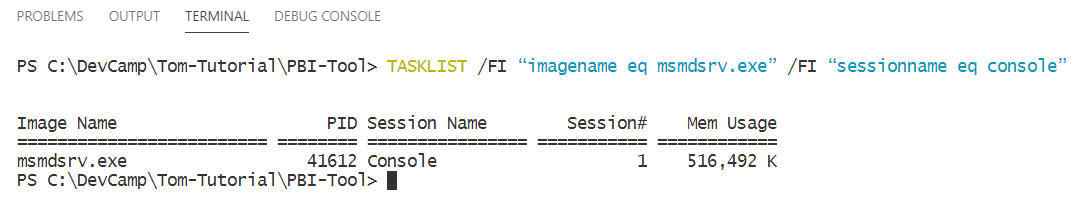


There is one more important thing you need to do before you can run this program. You must determine the port number that Power BI Desktop is using to run the project you opened named **Wingtip Sales.pbix**. The issue is that Power BI Desktop selects a random port number whenever you open a PBIX project file. Once you must determine what the port number is on your machine, you can use it to modify the **connectString** which will allow you to connect to the data model of the **Wingtip Sales.pbix** project.

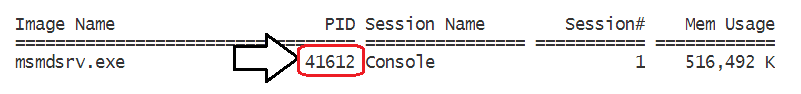
1. Determine the port number used by Power BI Desktop using command-line instructions.
   1. Return the Terminal console. Type in and execute the following **TASKLIST** command.

TASKLIST /FI “imagename eq msmdsrv.exe” /FI “sessionname eq console”

* 1. When you run the **TASKLIST** command, it displays its output in a table format



* 1. Determine the PID (i.e. Process ID) for the image named **msdsrv.exe**.

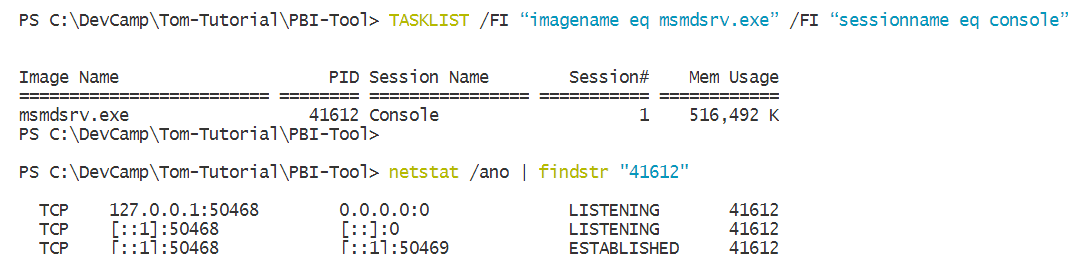


You're only half way there at this point. Now you have the process ID (PID), you will use that to find the port number.

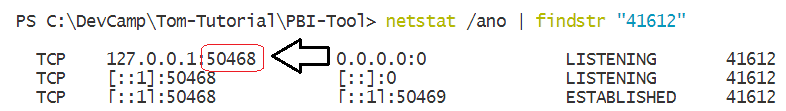
* 1. Type in the following **netstat** command and replace the PID in quotes with the one you discovered on your computer.

netstat /ano | findstr "41612"

* 1. Execute the **netstat** command and examine its output.

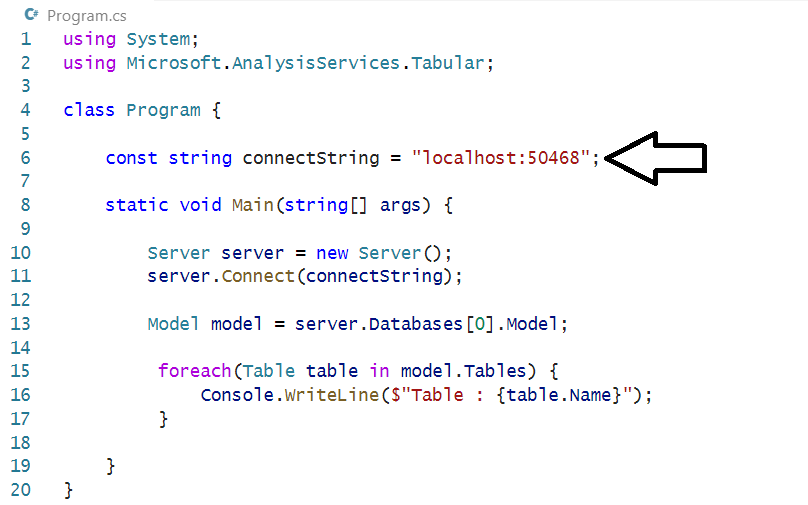


* 1. You should be able to discover the port being used by Power BI Desktop as shown in the following screenshot.



Using the **TASKLIST** command and the **netstat** command is just one way to determine the Power BI Desktop port number. You can also find the current port number using tools such as DAX Studio and the Tabular Editor. Check out [this blog post](https://www.biinsight.com/four-different-ways-to-find-your-power-bi-desktop-local-port-number/) to learn more.

1. Run the program and connect to Power BI Desktop.
   1. Open **Program.cs** and update the **connectString** constant with the correct port number for your computer.

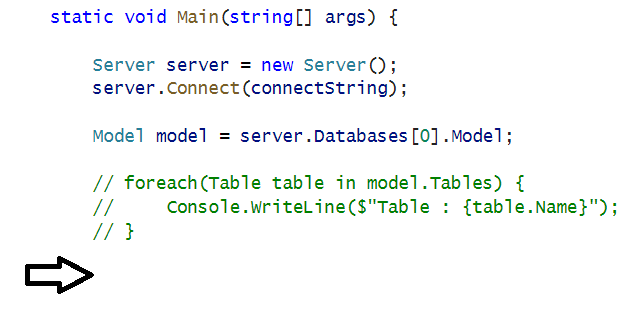


* 1. Run a debugging session by running the **Run > Start Debugging** command or pressing the **{F5}** key.
  2. When the program runs, it enumerates through the tables in the data model and displays their names in the Debug Console



Congratulations. You can now say you that have programmed using the Tabular Object Model. Your friends will be so jealous!

1. Write code to add a new measure to the Sales table.
   1. In **Program.cs**, comment out the lines of code for the **foreach** loop and place the cursor below to add more code.



* 1. Add the following code at the end of the **Main** function.

Table table = model.Tables["Sales"];

if (table.Measures.ContainsName("VS Code Measure")) {

Measure measure = table.Measures["VS Code Measure"];

measure.Expression = "\"Hello Again World\"";

}

else {

Measure measure = new Measure() {

Name = "VS Code Measure",

Expression = "\"Hello World\""

};

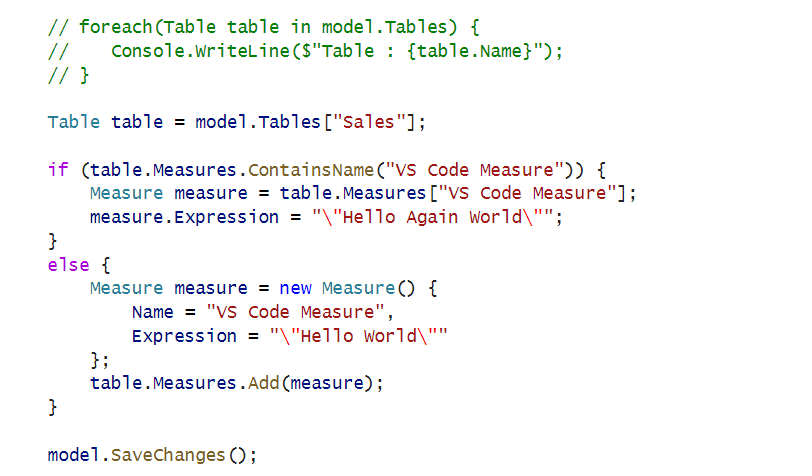
table.Measures.Add(measure);

}

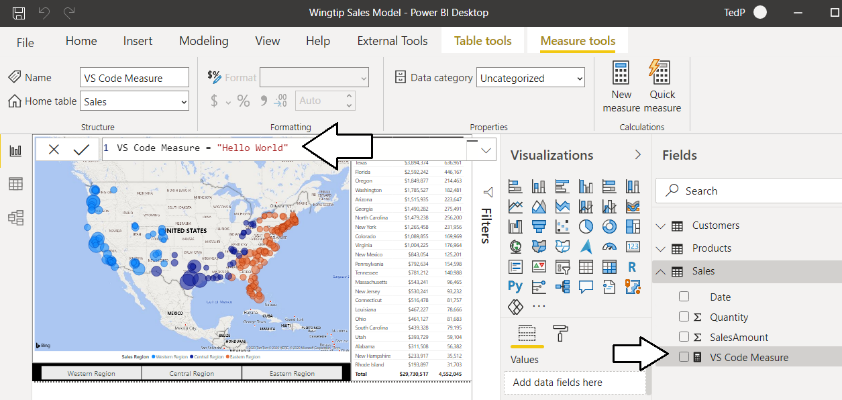
model.SaveChanges();

The first time you run this code, it will add a new measure to the **Sales** table named **VS Code Measure**. The second time you run this code, it will determine that a measure named **VS Code Measure** already exists and it will modify the measure's expression. The main point of this code is to demonstrate that you can use TOM to create new measures and to modify existing measures.

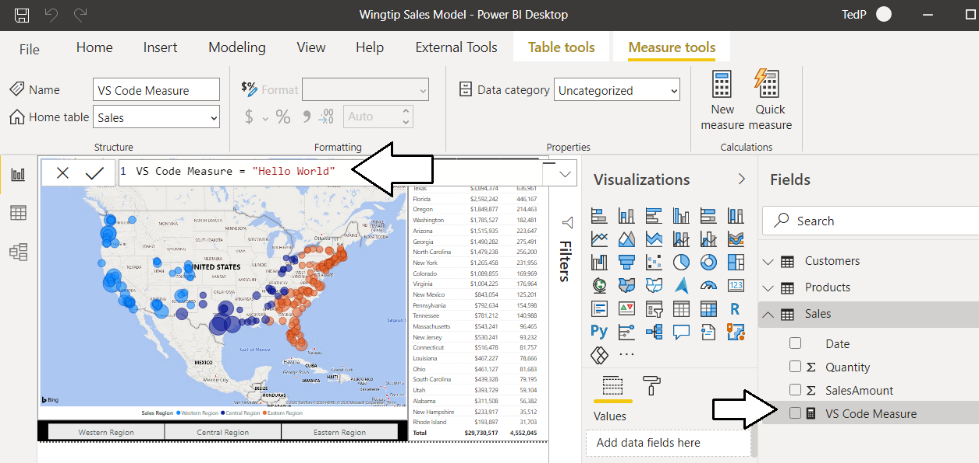
* 1. The code at the bottom of the **Main** function in **Program.cs** should now match the following screenshot.



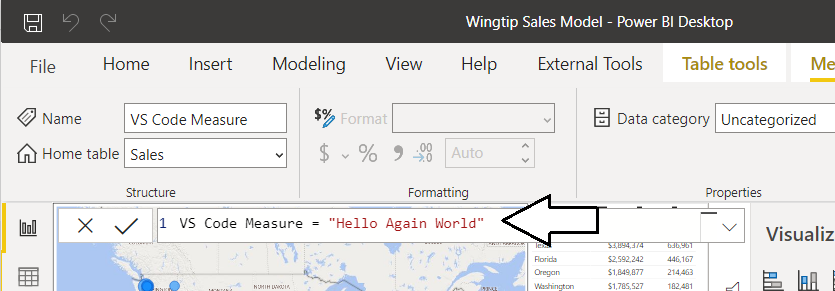
* 1. Save your changes to **Program.cs**.
  2. Run a debugging session by running the **Run > Start Debugging** command or pressing the **{F5}** key.
  3. Return to Power BI Desktop. The **Sales** table should contain **VS Code Measure** with the expression *"Hello World"*.



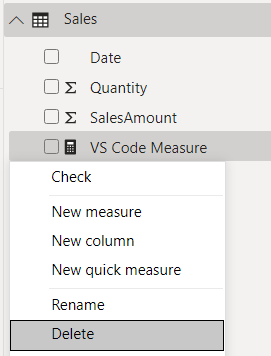
* 1. Inspect the expression for **VS Code Measure**. The should expression should be a simple string "Hello World".



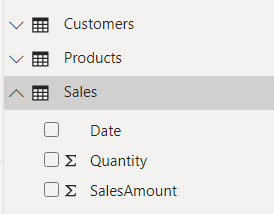
* 1. Return to Visual Studio Code.
  2. Run a second debugging session by running the **Run > Start Debugging** command or pressing the **{F5}** key.
  3. Return to Power BI Desktop and verify that **VS Code Measure** now has an updated expression *"Hello Again World"*.



* 1. Delete the measure named **VS Code Measure** by right-clicking it in the **Fields** list and select the **Delete** command.



* 1. The **Sales** table should no longer contain a measure named **VS Code Measure**.

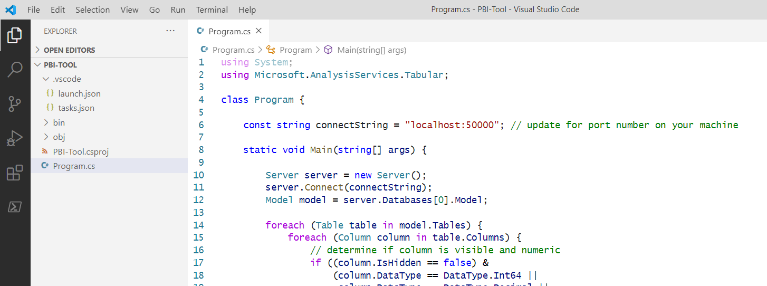


You have used the Tabular Object Model to connect to a Power BI Desktop project and perform some simple read and write operations. Now it's time to move ahead and write code using the Tabular Object Model that performs real-world data modeling tasks.

### Exercise 2: Writing TOM Code to Add Auto Measures to Tables

In this exercise, you will modify the PBI-Tool console application automatically add measures to tables which have numeric columns that are not hidden from report view. You will test code that iterates through every **Table** object in the model and then iterates through the **Column** objects for each **Table** object. The code will automatically add a new measure each time it find a numeric column which has an **IsHidden** property value of false.

1. Copy and paste the Exercise 2 starter code into **Program.cs**.
   1. Open **Program.cs** and delete all the code that is currently inside.
   2. Copy the Exercise 2 starter code from the [Exercise02-Program.cs](https://raw.githubusercontent.com/PowerBiDevCamp/Tabular-Object-Model-Tutorial/main/StarterFiles/Exercise02-Program.cs) file in the **StarterFiles** folder to the Windows clipboard.
   3. Return to Visual Studio Code and paste in the contents of the Windows clipboard into **Program.cs**.



1. Walk through the code in **Program.cs** to understand what it does.
   1. The code contains an outer **foreach** loop to iterate through tables and an inner **foreach** loop to iterate through columns.

Server server = new Server();

server.Connect(connectString);

Model model = server.Databases[0].Model;

foreach (Table table in model.Tables) {

foreach (Column column in table.Columns) {

// iterate through each column of every table

}

}

* 1. Inside the inner foreach loop, there is an **if** statement to check whether each column is both non-hidden and numeric,

// determine if column is visible and numeric

if ((column.IsHidden == false) &

(column.DataType == DataType.Int64 ||

column.DataType == DataType.Decimal ||

column.DataType == DataType.Double)) {

// add auto measure for each visible, numeric colum

}

* 1. Inside the **if** statement, there is code to generate a new measure any time if finds a non-hidden, numeric column.

// add automeasure for this column new measure

string measureName = $"Sum of {column.Name} ({table.Name})";

string expression = $"SUM('{table.Name}'[{column.Name}])";

string displayFolder = "Auto Measures";

Measure measure = new Measure() {

Name = measureName,

Expression = expression,

DisplayFolder = displayFolder

};

measure.Annotations.Add(new Annotation() { Value = "This is an Auto Measure" });

if (!table.Measures.ContainsName(measureName)) {

table.Measures.Add(measure);

}

Else {

table.Measures[measureName].Expression = expression;

table.Measures[measureName].DisplayFolder = displayFolder;

}

* 1. After the outer **foreach** loop, there's a call to **model.SaveChanges** to push the changes to the model in Power BI Desktop.

foreach (Table table in model.Tables) {

foreach (Column column in table.Columns) {

// code to add auto measures omitted for brevity

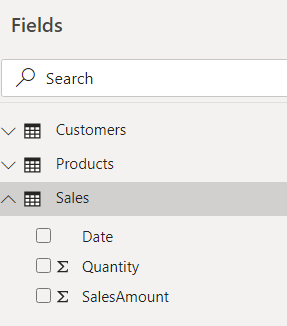
}

}

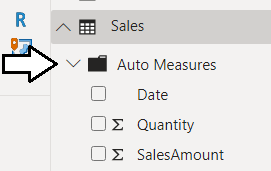
// save changes back to data model in Power BI Desktop

model.SaveChanges();

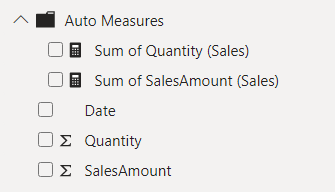
1. Run and test the code in **Program.cs**.
   1. Before running the program, return to Power BI Desktop and verify what fields are in the **Sales** table.



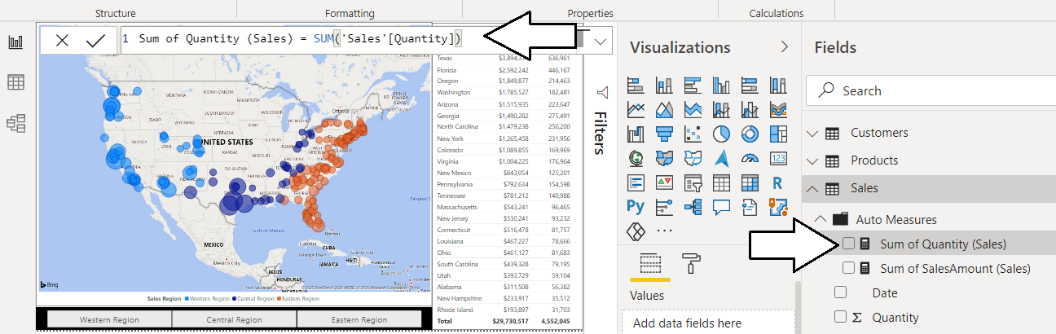
* 1. Return to Visual Studio Code.
  2. Run a debugging session by running the **Run > Start Debugging** command or pressing the **{F5}** key.
  3. Wait for the program to complete.
  4. Return to Power BI Desktop and verify you can see a new display folder named **Auto Measures**.
  5. Expand the **Auto Measures** folder.



* 1. Inside the **Auto Measure** folder, you should see two new measures as shown in the following screenshot.



* 1. Examine the DAX expression generated for each of the measures in the **Auto Measures** display folder.



You have now reached the end of lab two. In the next lab, you will learn how to execute a DAX query and handle query results.

### Exercise 3 Executing DAX Queries using the AMOMD Client Library

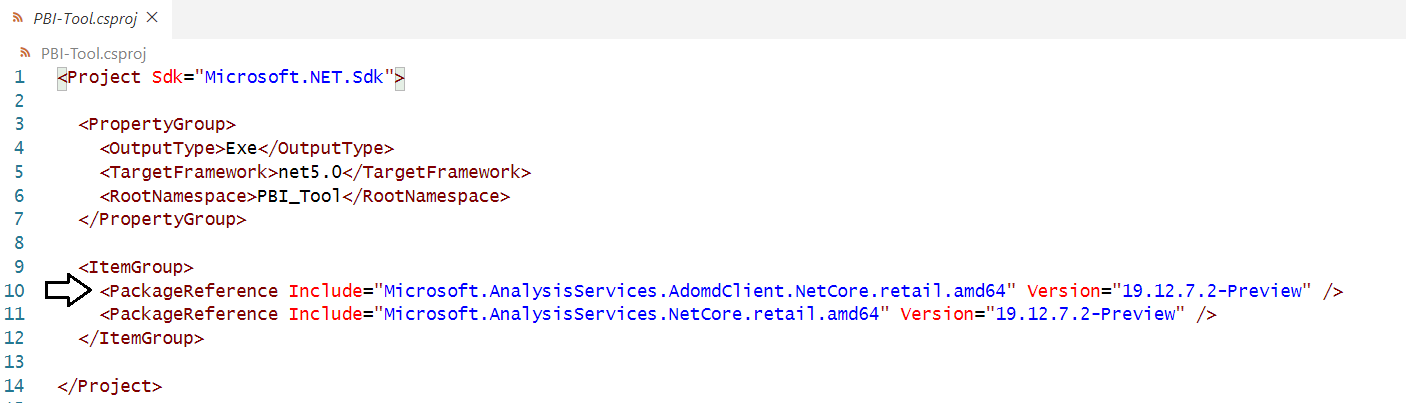
In this exercise, you will learn to execute DAX queries against a data model in Power BI Desktop using the AMOMD Client library. You will begin by adding a new package for AMOMD. Next, you will write and test code that executes DAX queries and converts the query results into a standard CSV file. In the second part of this exercise, you will mix TOM programming together with AMOMD in order to write code which determines what measures to create by examining the results of a DAX query.

1. Add a **AMOMD Client** NuGet package to your project using the so you can program against the Tabular Object Model.
   1. Return to the Terminal console.
   2. Type and execute the following **dotnet add package** command to add the **AMO** NuGet package

dotnet add package Microsoft.AnalysisServices.AdomdClient.NetCore.retail.amd64 --version 19.12.7.2-Preview

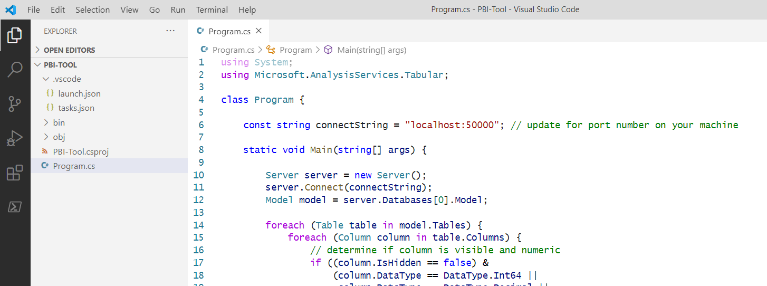
At the time that lab exercise was written, the latest version of this package was **19.12.7.2-Preview**. You can check out the information at the following URL to determine the latest release: <https://www.nuget.org/packages/Microsoft.AnalysisServices.AdomdClient.NetCore.retail.amd64>.

* 1. If you look at the **PBI-Tool.csproj** file, you will see there is a new package reference for the AMOMD Client library.

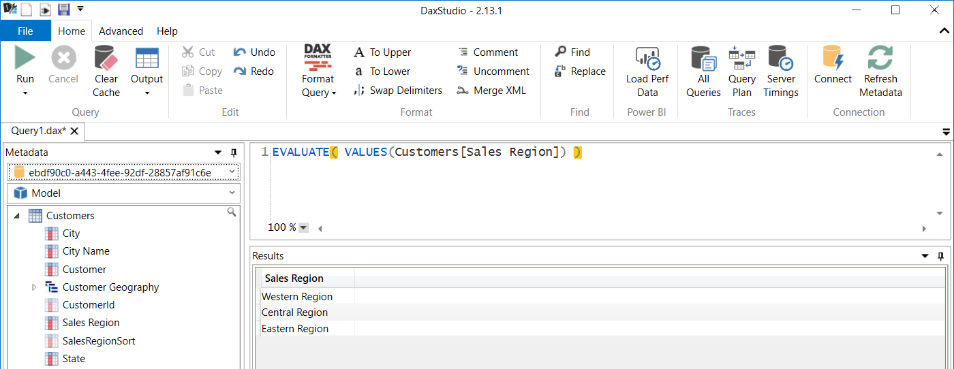


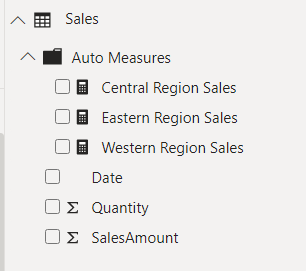
* 1. Close the **PBI-Tool.csproj** file without saving any changes.

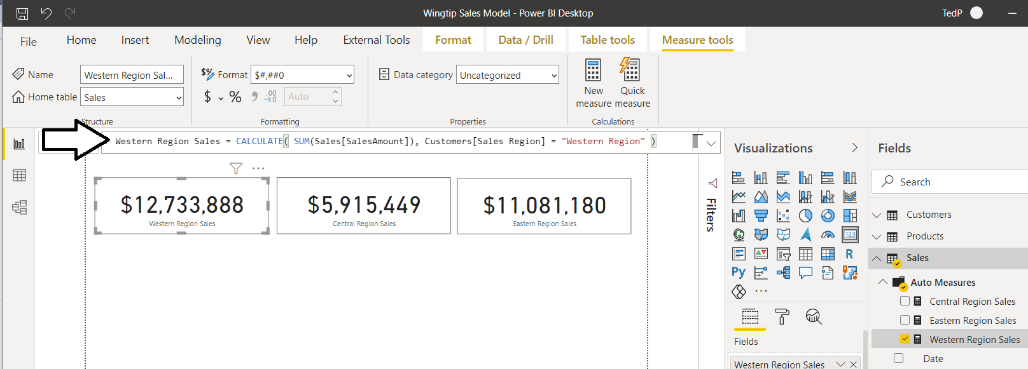
1. Copy and paste the Exercise 3 starter code into **Program.cs**.
   1. Open **Program.cs** and delete all the code that is currently inside.
   2. Copy the Exercise 3 starter code from the xxxx file in the **StarterFiles** folder to the Windows clipboard.
   3. Return to Visual Studio Code and paste in the contents of the Windows clipboard into **Program.cs**.



1. Walk through the code in **Program.cs** to understand what it does.







### Exercise 4: xxx

In this exercise, you will download xx.

1. Xxxxx
   1. xxxx