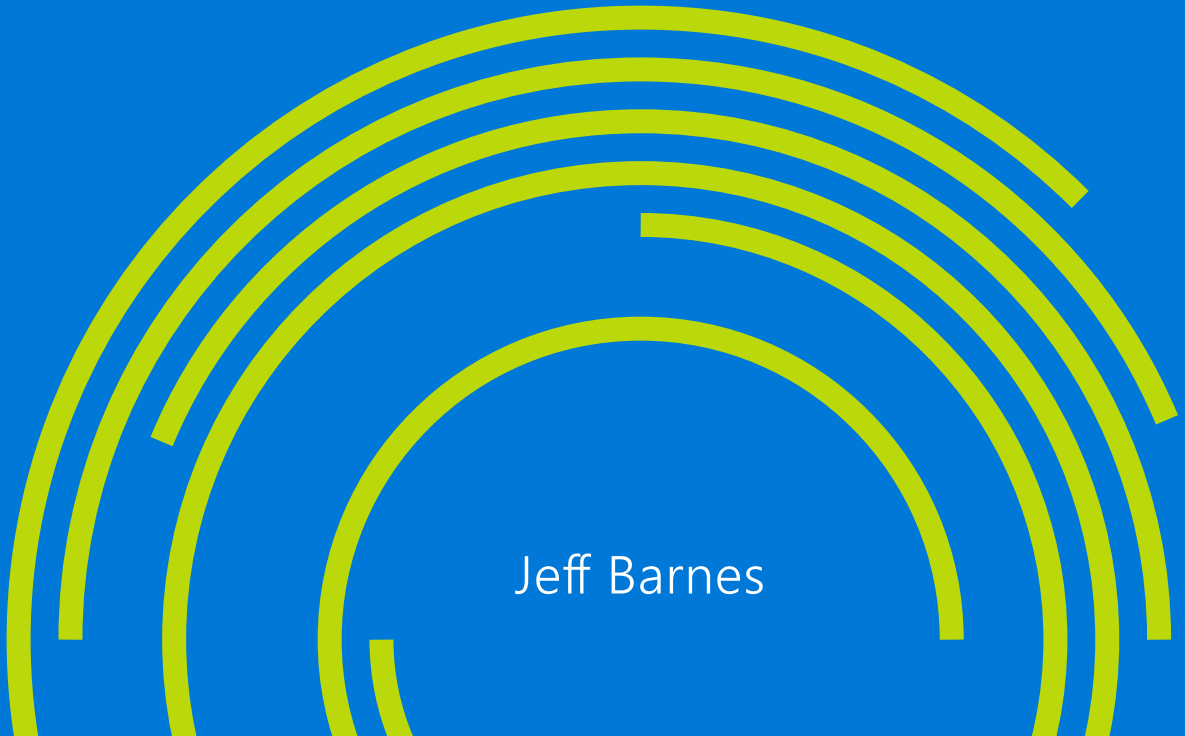
Microsoft

Azure Machine Learning

Microsoft Azure Essentials



Jeff Barnes

PUBLISHED BY
Microsoft Press
A division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

ISBN: 978-0-7356-9817-8

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Support at mspinput@microsoft.com. Please tell us what you think of this book at http://aka.ms/tellpress.

This book is provided "as-is" and expresses the authors' views and opinions. The views, opinions, and information expressed in this book, including URL and other Internet website references, may change without notice.

Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Acquisitions, Developmental, and Project Editor: Devon Musgrave
Editorial Production: nSight, Inc.
Copyeditor: Teresa Horton
Cover: Twist Creative

Table of Contents

Foreword

I'm thrilled to be able to share these Microsoft Azure Essentials ebooks with you. The power that Microsoft Azure gives you is thrilling but not unheard of from Microsoft. Many don't realize that Microsoft has been building and managing datacenters for over 25 years. Today, the company's cloud datacenters provide the core infrastructure and foundational technologies for its 200-plus online services, including Bing, MSN, Office 365, Xbox Live, Skype, OneDrive, and, of course, Microsoft Azure. The infrastructure is comprised of many hundreds of thousands of servers, content distribution networks, edge computing nodes, and fiber optic networks. Azure is built and managed by a team of experts working 24x7x365 to support services for millions of customers' businesses and living and working all over the globe.

Today, Azure is available in 141 countries, including China, and supports 10 languages and 19 currencies, all backed by Microsoft's $15 billion investment in global datacenter infrastructure. Azure is continuously investing in the latest infrastructure technologies, with a focus on high reliability, operational excellence, cost-effectiveness, environmental sustainability, and a trustworthy online experience for customers and partners worldwide.

Microsoft Azure brings so many services to your fingertips in a reliable, secure, and environmentally sustainable way. You can do immense things with Azure, such as create a single VM with 32TB of storage driving more than 50,000 IOPS or utilize hundreds of thousands of CPU cores to solve your most difficult computational problems.

Perhaps you need to turn workloads on and off, or perhaps your company is growing fast! Some companies have workloads with unpredictable bursting, while others know when they are about to receive an influx of traffic. You pay only for what you use, and Azure is designed to work with common cloud computing patterns.

From Windows to Linux, SQL to NoSQL, Traffic Management to Virtual Networks, Cloud Services to Web Sites and beyond, we have so much to share with you in the coming months and years.

I hope you enjoy this Microsoft Azure Essentials series from Microsoft Press. The first three ebooks cover fundamentals of Azure, Azure Automation, and Azure Machine Learning. And I hope you enjoy living and working with Microsoft Azure as much as we do.

Scott Guthrie

Executive Vice President

Cloud and Enterprise group, Microsoft Corporation

Introduction

Microsoft Azure Machine Learning (ML) is a service that a developer can use to build predictive analytics models (using training datasets from a variety of data sources) and then easily deploy those models for consumption as cloud web services. Azure ML Studio provides rich functionality to support many end-to-end workflow scenarios for constructing predictive models, from easy access to common data sources, rich data exploration and visualization tools, application of popular ML algorithms, and powerful model evaluation, experimentation, and web publication tooling.

This ebook will present an overview of modern data science theory and principles, the associated workflow, and then cover some of the more common machine learning algorithms in use today. We will build a variety of predictive analytics models using real world data, evaluate several different machine learning algorithms and modeling strategies, and then deploy the finished models as machine learning web service on Azure within a matter of minutes. The book will also expand on a working Azure Machine Learning predictive model example to explore the types of client and server applications you can create to consume Azure Machine Learning web services.

The scenarios and end-to-end examples in this book are intended to provide sufficient information for you to quickly begin leveraging the capabilities of Azure ML Studio and then easily extend the sample scenarios to create your own powerful predictive analytic experiments. The book wraps up by providing details on how to apply "continuous learning" techniques to programmatically "retrain" Azure ML predictive models without any human intervention.

Who should read this book

This book focuses on providing essential information about the theory and application of data science principles and techniques and their applications within the context of Azure Machine Learning Studio. The book is targeted towards both data science hobbyists and veterans, along with developers and IT professionals who are new to machine learning and cloud computing. Azure ML makes it just as approachable for a novice as a seasoned data scientist, helping you quickly be productive and on your way towards creating and testing machine learning solutions.

Detailed, step-by-step examples and demonstrations are included to help the reader understand how to get started with each of the key predictive analytic algorithms in use today and their corresponding implementations in Azure ML Studio. This material is useful not only for those who have no prior experience with Azure Machine Learning, but also for those who are experienced in the field of data science. In all cases, the end-to-end demos help reinforce the machine learning concepts with concrete examples and real-life scenarios. The chapters do build on each other to some extent; however, there is no requirement that you perform the hands-on demonstrations from previous

chapters to understand any particular chapter.

Assumptions

We expect that you have at least a minimal understanding of cloud computing concepts and basic web services. There are no specific skills required overall for getting the most out of this book, but having some knowledge of the topic of each chapter will help you gain a deeper understanding. For example, the chapter on creating Azure ML client and server applications will make more sense if you have some understanding of web development skills. Azure Machine Learning Studio automatically generates code samples to consume predictive analytic web services in C#, Python, and R for each Azure ML experiment. A working knowledge of one of these languages is helpful but not necessary.

This book might not be for you if…

This book might not be for you if you are looking for an in-depth discussion of the deeper mathematical and statistical theories behind the data science algorithms covered in the book. The goal was to convey the core concepts and implementation details of Azure Machine Learning Studio to the widest audience possible—who may not have a deep background in mathematics and statistics.

Organization of this book

This book explores the background, theory, and practical applications of today's modern data science algorithms using Azure Machine Learning Studio. Azure ML predictive models are then generated, evaluated, and published as web services for consumption and testing by a wide variety of clients to complete the feedback loop.

The topics explored in this book include:

- **Chapter 1, "Introduction to the science of data,"** shows how Azure Machine Learning represents a critical step forward in democratizing data science by making available a fully-managed cloud service for building predictive analytics solutions.
- **Chapter 2, "Getting started with Azure Machine Learning,"** covers the basic concepts behind the science and methodology of predictive analytics.
- **Chapter 3, "Using Azure ML Studio,"** explores the basic fundamentals of Azure Machine Learning Studio and helps you get started on your path towards data science greatness.
- **Chapter 4, "Creating Azure ML client and server applications."** expands on a working Azure Machine Learning predictive model and explores the types of client and server applications that you can create to consume Azure Machine Learning web services.

- **Chapter 5, "Regression analytics,"** takes a deeper look at some of the more advanced machine learning algorithms that are exposed in Azure ML Studio.
- **Chapter 6, "Cluster analytics,"** explores scenarios where the machine conducts its own analysis on the dataset, determines relationships, infers logical groupings, and generally attempts to make sense of chaos by literally determining the forests from the trees.
- **Chapter 7, "The Azure ML Matchbox recommender,"** explains one of the most powerful and pervasive implementations of predictive analytics in use today on the web today and how it is crucial to success in many consumer industries.
- **Chapter 8, "Retraining Azure ML models,"** explores the mechanisms for incorporating "continuous learning" into the workflow for our predictive models.

Conventions and features in this book

This book presents information using the following conventions designed to make the information readable and easy to follow:

- To create specific Azure resources, follow the numbered steps listing each action you must take to complete the exercise.
- There are currently two management portals for Azure: the Azure Management Portal at http://manage.windowsazure.com and the new Azure Preview Portal at http://portal.azure.com. This book assumes the use of the original Azure Management Portal in all cases.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press Tab.

System requirements

For many of the examples in this book, you need only Internet access and a browser (Internet Explorer 10 or higher) to access the Azure portal. Chapter 4, "Creating Azure ML client and server applications," and many of the remaining chapters use Visual Studio to show client applications and concepts used in developing applications for consuming Azure Machine Learning web services. For these examples, you will need Visual Studio 2013. You can download a free copy of Visual Studio Express at the link below. Be sure to scroll down the page to the link for "Express 2013 for Windows Desktop":
http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

The following are system requirements:

- Windows 7 Service Pack 1, Windows 8, Windows 8.1, Windows Server 2008 R2 SP1, Windows

Server 2012, or Windows Server 2012 R2

- Computer that has a 1.6GHz or faster processor (2GHz recommended)
- 1 GB (32 Bit) or 2 GB (64 Bit) RAM (Add 512 MB if running in a virtual machine)
- 20 GB of available hard disk space
- 5400 RPM hard disk drive
- DirectX 9 capable video card running at 1024 x 768 or higher-resolution display
- DVD-ROM drive (if installing Visual Studio from DVD)
- Internet connection

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2013.

Acknowledgments

This book is dedicated to my father who passed away during the time this book was being written, yet wisely predicted that computers would be a big deal one day and that I should start to "ride the wave" of this exciting new field. It has truly been quite a ride so far.

This book is the culmination of many long, sacrificed nights and weekends. I'd also like to thank my wife Susan, who can somehow always predict my next move long before I make it. And to my children, Ryan, Brooke, and Nicholas, for their constant support and encouragement.

Special thanks to the entire team at Microsoft Press for their awesome support and guidance on this journey. Most of all, it was a supreme pleasure to work with my editor, Devon Musgrave, who provided constant advice, guidance, and wisdom from the early days when this book was just an idea, all the way through to the final copy. Brian Blanchard was also critical to the success of this book as his keen editing and linguistic magic helped shape many sections of this book.

Errata, updates, & support

We've made every effort to ensure the accuracy of this book. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

http://aka.ms/AzureML/errata

If you discover an error that is not already listed, please submit it to us at the same page.

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to http://support.microsoft.com.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

Free training from Microsoft Virtual Academy

The Microsoft Azure training courses from Microsoft Virtual Academy cover key technical topics to help developers gain the knowledge they need to be a success. Learn Microsoft Azure from the true experts. Microsoft Azure training includes courses focused on learning Azure Virtual Machines and virtual networks. In addition, gain insight into platform as a service (PaaS) implementation for IT Pros, including using PowerShell for automation and management, using Active Directory, migrating from on-premises to cloud infrastructure, and important licensing information.

http://www.microsoftvirtualacademy.com/product-training/microsoft-azure

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: http://twitter.com/MicrosoftPress

Chapter 4

Creating Azure Machine Learning client and server applications

In this chapter, we build on the example provided in Chapter 3, "Using Azure ML Studio," where we created our first Azure Machine Learning experiment for predicting income levels using an end-to-end example. As a brief refresher, we started with a raw data set downloaded from the UCI data repository, then loaded and saved it into our Azure Machine Learning workspace. Next, we created a workflow to train a new predictive model using the Two-Class Boosted Decision Tree module. When it was ready to publish, we set the publish inputs and outputs and exposed the Azure Machine Learning prediction model as a new web service. We then briefly tested our service by using the "test" link to pop up a window to interactively enter the values to be used to call the web service, and see the resulting response.

The purpose of this chapter is to further expand on a working Azure Machine Learning predictive model and explore the types of client and server applications that you can create to consume Azure Machine Learning web services. Developing a cool and innovative machine learning predictive model is really only half the fun. The other half comes when you can quickly expose and incorporate a new Azure Machine Learning predictive model web service into your core business applications to expedite testing or production scenarios.

In this chapter, we walk through building various client and server applications meant to consume Azure Machine Learning predictive models exposed as web services. We start with some very simple basic client applications and then explore a full-blown ASP.NET Web API web service implementation that implements the REST protocol and provides Azure Machine Learning predictions using JSON payloads for inputs and outputs. This approach allows you to quickly and easily expose Azure Machine Learning web services to a wide variety of modern web and mobile applications for mass consumption.

Why create Azure Machine Learning client applications?

One of the drawbacks with using the built-in, quick tester, pop-up application provided with Azure Machine Learning Studio is that you must know exactly what all the valid input options are for each of the 14 input parameters to successfully make a prediction.

The following is a listing of the attributes and allowed values from the UCI data repository site at http://archive.ics.uci.edu/ml/datasets/Census+Income.

- **age** Continuous
- **workclass** Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
- **fnlwgt** Continuous
- **education** Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool
- **education-num** Continuous
- **marital-status** Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse
- **occupation** Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces
- **relationship** Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried
- **race** White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black
- **sex** Female, Male
- **capital-gain** Continuous
- **capital-loss** Continuous
- **hours-per-week** Continuous
- **native-country** United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holland-Netherlands

Think about how time-consuming and impractical it now becomes to test a new Azure Machine Learning prediction model. Having to memorize or look up all the allowable parameter options for each variable for each iteration of testing is not the most efficient way to test a model, not to mention the fact that these parameters are also case-sensitive. Input parameter accuracy (even to the degree of proper case sensitivity) becomes a crucial element to providing accurate machine learning predictions.

Creating a client application that simplifies the process and providing drop-down list boxes of valid choices allows the user to quickly and easily submit various combinations of prediction inputs. The user

then gets immediate feedback by seeing the corresponding results in real time.

Azure Machine Learning web services sample code

Let's start by looking at the default sample code that is provided for calling Azure Machine Learning web services. After you have exposed an Azure Machine Learning web service, you can reach a code sample page, by navigating to your Azure Machine Learning workspace from the main Azure menu. Follow these navigation steps:

1. Navigate to Azure Machine Learning/<YourWorkSpace>. Select the Web Services tab as shown in Figure 4-1.

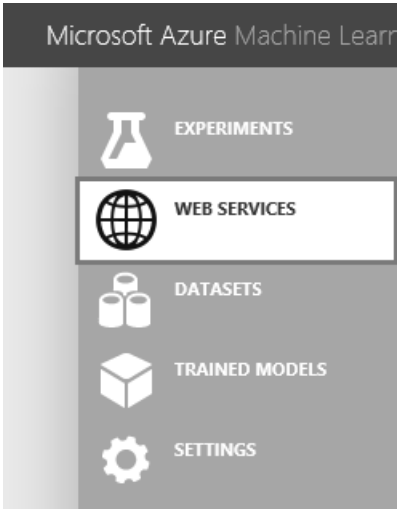


FIGURE 4-1 Navigating to the Azure Machine Learning workspace and selecting the Web Services tab.

2. Select the specific implementation version for the Azure Machine Learning web service as shown in Figure 4-2.

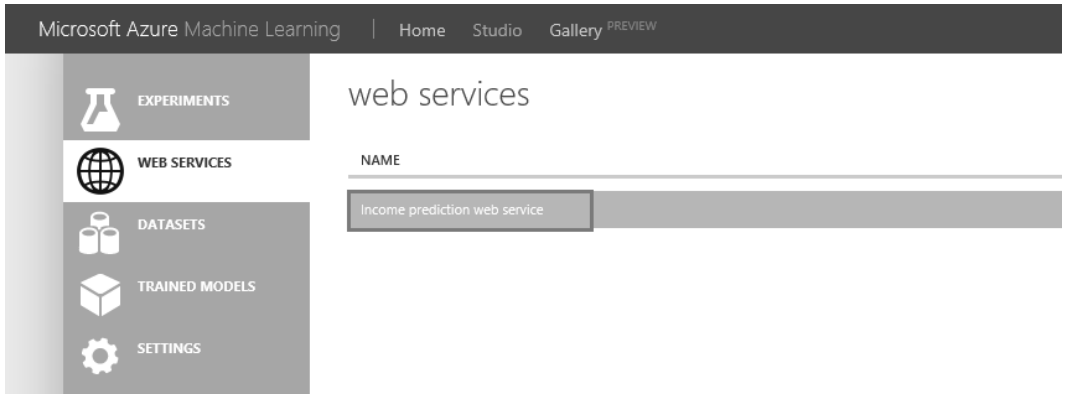


FIGURE 4-2 Select the specific implementation version for the Azure Machine Learning web service.

3. Select the API Help Page for the Request/Response option as shown in Figure 4-3.

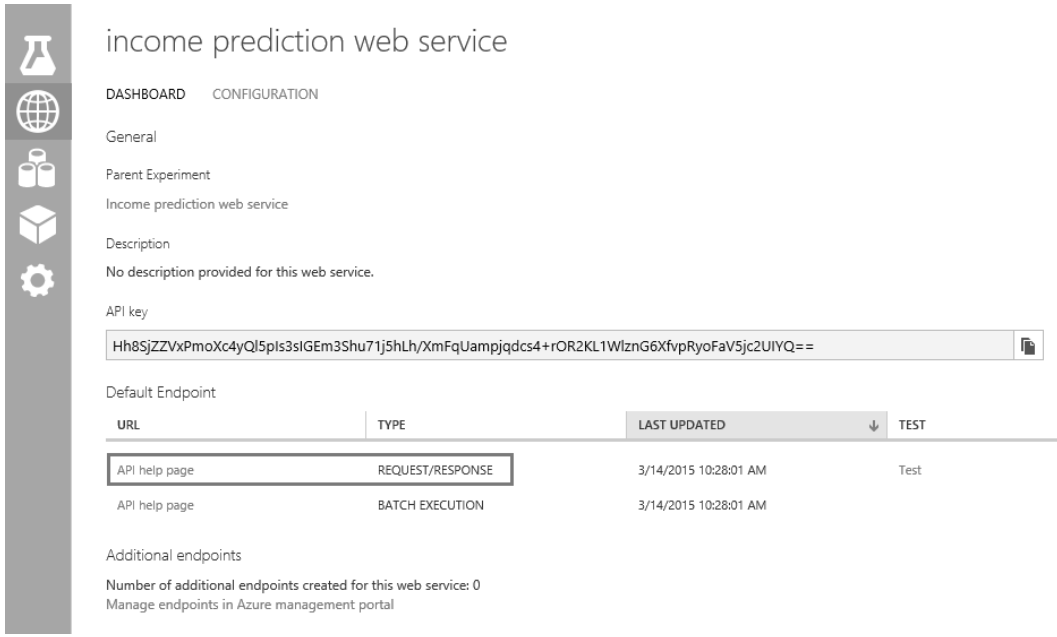


FIGURE 4-3 Select the API Help Page for the Request/Response methods.

4. You will then be directed to a webpage where you can easily view all the technical details necessary to successfully invoke this Azure Machine Learning web service via an HTTP POST request as seen in Figure 4-4.

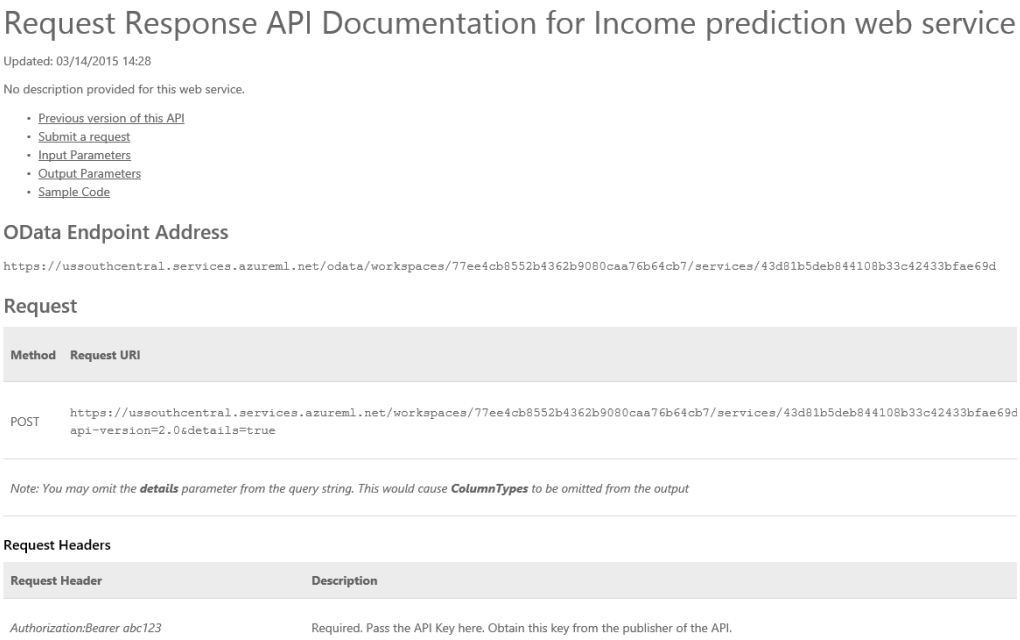


FIGURE 4-4 The Azure Machine Learning web service API Help page.

This Help page for the Azure Machine Learning API web service provides specific implementation details about the following aspects of making an Azure Machine Learning web service via an HTTP POST request:

- OData End Point Address
- Method Request URI Format
- POST Request Headers
- Sample REQUEST body
- RESPONSE – Status Codes
- RESPONSE – Headers
- RESPONSE – Body
- RESPONSE – Sample Reply

If you scroll all the way down the page, you will see a Sample Code section. This will help you get a jump-start on creating new clients that can call your new Azure Machine Learning web service. Note in Figure 4-5 that implementation samples are provided for the programming languages C#, Python, and R.

Sample Code

C# | Python | R

```
// This code requires the Nuget package Microsoft.AspNet.WebApi.Client to be installed.
// Instructions for doing this in Visual Studio:
// Tools -> Nuget Package Manager -> Package Manager Console
// Install-Package Microsoft.AspNet.WebApi.Client

using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Formatting;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;

namespace CallRequestResponseService
{

    public class StringTable
    {
        public string[] ColumnNames { get; set; }
        public string[,] Values { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
```

FIGURE 4-5 Sample code for invoking an Azure Machine Learning web service.

C# console app sample code

Let's start by looking at the default sample code that is provided for calling our Azure Machine Language web service from C#. To get started, you will need a copy of Visual Studio. You can download a free copy of Visual Studio Express at the following link. Be sure to scroll down the page to the link for Express 2013 for Windows Desktop:

http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

Start by opening Visual Studio 2013. Then select File/New Project/Visual C#/Windows Desktop/Console Application. You will see a screen similar to Figure 4-6.

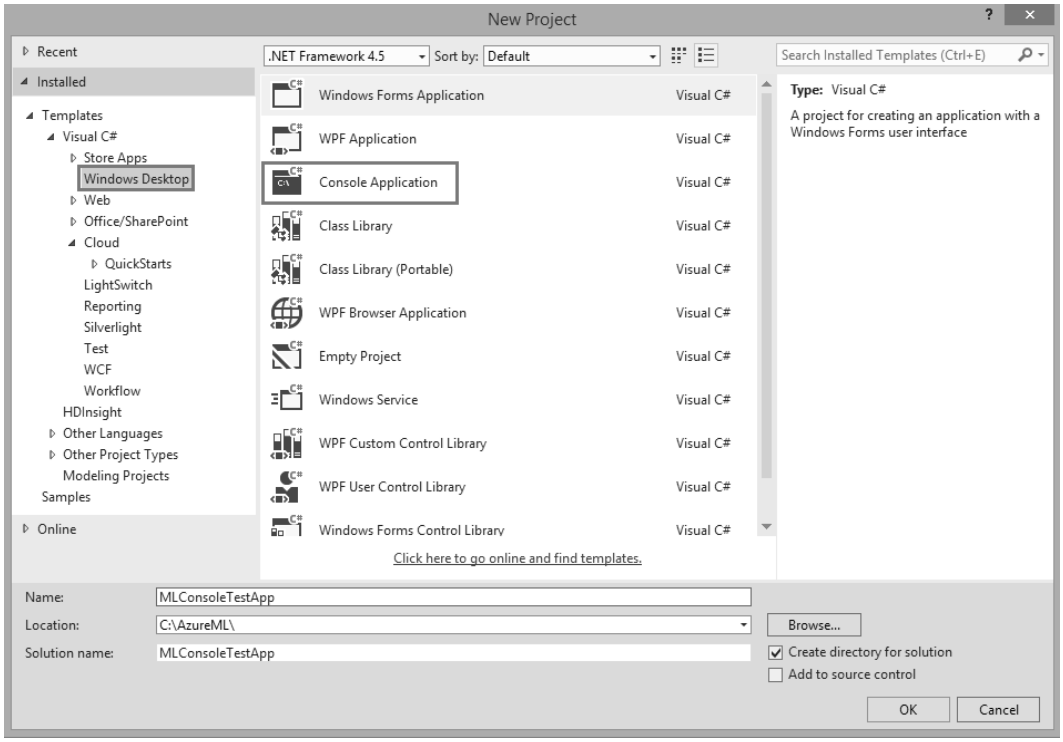


FIGURE 4-6 Creating a Desktop console application in C#.

Choose MLConsoleTestApp as the project name. Pick a location folder for your project. Once the solution has been generated, cut, copy, and paste the entire sample C# code from the Azure Machine Learning web service API Help page into the Program.cs module. Be sure to delete all of the original code in the module.

The next step is to install a required NuGet package to allow the application to make HTTP POST requests. Open the NuGet Package Manager Console by selecting Tools/NuGet Package Manager/Package Manager Console as shown in Figure 4-7.

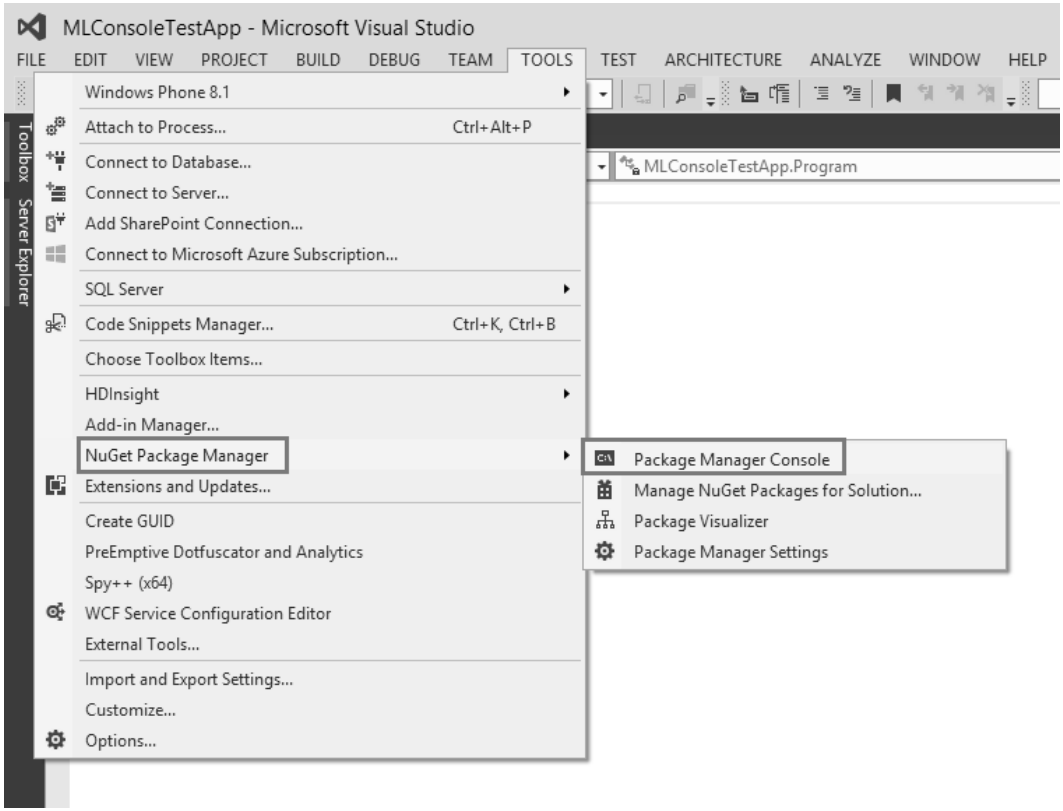


FIGURE 4-7 Invoking the NuGet Package Manager Console.

Once you have a NuGet Package Manager Console window, enter the following command to start the installation of the Microsoft.AspNet.WebApi.Client NuGet packages and their dependencies:

```
Install-Package Microsoft.AspNet.WebApi.Client
```

After the command has finished, you should be able to successfully build the application with no compiler errors.

Now, let's examine the sample C# code in Figure 4-8. Notice that there is a class called StringTable that is used to define a Request to our Azure Machine Learning web service.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Formatting;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;

namespace CallRequestResponseService
{
    2 references
    public class StringTable
    {
        1 reference
        public string[] ColumnNames { get; set; }
        1 reference
        public string[,] Values { get; set; }
    }
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            InvokeRequestResponseService().Wait();
        }
        1 reference
        static async Task InvokeRequestResponseService()...
    }
}
```

FIGURE 4-8 Sample Visual Studio 2013 C# console application.

The StringTable class contains two string arrays used for invoking the web service.

- The ColumnNames string array contains the column names for the input parameters.
- The Values string array will contain the individual values for each corresponding parameter passed into the web service call.

All the magic really happens in the Main routine, where the program invokes the InvokeRequestResponseService()method call and then displays the results.

Figure 4-9 shows the InvokeRequestResponseService()method call where we can see the actions required to make a successful web service call.

```
static async Task InvokeRequestResponseService()
{
    using (var client = new HttpClient())
    {
        var scoreRequest = new
        {
            Inputs = new Dictionary<string, StringTable>() {
                {
                    "input1",
                    new StringTable()
                    {
                        ColumnNames = new string[] {"age", "workclass", "fnlwgt", "education", "education-num", "marital-status", "occupation", "relationship", "race", "sex'
                        Values = new string[,] {  { "0", "value", "0", "value", "0", "value", "value", "value", "value", "value", "0", "0", "0", "value", "value" },  { "0",
                    }
                },
                },
            GlobalParameters = new Dictionary<string, string>()
            {
            }
        };
        const string apiKey = "abc123"; // Replace this with the API key for the web service
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", apiKey);
        client.BaseAddress = new Uri("https://ussouthcentral.services.azureml.net/workspaces/77ee4cb8552b4362b9080caa76b64cb7/services/43d81b5deb844108b33c42433bfae69d/execu
        HttpResponseMessage response = await client.PostAsJsonAsync("", scoreRequest);
        if (response.IsSuccessStatusCode)
        {
            string result = await response.Content.ReadAsStringAsync();
            Console.WriteLine("Result: {0}", result);
        }
        else
        {
            Console.WriteLine("Failed with status code: {0}", response.StatusCode);
        }
    }
```

FIGURE 4-9 Sample C# code to invoke Azure Machine Learning web services.

There are some noteworthy aspects to point out in the preceding sample code.

- A dictionary object named Inputs is populated with two string arrays.
- The first string array named ColumnNames is prepopulated with the names for the web service input parameters.
- The second string array named Values is prepopulated with default numeric and string values for the web service input parameters.
- All the initial code sample input values are set to 0 or value to invoke a successful web service call.
- The correct input values would come from selecting appropriate options for each of the 14 parameters for the UCI data repository site at http://archive.ics.uci.edu/ml/datasets/Census+Income.
- An HTTP Request header for Authorization is set based on the Azure Machine Learning web service API key field for this web service invocation.
- You will need to provide your own value for the apikey field from your Azure Machine Learning web service.
- The client.BaseAddress variable is set to the specific URL instance of your Azure Machine Learning web service.

You can find the API key for your web service by navigating to the Web Services tab of your Azure Machine Learning Studio workspace. The API key can be found by clicking the specific web service version for your experiment as shown in Figure 4-10.

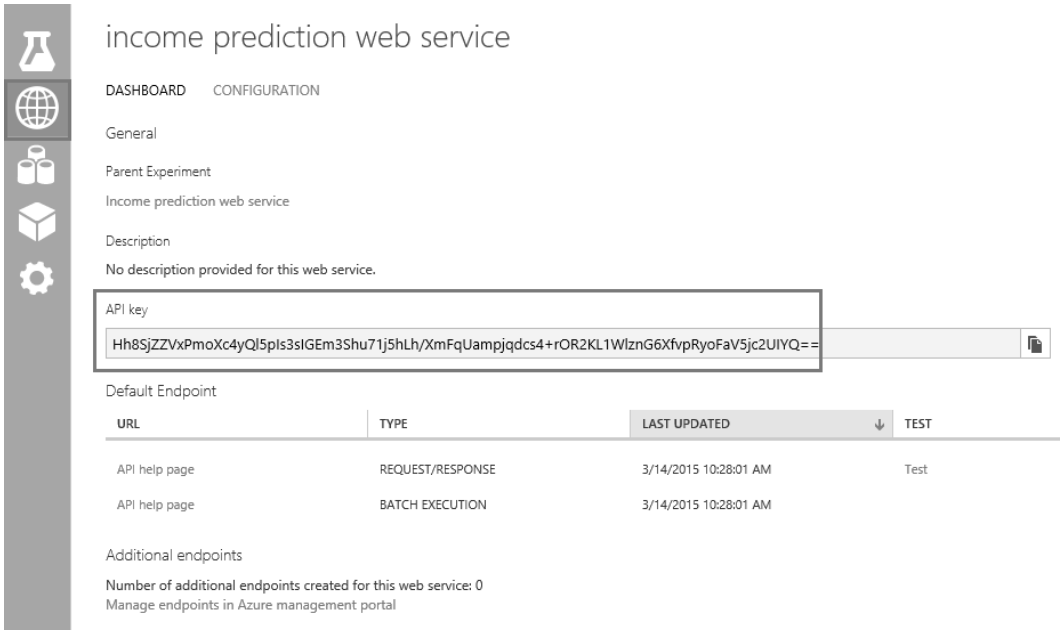


FIGURE 4-10 The Azure Machine Learning Web Services page showing the location of the API key.

Note that if we run the console app at this point, we will get the results shown in Figure 4-11, based on providing all zeroes as inputs.

FIGURE 4-11 Results returned from the C# console application after invoking the Azure Machine Learning web service.

In this example, the response back from our call to the Azure Machine Learning web service indicates that this input candidate likely makes less than $50,000 a year. The corresponding confidence level is 0.0010072038276121.

Now watch what happens if we replace the Values input string array parameters with valid data that is likely to have a higher score as in the following code fragment:

```
Values = new string[,] {  {
                                                    "52",
                                                    "Self-emp-not-inc",
                                                    "209642",
                                                    "HS-grad",
                                                    "12",
                                                    "Married-civ-spouse",
                                                    "Exec-managerial",
                                                    "Husband",
                                                    "White",
                                                    "Male",
                                                    "0",
                                                    "0",
                                                    "45",
                                                    "United-States",
                                                    "0"
                                                }
                                            }
                                        }
                                    },
```

Figure 4-12 shows the results of a more accurate prediction. Based on the updated, nonzero, and valid input parameters given, the candidate is likely to make more than $50,000 a year with a confidence level of 0.549394130706787.

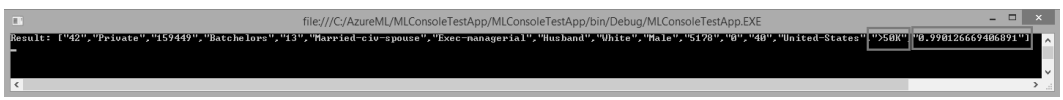


FIGURE 4-12 Results from the Azure Machine Learning web service call after adjusting the input parameters.

R sample code

In addition to C#, one of the additional code samples provided for invoking an Azure Machine Learning web service is in the R programming language. In the world of data science, R is a very popular programming language for developing data analytics, statistical modeling, and graphics applications. You can find an excellent tutorial on using the R programming language with Azure Machine Learning at http://azure.microsoft.com/en-us/documentation/articles/machine-learning-r-quickstart/.

To get started with the R code sample, you will need to download a few prerequisites, including the following:

- Open source R (free) downloads are available at the Comprehensive R Archive Network or CRAN at http://www.r-project.org/.

- RStudio is a powerful and productive user interface for R. It's free and open source, and works great on Windows, Mac, and Linux platforms. Download the desktop version at http://www.rstudio.com/products/rstudio/.
- A tutorial introduction to RStudio is available at https://support.rstudio.com/hc/en-us/sections/200107586-Using-RStudio.

Once you have installed all these prerequisites, start the RStudio application and paste in the R sample code from the Azure Machine Learning web services page. Figure 4-13 shows a screenshot of the RStudio environment and the pasted sample Azure Machine Learning R code to invoke our web service.

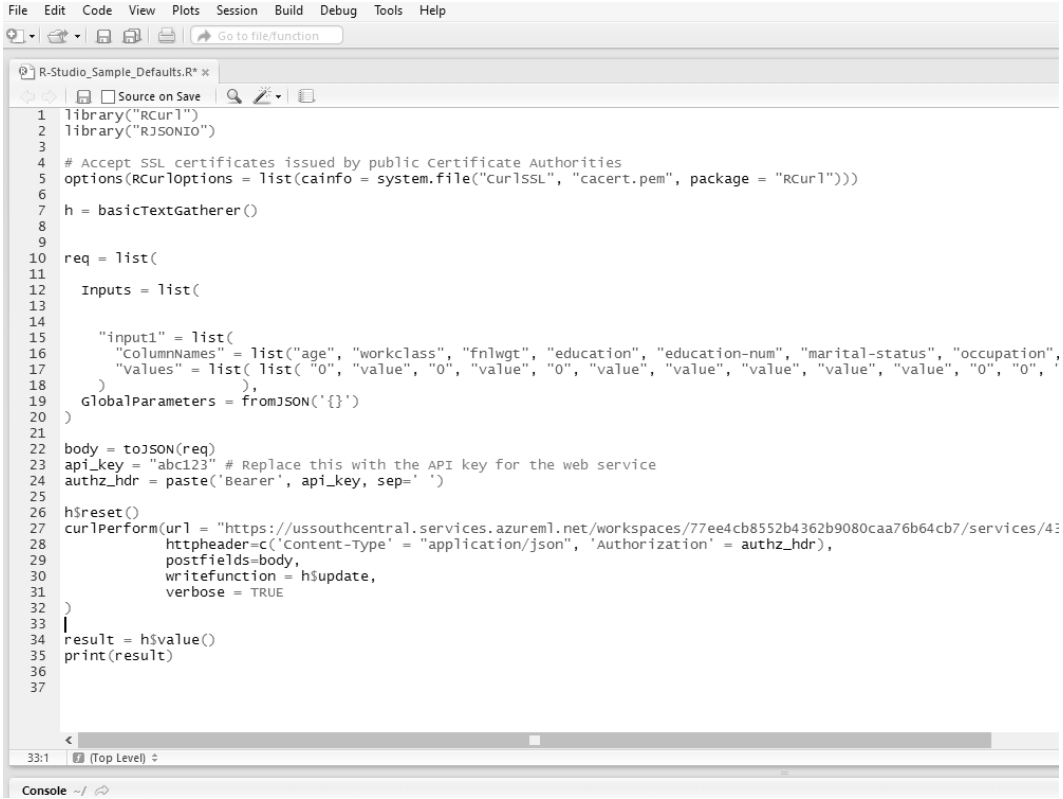


FIGURE 4-13 Sample R code in RStudio to invoke the Azure Machine Learning web service.

Before we can run the R code sample, we must first install two required packages in RStudio that appear at the top of the Azure Machine Learning web service R code sample:

- library("RCurl")
- library("RJSONIO")

To do this, simply click the menu option for Tools, then click Install Packages, as shown in Figure 4-14.

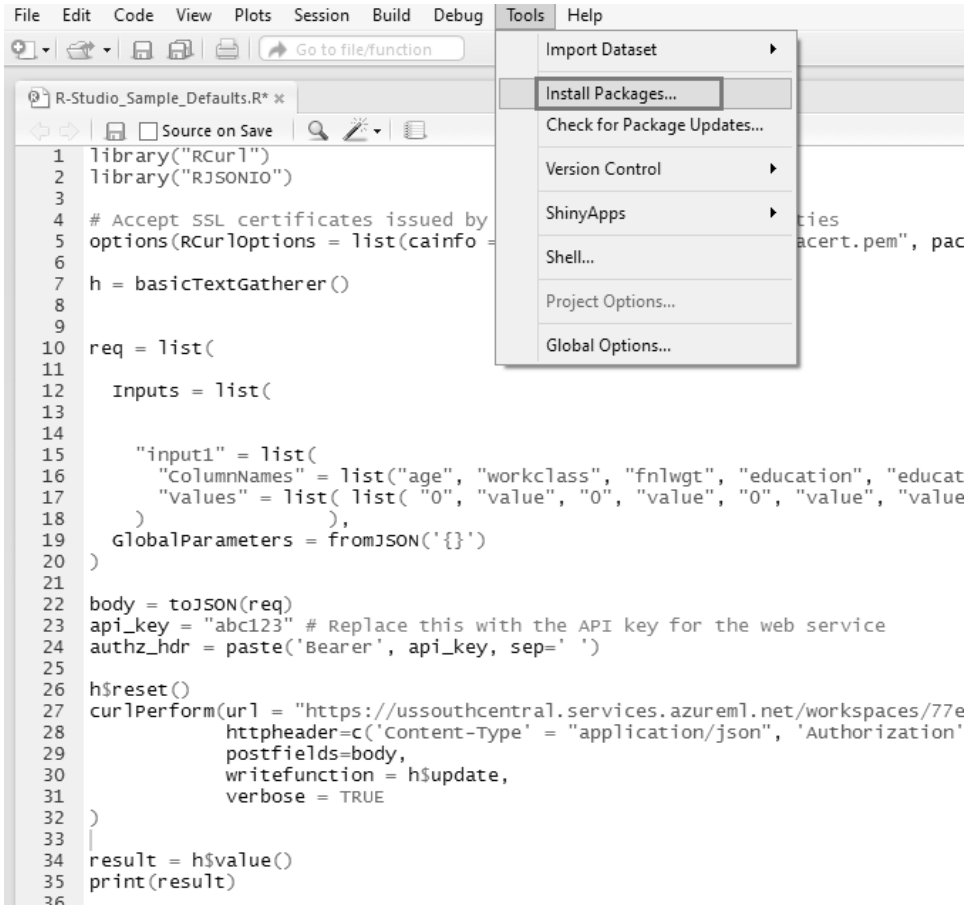


FIGURE 4-14 Invoking the Install Packages option in the RStudio environment.

Next, type the following package names: **RCurl, RJSONIO.** Click Install as shown in Figure 4-15.

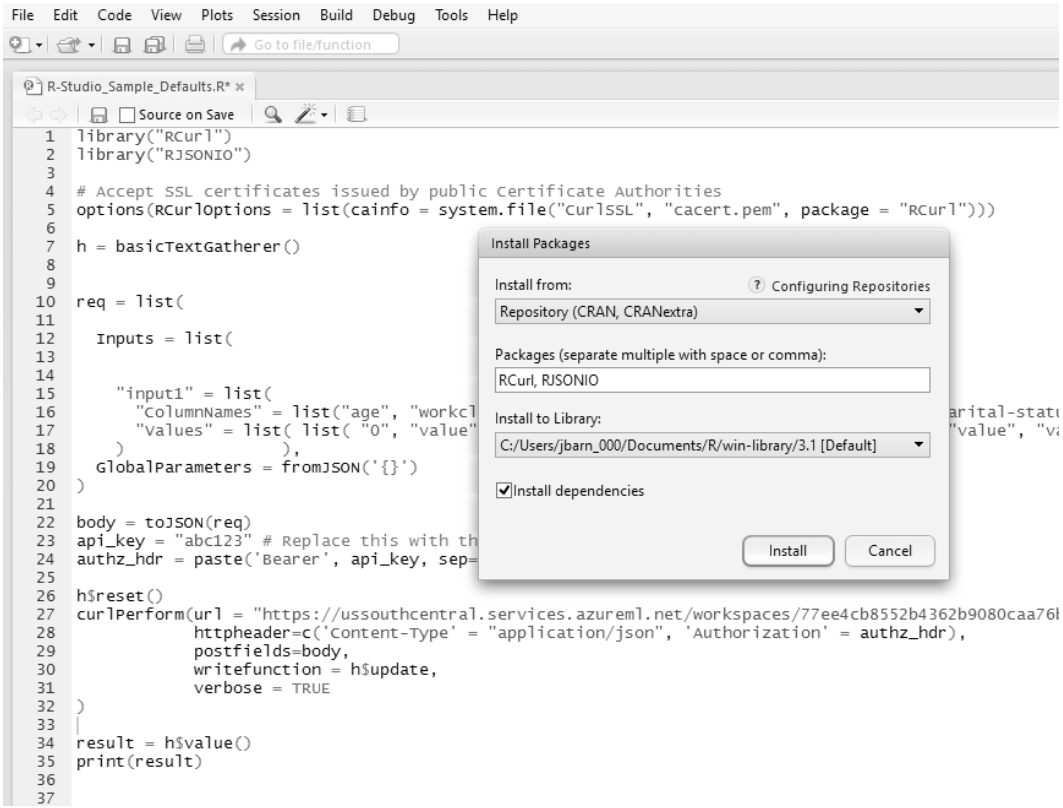


FIGURE 4-15 Installing the R packages for RCurl and RJSONIO.

After the packages have been downloaded and installed, we are almost ready to run the R code sample. First, though, we need to update the required parameters for the Azure Machine Learning web service API key.

You can find the API key for your web service by navigating to the Web Services tab of your Azure Machine Learning Studio workspace. The API key can be found by clicking on the specific web service version for your experiment. It is then displayed on the web services dashboard.

Once you have updated the apikey parameter in the R code sample, the code will be ready to run. You can now run through the code by selecting or highlighting all the code in the main window and then clicking Run in the top left quadrant of the RStudio screen. As you step thru each line of the R code, you will see the code being executed on the right side of the RStudio environment. The bottom left portion of the screen contains the results of each call. The results of the call to our Azure Machine Learning web service are shown in Figure 4-16.

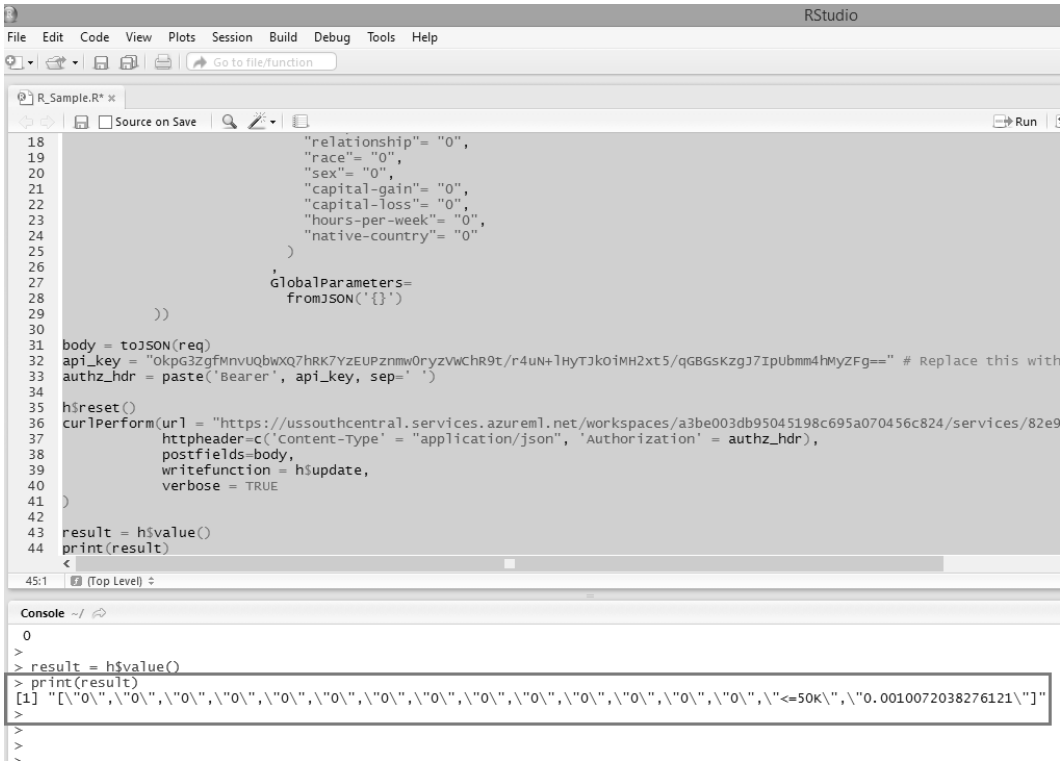


FIGURE 4-16 Invoking the Azure Machine Learning web service from within RStudio.

Because all the input parameters were set to 0 in the R code sample, the result of the call is a prediction of <=50K and a very low confidence factor of 0.0010072038276121. Just as in the previous C# console code sample, if we change the input parameters to contain valid inputs, we will get a different outcome. You can do this by updating the Values list parameters as in the following code sample.

```
"Values" =
  list(
    list(
      "52",
      "Self-emp-not-inc",
      "209642",
      "HS-grad",
      "12",
      "Married-civ-spouse",
      "Exec-managerial",
      "Husband",
      "White",
      "Male",
      "0",
```

```
        "0",
        "45",
        "United-States",
        "0"
        )
      )
  )
```

Now, when we rerun the updated R code sample in RStudio with valid (and more reasonable) input parameters, we get a payload back with all the original input parameters, along with a prediction of >50 with a confidence factor of 0.549394130706787, as shown in Figure 4-17.

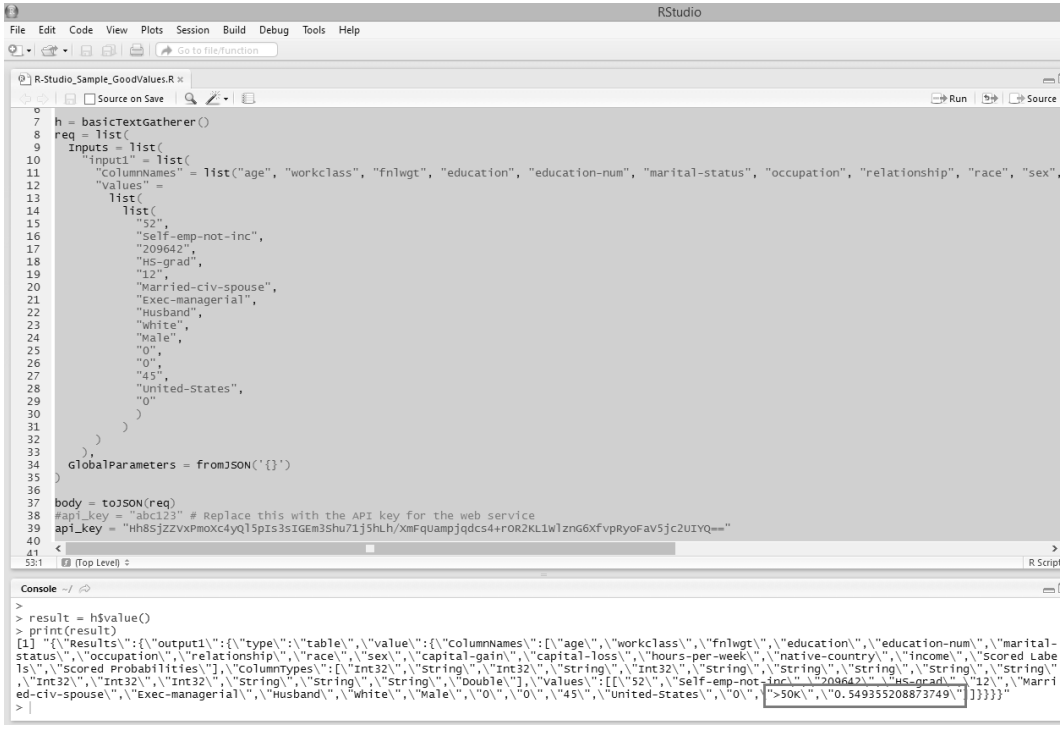


FIGURE 4-17 Invocation of the Azure Machine Learning web service from the R sample code with different input parameters.

Moving beyond simple clients

The last two code samples in C# and R were provided as part of the Azure Machine Learning web service environment for use as basic guidance for invoking an Azure Machine Learning web service.

These code samples are effective in helping you get started with building simple clients in various programming languages such as C#, Python, and R, to invoke Azure Machine Learning web services.

They fall short, however, in terms of providing an optimized user experience for entering the many various parameter input combinations required to fully exercise and test an Azure Machine Learning predictive model.

To that end, we explore the options for creating more user-friendly clients. We extend the web services so that the widest variety of desktop, web, and mobile clients can consume the Azure Machine Learning predictive models developed within Azure Machine Learning Studio.

Cross-Origin Resource Sharing and Azure Machine Learning web services

Cross-Origin Resource Sharing (CORS) is a mechanism that allows web objects (e.g., JavaScript) from one domain to request objects (web service requests) from another domain. Cross-origin requests initiated from scripts have been subject to restrictions primarily for security reasons. CORS provides a way for web servers to support cross-site access controls, which enable secure cross-site data transfers.

At the time of this writing, it should be noted that Azure Machine Learning web services do not currently support CORS requests. See the following link for more information:

https://social.msdn.microsoft.com/Forums/en-US/b6ddeb77-30e1-45b2-b7c1-eb4492142c0a/azure-ml-published-web-services-cross-origin-requests?forum=MachineLearning.

This CORS restriction really means that if you wish to fully exploit Azure Machine Learning web services for deployment, testing, and production for a wide variety of (web) clients, you will need to host your own server-side applications. You basically have two choices.

- Host a web application, such as an ASP.NET webpage, and invoke the Azure Machine Learning web service server-side to conform to the current Azure Machine Learning CORS restrictions.
- Host your own web service that does provide CORS support and can in turn invoke the Azure Machine Learning web service on behalf of a wide variety of web and mobile clients via modern protocols and data formats like REST and JSON.

We explore each of these options in the next sections.

Create an ASP.NET Azure Machine Learning web client

To provide a better experience for testing our new Azure Machine Learning income prediction web service, we will build an ASPX webpage. This webpage is based on the sample C# code provided with our Azure Machine Learning web service. If you don't already have a copy of Visual Studio, you can download a free copy of Visual Studio 2013 Express at http://www.visualstudio.com/en-us/products/

visual-studio-express-vs.aspx. Let's start by creating a new ASP.NET Web Application project as shown in Figure 4-18.

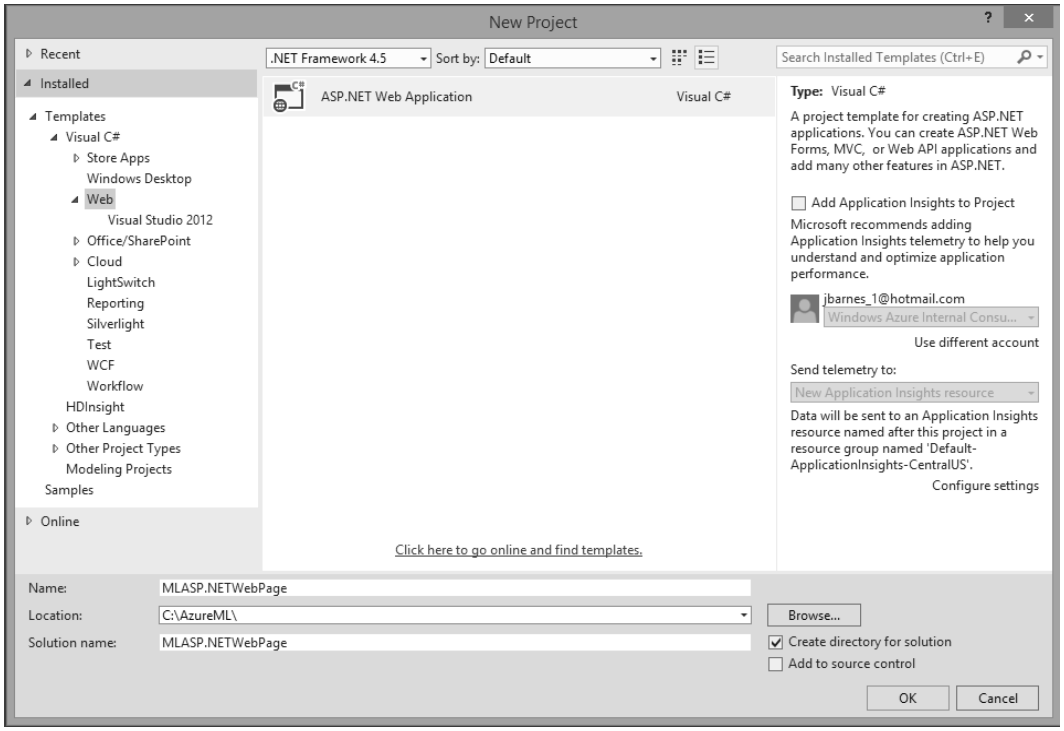


FIGURE 4-18 Create a new ASP.NET web application in Visual Studio 2013.

Next, select an Empty web project template as shown in Figure 4-19.

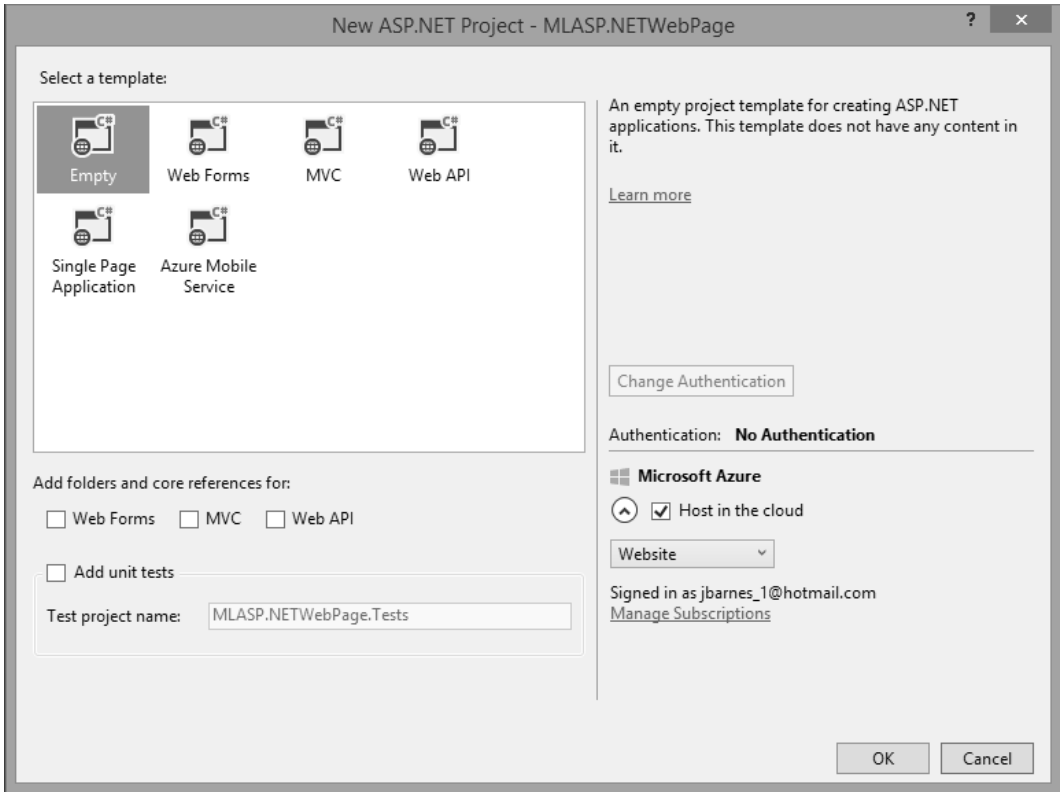


FIGURE 4-19 Selecting an Empty ASP.NET project template in Visual Studio 2013.

Visual Studio then create an empty ASP.NET Web solution for you. Next, right-click the project and select Add from the shortcut menu as shown in Figure 4-20.

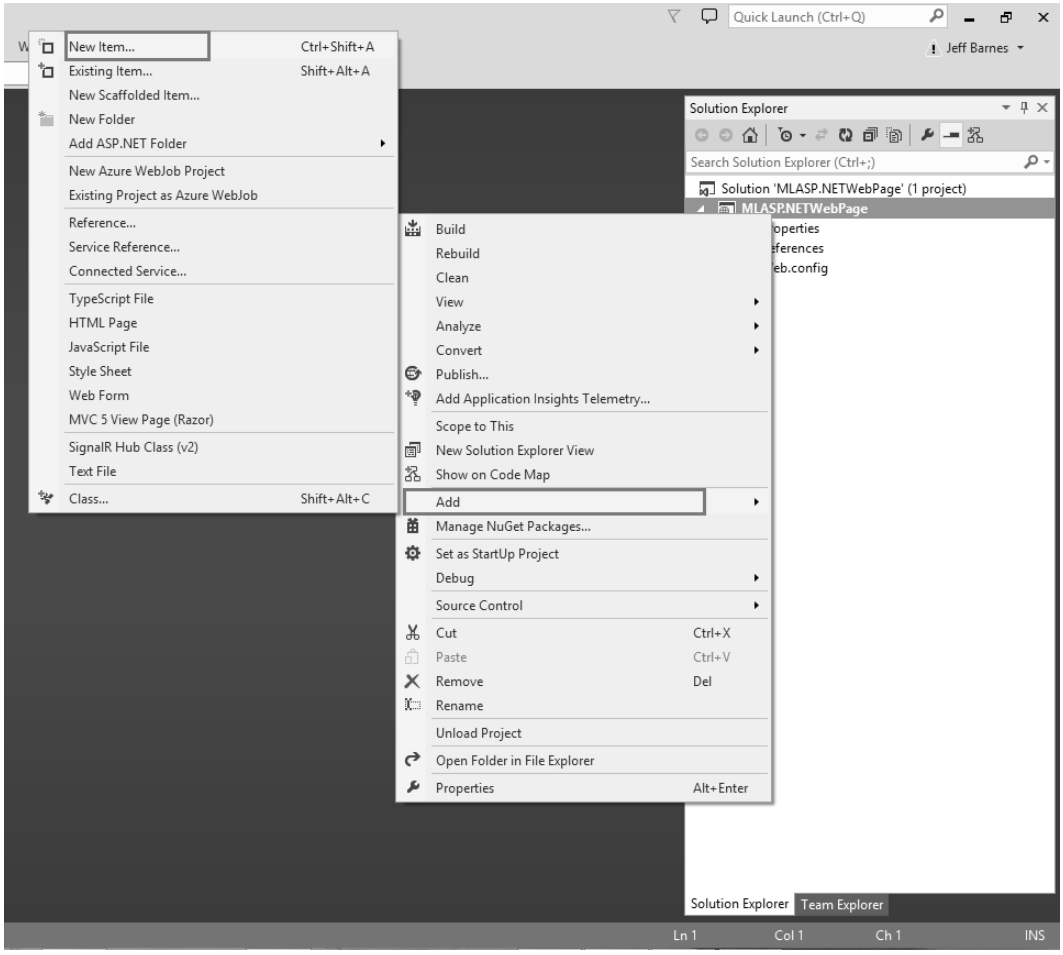


FIGURE 4-20 Adding a new item to the Visual Studio 2013 project.

Choose a Web Form and name the page Default.aspx as shown in Figure 4-21.

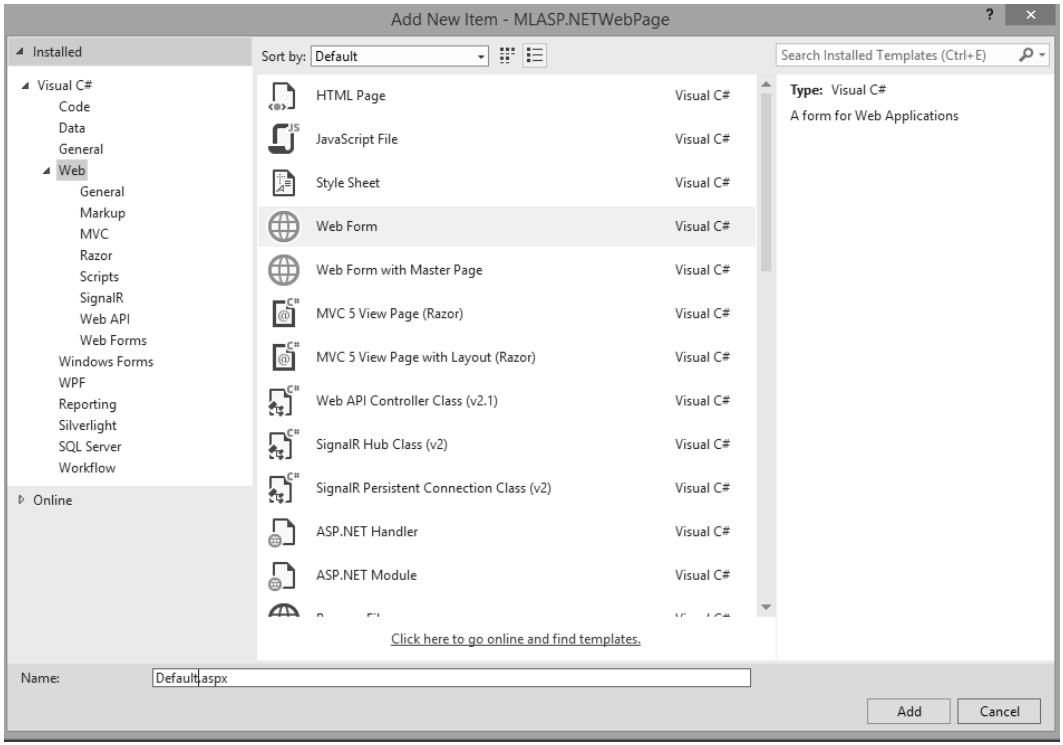


FIGURE 4-21 Selecting a Web Form item to add to the Visual Studio project.

Making it easier to test our Azure Machine Learning web service

Next, we modify the ASPX page to create a webpage layout that uses input controls like drop-down list boxes, as shown in Figure 4-22. This provides the user a valid list of options for each parameter required for the inputs to the Azure Machine Learning web service.

You can get a listing of all of the allowable values for each parameter from the original location where we obtained our sample dataset, at the UCI Machine Learning Repository from the link at http://archive.ics.uci.edu/ml/datasets/Census+Income.

```
<label>Education:</label>
 <asp:DropDownList ID="Education" class="lbox" runat="server" AppendDataBoundItems="true" DataTextField="Company_Name" DataValueField="id">
       <asp:ListItem Text="Doctorate" Value="Doctorate" />
       <asp:ListItem Text="Masters" Value="Masters" />
       <asp:ListItem Text="Bachelors" Value="Bachelors" />
       <asp:ListItem Text="Some-college" Value="Some-college" />
       <asp:ListItem Text="HS-grad" Value="HS-grad" />
       <asp:ListItem Text="Prof-school" Value="Prof-school" />
       <asp:ListItem Text="Assoc-acdm" Value="Assoc-acdm" />
       <asp:ListItem Text="Assoc-voc" Value="Assoc-voc" />
       <asp:ListItem Text="12th" Value="12th" />
       <asp:ListItem Text="11th" Value="11th" />
       <asp:ListItem Text="10th" Value="10th" />
       <asp:ListItem Text="9th" Value="9th" />
       <asp:ListItem Text="7th-8th" Value="7th-8th" />
       <asp:ListItem Text="5th-6th" Value="5th-6th" />
       <asp:ListItem Text="1st-4th" Value="1st-4th" />
       <asp:ListItem Text="Preschool" Value="Preschool" />
</asp:DropDownList>
```

FIGURE 4-22 Default.aspx file showing the markup for a drop-down list box for selecting education level.

We end up with a much more user-friendly interface that facilitates the quick input of various (and valid) parameter input values to test our Azure Machine Learning web service. Figure 4-23 shows a screenshot of the new user interface.

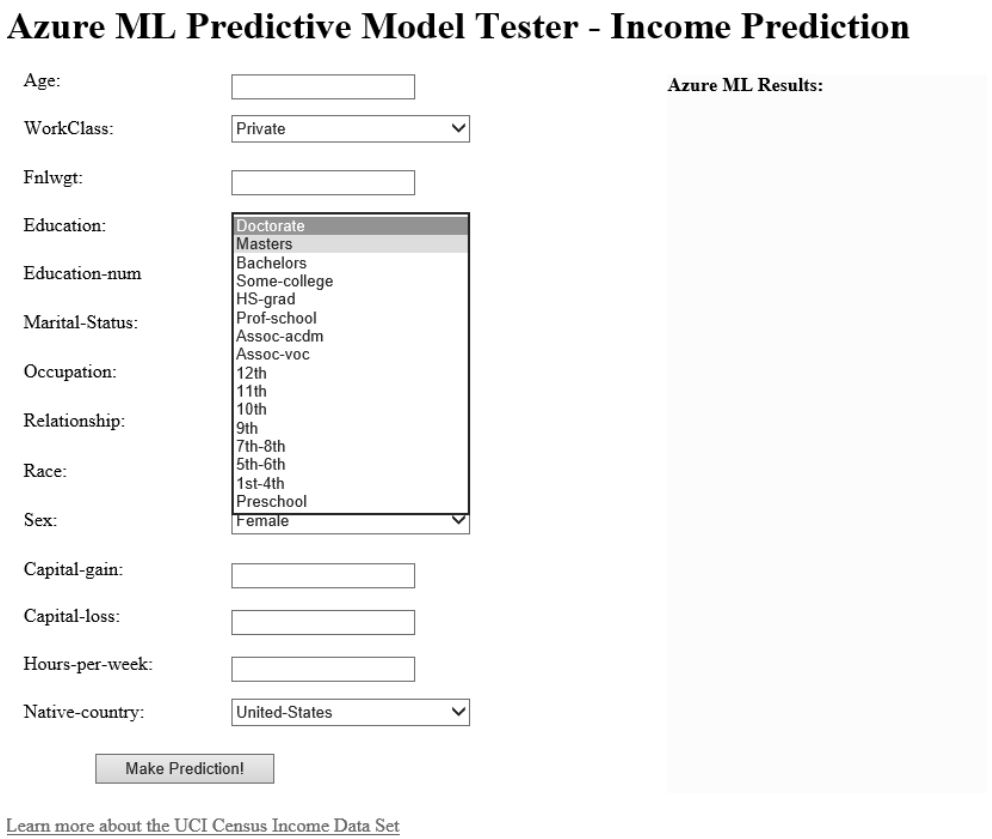


FIGURE 4-23 ASP.NET web form showing a drop-down list box for selecting education levels.

Validating the user input

To further validate our user-provided input, we can also insert client-side JavaScript code, as shown in Figure 4-24, to further validate the input values, before we call our Azure Machine Learning web service.

```
<script type="text/javascript">
    function dataValid() {
        //Validate Age
        var inAge = document.getElementById("age").value;
        if ((inAge < 0) || (inAge > 110))
        {
                alert("Please an Age between 0 - 110");
                return false;
        }

        //Validate Fnlwgt
        var inFnlwgt = document.getElementById("Fnlwgt").value;
        if ((inFnlwgt < 12285) || (inFnlwgt > 1484705)) {
            alert("Please a Fnlwgt between 12285 - 1484705");
            return false;
        }

        //Validate EducationNum
        var inEducationNum = document.getElementById("EducationNum").value;
        if ((inEducationNum < 1) || (inEducationNum > 16)) {
            alert("Please a EducationNum between 1 - 16");
            return false;
        }
        //Validate capitalgain
        var inCapitalGain = document.getElementById("capitalgain").value;
        if ((inCapitalGain < 0) || (inCapitalGain > 99999)) {
            alert("Please a CapitalGain between 0 - 99999");
            return false;
        }

        //Validate capitalloss
        var inCapitalLoss = document.getElementById("capitalloss").value;
        if ((inCapitalLoss < 0) || (inCapitalLoss > 4356)) {
            alert("Please a CapitalLoss between 0 - 4356");
            return false;
        }
        //Validate hoursperweek
        var inHoursPerWeek = document.getElementById("hoursperweek").value;
        if ((inHoursPerWeek < 1) || (inHoursPerWeek > 99)) {
            alert("Please a Hours Per Week between 1 - 99");
            return false;
        }
```

FIGURE 4-24 Client-side JavaScript to validate the input fields for the ASP.NET web form page.

When the user clicks the Make Prediction command button on the ASP.NET web form page, we simply populate the Values string array with the various input values from the webpage as inputs, as shown in Figure 4-25.

```
// Extract Input Values
string inAge = this.age.Text;
string inWorkClass = this.Workclass.SelectedItem.ToString();
string infnlwgt = this.Fnlwgt.Text;
string inEducation = this.Education.SelectedItem.ToString();
string inEducationNum = this.EducationNum.Text;
string inMaritalStatus = this.MaritalStatus.SelectedItem.ToString();
string inOccupation = this.Occupation.SelectedItem.ToString();
string inRelationship = this.Relationship.SelectedItem.ToString();
string inRace = this.Race.SelectedItem.ToString();
string inSex = this.Sex.SelectedItem.ToString();
string inCapitalGain = this.capitalgain.Text;
string inCapitalLoss = this.capitalloss.Text;
string inHoursPerWeek = this.hoursperweek.Text;
string inNativeCountry = this.NativeCountry.SelectedItem.ToString();
using (var client = new HttpClient())
{
    var scoreRequest = new
    {   Inputs = new Dictionary<string, StringTable>() {
            { "input1",
                new StringTable()
                {   ColumnNames = new string[] {"age", "workclass", "fnlwgt", "education", "education-num
                    //Populate Input Parameters with input values
                    Values = new string[,] {  {
                                                inAge,
                                                inWorkClass,
                                                infnlwgt,
                                                inEducation,
                                                inEducationNum,
                                                inMaritalStatus,
                                                inOccupation,
                                                inRelationship,
                                                inRace,
                                                inSex,
                                                inCapitalGain,
                                                inCapitalLoss,
                                                inHoursPerWeek,
                                                inNativeCountry,
                                                "0"
                                            }
                                        }
                                    }
                                }
                            },
```

FIGURE 4-25 Capturing the user inputs in the ASP.NET web form application and assembling the payload to call the Azure Machine Learning web service.

We can then invoke our call to the Azure Machine Learning web service using a similar code block as the one from the C# console application, as shown in Figure 4-26.

```
private async Task InvokeAzureMLRequestResponseService(ScoreRequest scoreRequest)
{
    using (var client = new HttpClient())
    {
        const string apiKey = "OkpG3ZgfMnvUQbWXQ7hRK7YzEUPznmw0ryzVWChR9t/r4uN+lHyTJkOiMH2xt5/qGBGsKzgJ7IpUbmm4hMyZFg==";
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", apiKey);
        client.BaseAddress = new Uri("https://ussouthcentral.services.azureml.net/workspaces/a3be003db95045198c695a070456c824/services/82e999
        HttpResponseMessage response = await client.PostAsJsonAsync("", scoreRequest).ConfigureAwait(false);
        if (response.IsSuccessStatusCode)
        {
            string result = await response.Content.ReadAsStringAsync();
            outMLResultData = result;
            DisplayMLResp(result);
        }
        else
        {
            this.MLResultData.Text = "<div>" + response.ToString() + "</div>";
        }
    }
}
```

FIGURE 4-26 Invoking the Azure Machine Learning web service from an ASP.NET client.

Figure 4-27 shows how our completed Azure Machine Learning web service Tester ASP.NET webpage now looks with all the input parameters and easy-to-use drop-down lists on the left side. The right side of the screen shows the results returned from the Azure Machine Learning web service call.

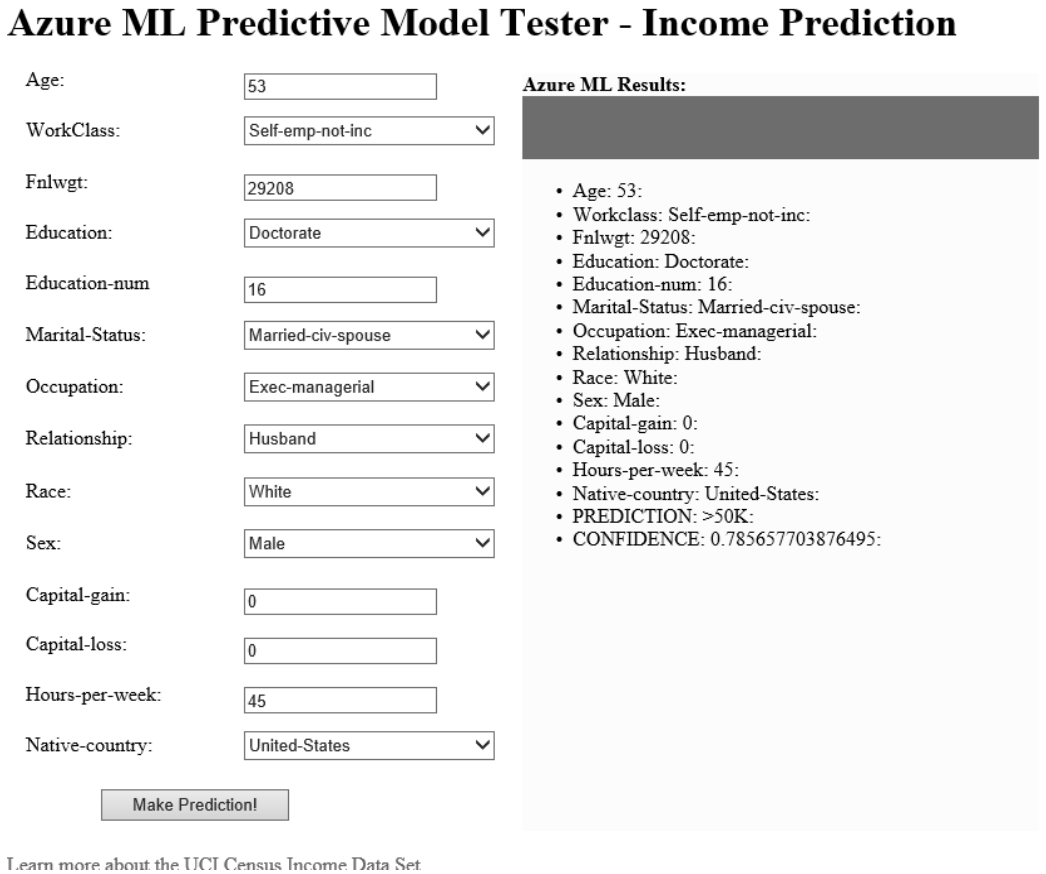


FIGURE 4-27 The finished ASP.NET webpage showing a sample Azure Machine Learning web service call and response.

You can see how this type of simple ASP.NET webpage application can make it far easier to interact with our Azure Machine Learning web service to perform basic testing operations via the use of basic UI controls like list boxes and client-side validation. This approach also gets around the current support restriction for CORS capabilities in Azure Machine Learning web services as the ASP.NET implementation is all done on the server.

The other advantage of this approach is that it allows you to quickly create and deploy an ASP.Net web application to the cloud to help test your Azure Machine Learning prediction model over the Internet. This means your teams can help evaluate your predictive models even faster.

Create a web service using ASP.NET Web API

The HTTP protocol can be used for much more than just serving up webpages. It is also a powerful platform for building APIs that can expose services and data. Almost every development platform today has an HTTP library, so HTTP services can reach a broad range of clients, including browsers, mobile devices, and traditional desktop applications.

Consequently, the ultimate solution to unlocking your brand new Azure Machine Learning prediction web service is to expose it via your own web service. You can maximize the usage across a wide variety of mobile and web clients.

We accomplish the goal of exposing our new Azure Machine Learning prediction model to the largest number of clients possible by creating a new web service using the ASP.NET Web API. The ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers, desktops, and mobile devices. The ASP.NET Web API makes it very easy, and it is an ideal platform for building HTTP REST applications with the .NET Framework.

Let's get started on our new web service by starting Visual Studio, clicking the File menu, selecting New, and then selecting Project. In the Templates pane, select Installed Templates and expand the Visual C# node. Under Visual C#, select Web. In the list of project templates, select ASP.NET Web Application. Name the project MLWebAPI and click OK. See Figure 4-28.

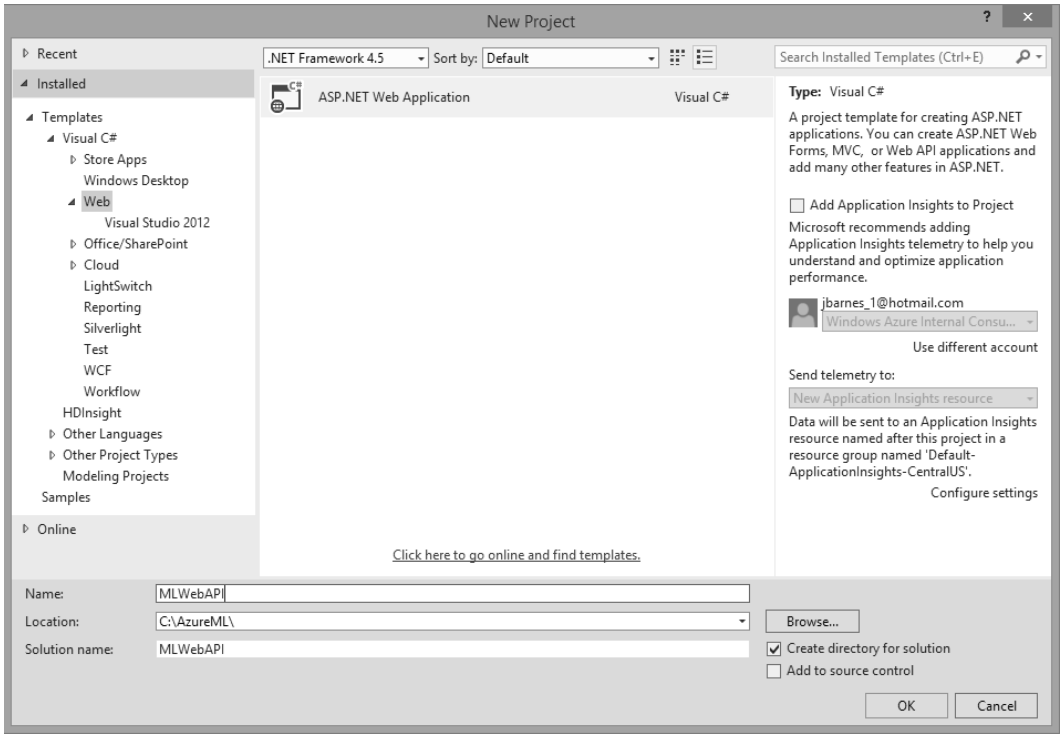


FIGURE 4-28 Creating a new ASP.NET Web API project in Visual Studio 2013.

Next, in the New ASP.NET Project dialog box, select the Empty project template. Under Add Folders And Core References For", select the Web API check box, as shown in Figure 4-29. Click OK.

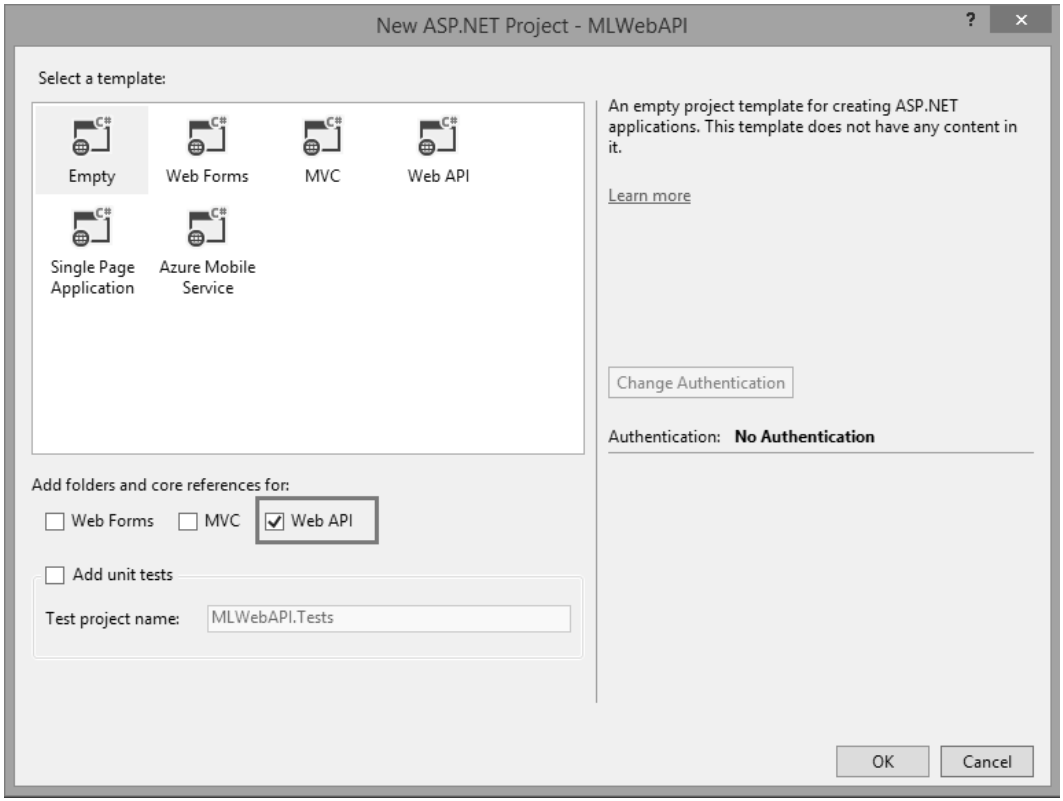


FIGURE 4-29 Selecting the Visual Studio 2013 options for an empty template and including support for Web API.

Visual Studio 2013 will then create for you a new, blank solution with folders for Controllers and Models along with a WebAPIConfig.cs file in the App_Start folder (see Figure 4-30).

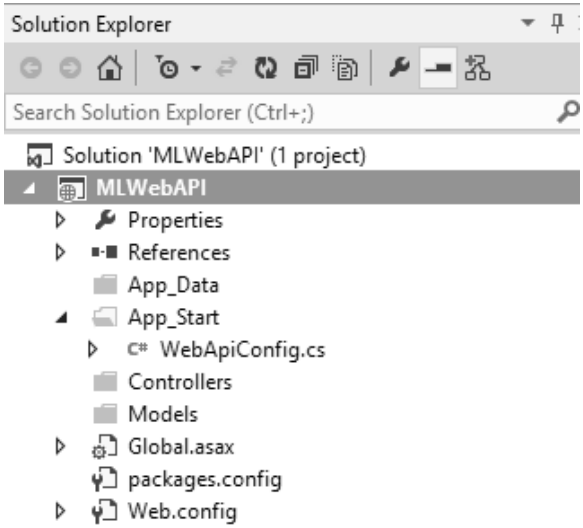


FIGURE 4-30 The Visual Studio project folder structure for a Web API solution.

Let's start by first creating a new model to define the output of our new Web API web service call that will invoke our Azure Machine Learning web service. Right-click the Models folder and then select Add/New Item, as shown in Figure 4-31.

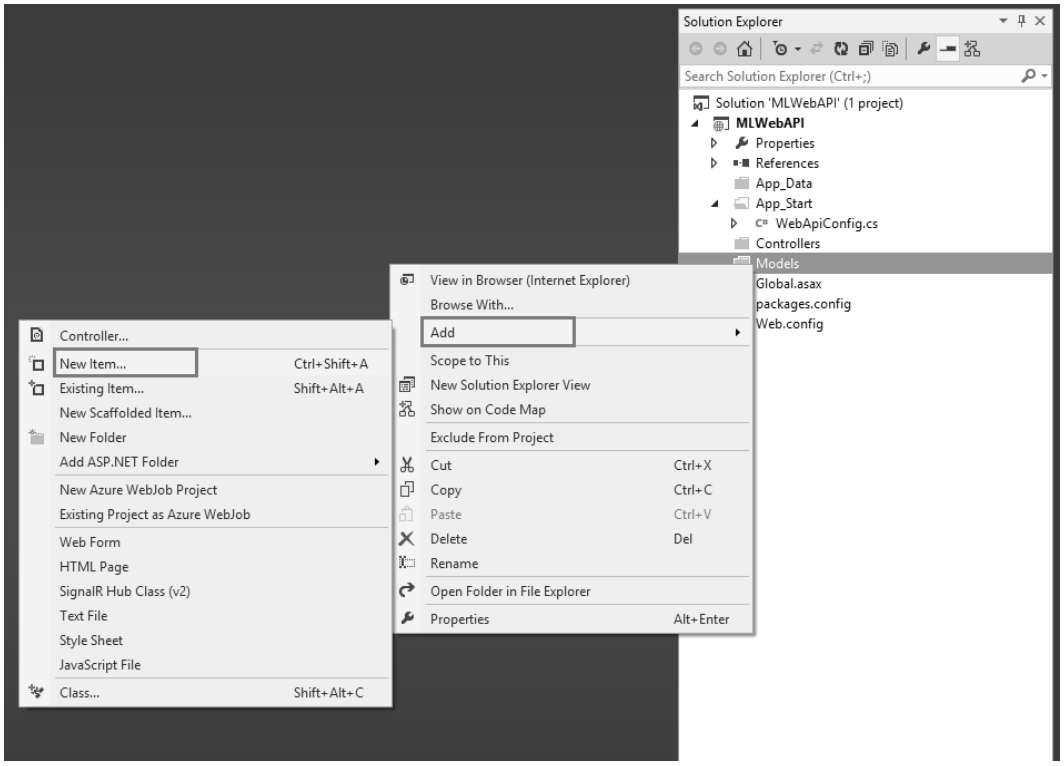


FIGURE 4-31 Adding a new item to the Visual Studio 2013 project.

Next, select Class, as shown in Figure 4-32, and name the file IncomePredictionResults.cs.

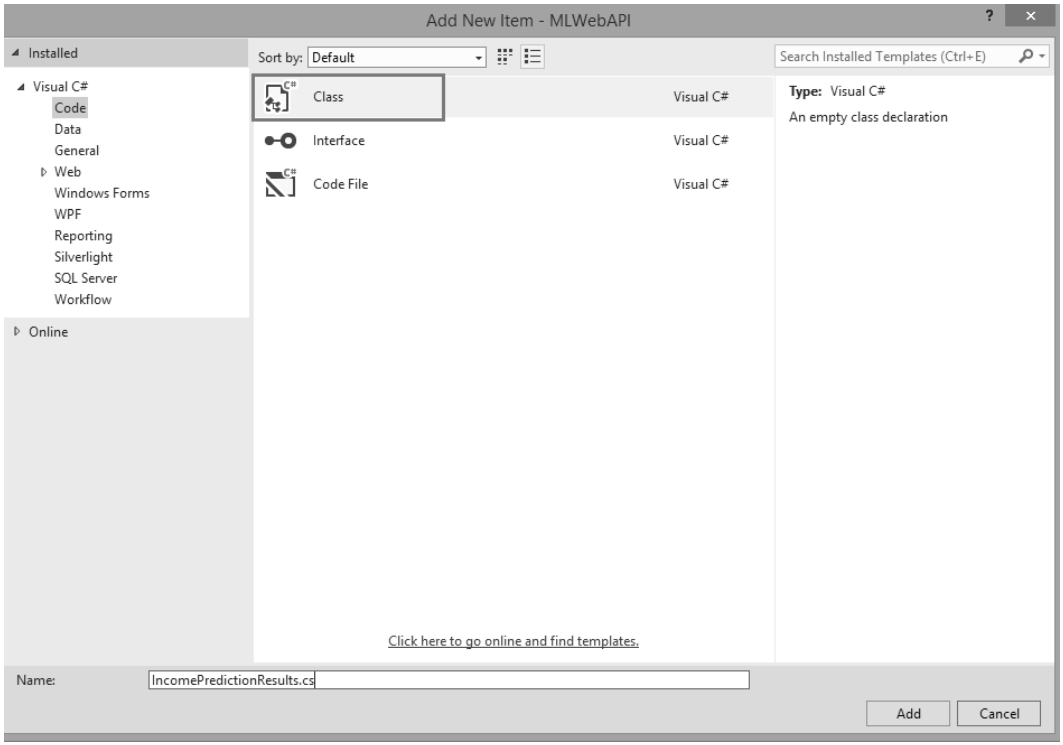


FIGURE 4-32 Adding a new class to the Visual Studio 2013 Web API project.

This creates a new, empty class that we can use to define the result fields of our Web API web service call in JSON format. We will populate this new class with the fields for all of the existing input parameters for our Azure Machine Learning web service call plus three additional fields for the following (see Figure 4-33):

- **MLPrediction** This field contains the prediction results (<=50k or >50K) returned from the call to the Azure Machine Learning web service.
- **MLConfidence** This field contains the confidence factor returned from the call to the Azure Machine Learning web service.
- **MLResponseTime** This is an additional field that we will populate as part of our web service to provide performance metrics on how long the actual call to the Azure Machine Learning web service took to return the prediction results.

```
public class IncomePredictionResults
{
    1 reference
    public string Age { get; set; }
    1 reference
    public string WorkClass { get; set; }
    1 reference
    public string Fnlwgt { get; set; }
    1 reference
    public string Education { get; set; }
    1 reference
    public string EducationNum { get; set; }
    1 reference
    public string MaritalStatus { get; set; }
    1 reference
    public string Occupation { get; set; }
    1 reference
    public string Relationship { get; set; }
    1 reference
    public string Race { get; set; }
    1 reference
    public string Sex { get; set; }
    1 reference
    public string CapitalGain { get; set; }
    1 reference
    public string CapitalLoss { get; set; }
    1 reference
    public string HoursPerWeek { get; set; }
    1 reference
    public string NativeCountry { get; set; }
    2 references
    public string MLPrediction { get; set; }
    1 reference
    public string MLConfidence { get; set; }
    1 reference
    public string MLResponseTime { get; set; }
}
```

FIGURE 4-33 C# class data structure for the returned results from the Web API web service.

In ASP.NET Web API terms, a controller is an object that handles HTTP requests. We'll add a new Controller class to our project by right-clicking the Controller folder in the project solution and then selecting Add and then Controller as shown in Figure 4-34.

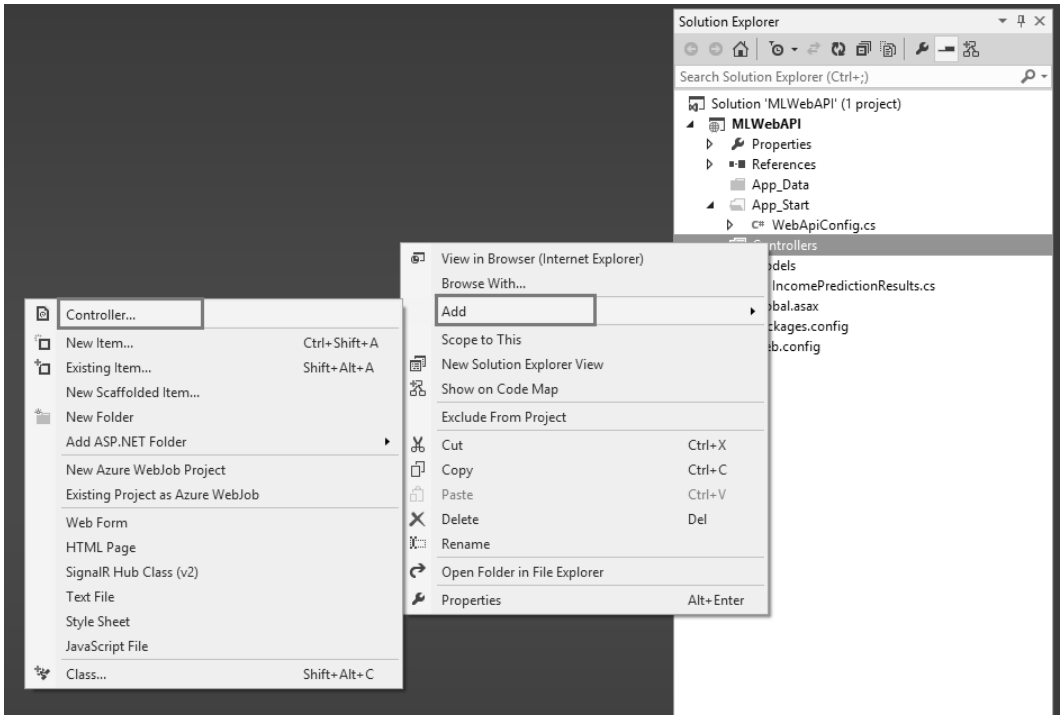


FIGURE 4-34 Adding a new Controller class to the Visual Studio 2013 Web API project.

Next, select the Web API 2 Controller – Empty option as shown in Figure 4-35.

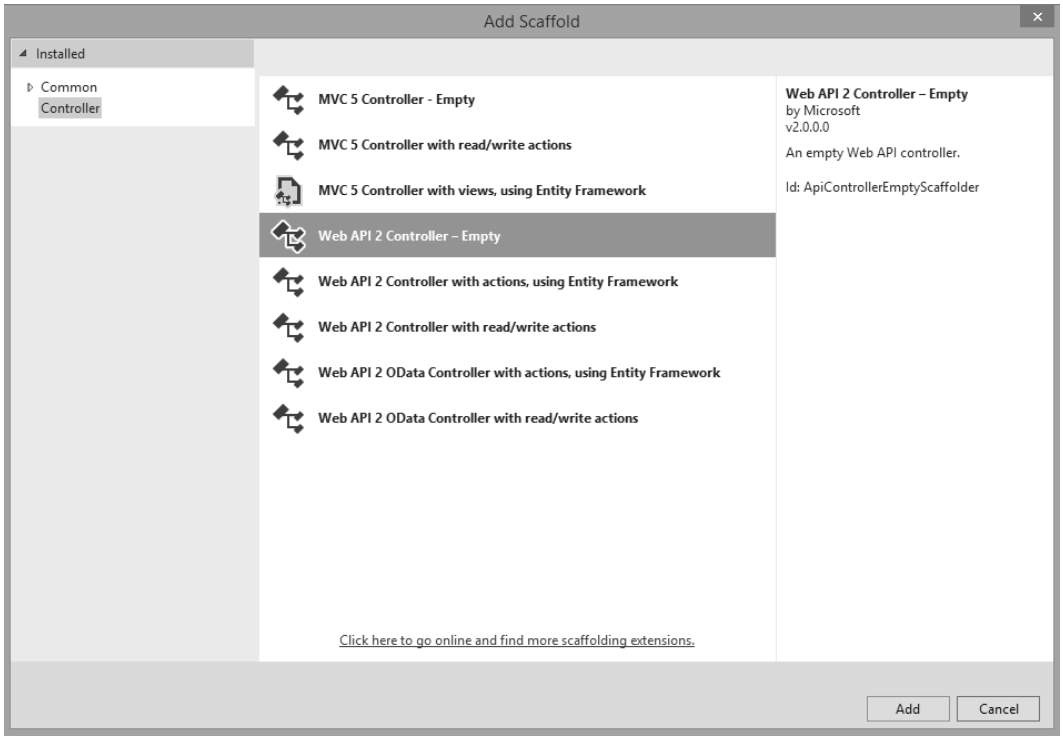


FIGURE 4-35 Adding the empty scaffolding for a Web API 2 controller.

When prompted for the new controller name, enter **IncomePredictionController** and then click Add as shown in Figure 4-36.

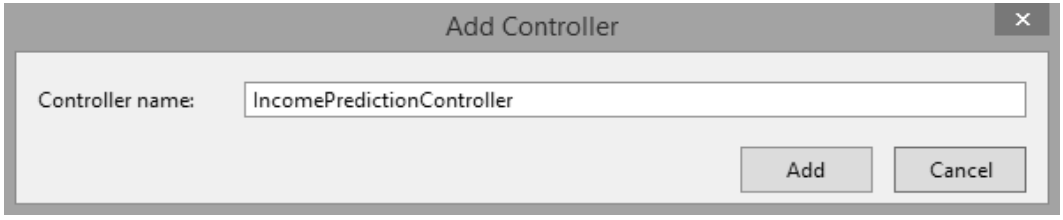


FIGURE 4-36 Adding and naming the new IncomePredictionController controller to the project.

This creates a new, empty Web API controller class that we can use for implementing the processing logic and functions required to call the Azure Machine Learning web service. Before we populate that code, let's take care of a few prerequisites.

Enabling CORS support

Browser security prevents a webpage from making AJAX requests to another domain. This restriction is called the same-origin policy, and it prevents a malicious site from reading sensitive data from another site. However, sometimes you might want to let other sites call your web API.

CORS is a W3C web standard that allows a server to relax the same-origin policy. Using CORS, a server can explicitly allow some cross-origin requests and reject others.

One of the key reasons that we are creating our own web service to invoke the Azure Machine Learning web service is to bypass the current Azure Machine Learning restrictions on the use of CORS. As stated earlier, the Azure Machine Learning web services currently only allow for the services to be invoked from a desktop app or server-side web application. To expose and use CORS in our own web service, we need to first install a NuGet package into our project.

To do this, start by right-clicking References for the project and selecting Manage NuGet Packages as shown in Figure 4-37.

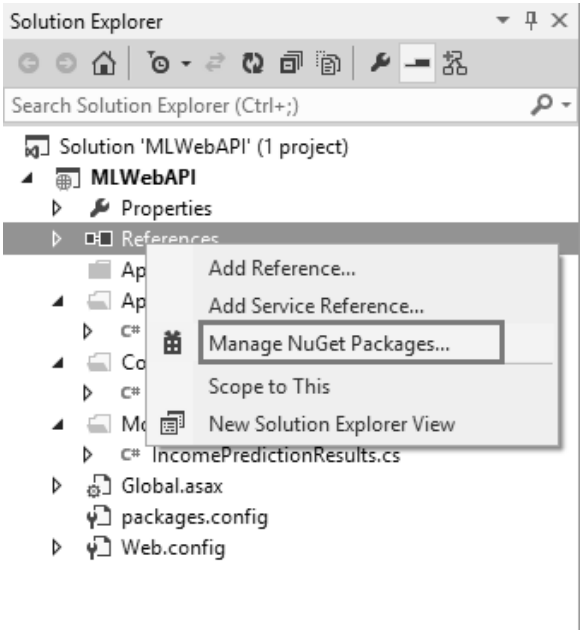


FIGURE 4-37 Selecting the Manage NuGet Packages option in Visual Studio 2013.

Once the NuGet Package Manager starts, select Online on the left side and then type **Microsoft ASP.NET Web API** into the search box in the top right corner. Scroll down through the results until you see the package for Microsoft ASP.NET 2.2 Web API Cross-Origin Resource Sharing (CORS) In ASP.NET Web API as shown in Figure 4-38.

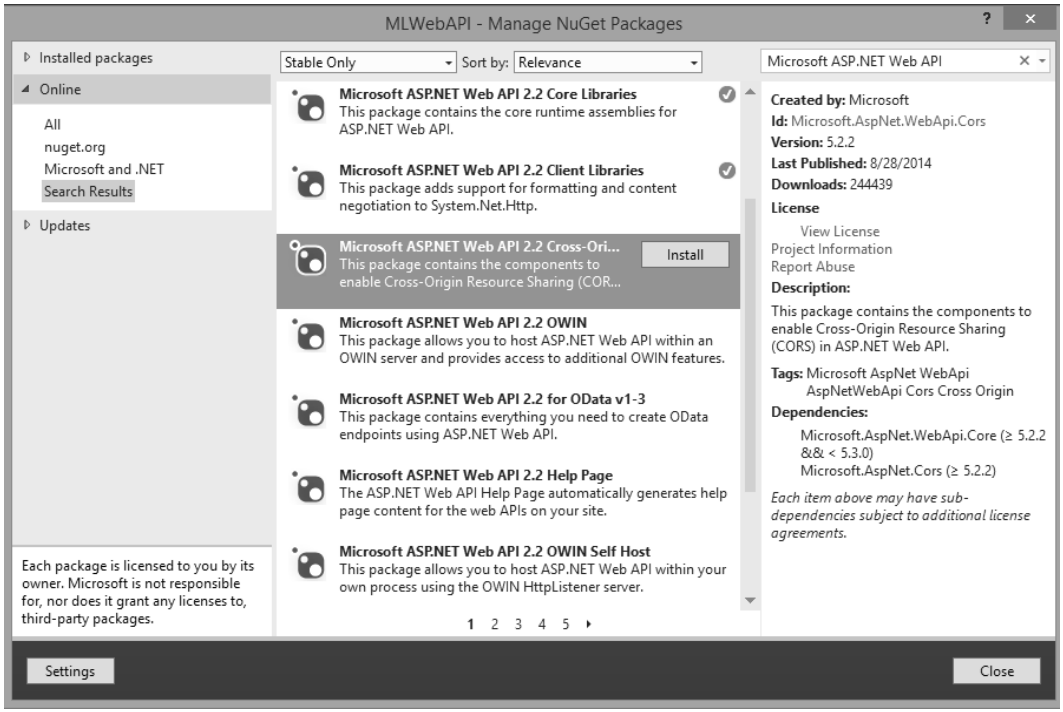


FIGURE 4-38 Installing the NuGet package for ASP.NET WEB API CORS support.

Select the appropriate NuGet package, read and accept the license terms, and install the package into the Visual Studio project. This is a key piece of the solution to allow support for CORS in our web service. Next, in the App_Start folder, open the WebApiConfig.cs file and insert the code snippet to enable CORS support in our web service as shown in Figure 4-39.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

namespace MLWebAPI
{
    0 references
    public static class WebApiConfig
    {
        0 references
        public static void Register(HttpConfiguration config)
        {
            // ENABLE Cross-Origin Resource Sharing (CORS)
            config.EnableCors();
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "ActionApi",
                routeTemplate: "api/{controller}/{action}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

        }
    }
}
```

FIGURE 4-39 Enabling CORS support in the WebAPIConfig class.

The other task necessary to enable CORS support in our web service is to decorate the IncomePredictionController class with the attributes shown in Figure 4-40.

```
using System.Web.Script.Serialization;
using System.IO;
using Newtonsoft.Json;

namespace MLWebAPI.Controllers
{
    // ALLOW *ALL* Cross-Origin Resource Sharing CORS Requests
    [EnableCors("*", "*", "*")]
    0 references
    public class IncomePredictionController : ApiController...
}
```

FIGURE 4-40 Enabling CORS support in the Web API Controller class.

Figure 4-41 shows the high-level structure of our Web API Controller class to handle web requests from our clients.

```
namespace MLWebAPI.Controllers
{
    // ALLOW *ALL* Cross-Origin Resource Sharing CORS Requests
    [EnableCors("*", "*", "*")]
    0 references
    public class IncomePredictionController : ApiController
    {
        2 references
        public class StringTable
        {
            1 reference
            public string[] ColumnNames { get; set; }
            1 reference
            public string[,] Values { get; set; }
        }
        public static string outMLResultData = "";

        [HttpGet]
        0 references
        public async Task<IncomePredictionResults> GetPrediction()...
        1 reference
        private static IncomePredictionResults ParseMLResponse(string result)...

    }
}
```

FIGURE 4-41 The IncomePredictionController C# class.

The StringTable class is used to package the input data parameters into string arrays of name–value pairs before invoking the Azure Machine Learning web service for income predictions, as shown in Figure 4-42.

```
public class StringTable
{
    1 reference
    public string[] ColumnNames { get; set; }
    1 reference
    public string[,] Values { get; set; }
}
```

FIGURE 4-42 The StringTable class.

Processing logic for the Web API web service

In our sample Web API web service, all the work is done in the asynchronous method call named GetPrediction(). Here is the high-level logic for processing our Azure Machine Learning web service request:

- Prepare a new IncomePredictionResults data structure for returning to the caller in JSON format.

- Read and parse the parameters passed in from the HTTP GET request.
- Set up a new StringTable data structure for invoking the Azure Machine Learning web service.
- Populate the StringTable data structure with the input parameters passed in to our web service from the HTTP GET request.
- Start a stopwatch object to start timing the Azure Machine Learning web service request.
- Make the Azure Machine Learning web service request (asynchronously).
- When the Azure Machine Learning web service call returns, stop the stopwatch object and save the elapsed time (in milliseconds) that it took to call the Azure Machine Learning web service and return the results.
- Populate the output data structure with the results of the Azure Machine Learning web service call.
- Return the results to the caller for the Web API web service request.

The following is the C# code for our Income Prediction Web API Controller class.

```
public async Task<IncomePredictionResults> GetPrediction()
{
//Prepare a new ML Response Data Structure for the results
IncomePredictionResults incomePredResults = new IncomePredictionResults();

//Parse the input parameters from the request
NameValueCollection nvc = HttpUtility.ParseQueryString(Request.RequestUri.Query);

//Validate Number of Input parameters (TODO: Add more validations)
if (nvc.Count < 14) { }

// Extract Input Values
string inAge = nvc[0];
string inWorkClass = nvc[1];
string infnlwgt = nvc[2];
string inEducation = nvc[3];
string inEducationNum = nvc[4];
string inMaritalStatus = nvc[5];
string inOccupation = nvc[6];
string inRelationship = nvc[7];
string inRace = nvc[8];
string inSex = nvc[9];
string inCapitalGain = nvc[10];
string inCapitalLoss = nvc[11];
string inHoursPerWeek = nvc[12];
string inNativeCountry = nvc[13];

using (var client = new HttpClient())
{
var scoreRequest = new
```

```
{
Inputs = new Dictionary<string, StringTable>() {
   { "input1",
      new StringTable()
      {  ColumnNames = new string[] {"age", "workclass", "fnlwgt", "education", "education-num",
"marital-status", "occupation", "relationship", "race", "sex", "capital-gain", "capital-loss",
"hours-per-week", "native-country", "income"},
         //Populate Input Parameters with input values
         Values = new string[,] {  {
                                    inAge,
                                    inWorkClass,
                                    infnlwgt,
                                    inEducation,
                                    inEducationNum,
                                    inMaritalStatus,
                                    inOccupation,
                                    inRelationship,
                                    inRace,
                                    inSex,
                                    inCapitalGain,
                                    inCapitalLoss,
                                    inHoursPerWeek,
                                    inNativeCountry,
                                    "0"
                                }
                            }
                        }
                    },
                    },
GlobalParameters = new Dictionary<string, string>()
{
}
};
var sw = new Stopwatch();
const string apiKey =
"Hh8SjZZVxPmoXc4yQl5pIs3sIGEm3Shu71j5hLh/XmFqUampjqdcs4+rOR2KL1WlznG6XfvpRyoFaV5jc2UIYQ==";
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", apiKey);
client.BaseAddress = new
Uri("https://ussouthcentral.services.azureml.net/workspaces/77ee4cb8552b4362b9080caa76b64cb7/ser
vices/43d81b5deb844108b33c42433bfae69d/execute?api-version=2.0&details=true");

//Time the ML Web Service Call
sw.Start();
HttpResponseMessage response = await client.PostAsJsonAsync("",
scoreRequest).ConfigureAwait(false);
sw.Stop();
string elapsed = sw.Elapsed.TotalSeconds.ToString();

//Check Status of Azure ML Web Service Call
if (response.IsSuccessStatusCode)
{
//Read the HTTP response
string MLResp = await response.Content.ReadAsStringAsync();//.ConfigureAwait(false);
```

```
//Parse ML Web Service Response and return a populated IncomePredictionResults response record
incomePredResults = ParseMLResponse(MLResp);

//Update for ML Service Response Time
incomePredResults.MLResponseTime = elapsed;
}
else
{
incomePredResults.MLPrediction = response.ReasonPhrase.ToString();
}

client.Dispose();
//return Ok(incomePredResults);
return incomePredResults;
}
}
```

The following code is for a helper method called ParseMLResponse() that will parse results from the Azure Machine Learning web service call and populate the IncomePredictionResults data structure with the returned results.

```
private static IncomePredictionResults ParseMLResponse(string result)
{
   var cleaned = result.Replace("\"", string.Empty);
   cleaned = cleaned.Replace("[", string.Empty);
   cleaned = cleaned.Replace("]", string.Empty);
   string[] mlResultsArr = cleaned.Split(",".ToCharArray());

   IncomePredictionResults incomePredResult = new IncomePredictionResults();
   for (int i = 0; i < mlResultsArr.Length; i++)
   {
      switch (i)
      {
         case 0:
            incomePredResult.Age = mlResultsArr[i].ToString(); break;
         case 1:
            incomePredResult.WorkClass = mlResultsArr[i].ToString(); break;
         case 2:
            incomePredResult.Fnlwgt = mlResultsArr[i].ToString(); break;
         case 3:
            incomePredResult.Education = mlResultsArr[i].ToString(); break;
         case 4:
            incomePredResult.EducationNum = mlResultsArr[i].ToString(); break;
         case 5:
            incomePredResult.MaritalStatus = mlResultsArr[i].ToString(); break;
         case 6:
            incomePredResult.Occupation = mlResultsArr[i].ToString(); break;
         case 7:
            incomePredResult.Relationship = mlResultsArr[i].ToString(); break;
         case 8:
            incomePredResult.Race = mlResultsArr[i].ToString(); break;
         case 9:
```

```
                incomePredResult.Sex = mlResultsArr[i].ToString(); break;
            case 10:
                incomePredResult.CapitalGain = mlResultsArr[i].ToString(); break;
            case 11:
                incomePredResult.CapitalLoss = mlResultsArr[i].ToString(); break;
            case 12:
                incomePredResult.HoursPerWeek = mlResultsArr[i].ToString(); break;
            case 13:
                incomePredResult.NativeCountry = mlResultsArr[i].ToString(); break;
            case 14:
                incomePredResult.MLPrediction = mlResultsArr[i].ToString(); break;
            case 15:
                incomePredResult.MLConfidence = mlResultsArr[i].ToString(); break;
            default:
                break;
        }
    }
    return incomePredResult;
}
```

Now we are ready to build and run our new ASP.NET Web API web service. When you first run the solution, you might notice that the browser returns a bad request. ASP.NET Web API services are invoked via the HTTP URL requests. To call our service, we will append the following API parameters to the end of the base URL:

```
/api/IncomePrediction/GetPrediction?age=53….[the rest of the parameters]
```

The following is a complete sample HTTP call to our ASP.NET Web API web service with all the input parameters populated with valid sample data:

```
http://localhost:24462/api/IncomePrediction/GetPrediction?age=53&workclass=Self-emp-not-inc&fnlw
gt=209642&education=HS-grad&education-num=9&marital-status=Married-civ-spouse&occupation=Exec-ma
nagerial&relationship=Husband&race=White&sex=Male&capital-gain=0&capital-loss=0&hours-per-week=4
5&native-country=United-States
```

Now if we test our new web service with a simple browser request, we will be prompted to download a JSON result file, as shown in Figure 4-43.

FIGURE 4-43 A JSON file is returned to the browser after a web request to the ASP.NET Web API web service with the income prediction results.

If we open the returned JSON file, we see a nicely formatted JSON data payload with all the input parameters along with the results for the Azure Machine Learning web service call, including the amount of time the Azure Machine Learning web service call actually took:

```
{"Age":"53",
"WorkClass":"Self-emp-not-inc",
"Fnlwgt":"209642",
```

```
"Education":"HS-grad",
"EducationNum":"9",
"MaritalStatus":"Married-civ-spouse",
"Occupation":"Exec-managerial",
"Relationship":"Husband",
"Race":"White",
"Sex":"Male",
"CapitalGain":"0",
"CapitalLoss":"0",
"HoursPerWeek":"45",
"NativeCountry":"United-States",
"MLPrediction":">50K",
"MLConfidence":"0.549394130706787",
"MLResponseTime":"1.1816294"
}
```


Now that we have a working ASP.NET Web API web service that can accept and process HTTP GET requests, we can create a simple web client to consume our new ASP.NET Web API web service. One easy way to do this is by creating a simple webpage that uses jQuery Mobile. The jQuery Mobile framework allows you to design a single responsive website or web application that will work on all popular smartphone, tablet, and desktop platforms. You can learn more about jQuery Mobile at http://jquerymobile.com.

Figure 4-44 shows a sample webpage that was built using jQuery Mobile to help facilitate the testing and usage of Azure Machine Learning web services across a multitude of devices.

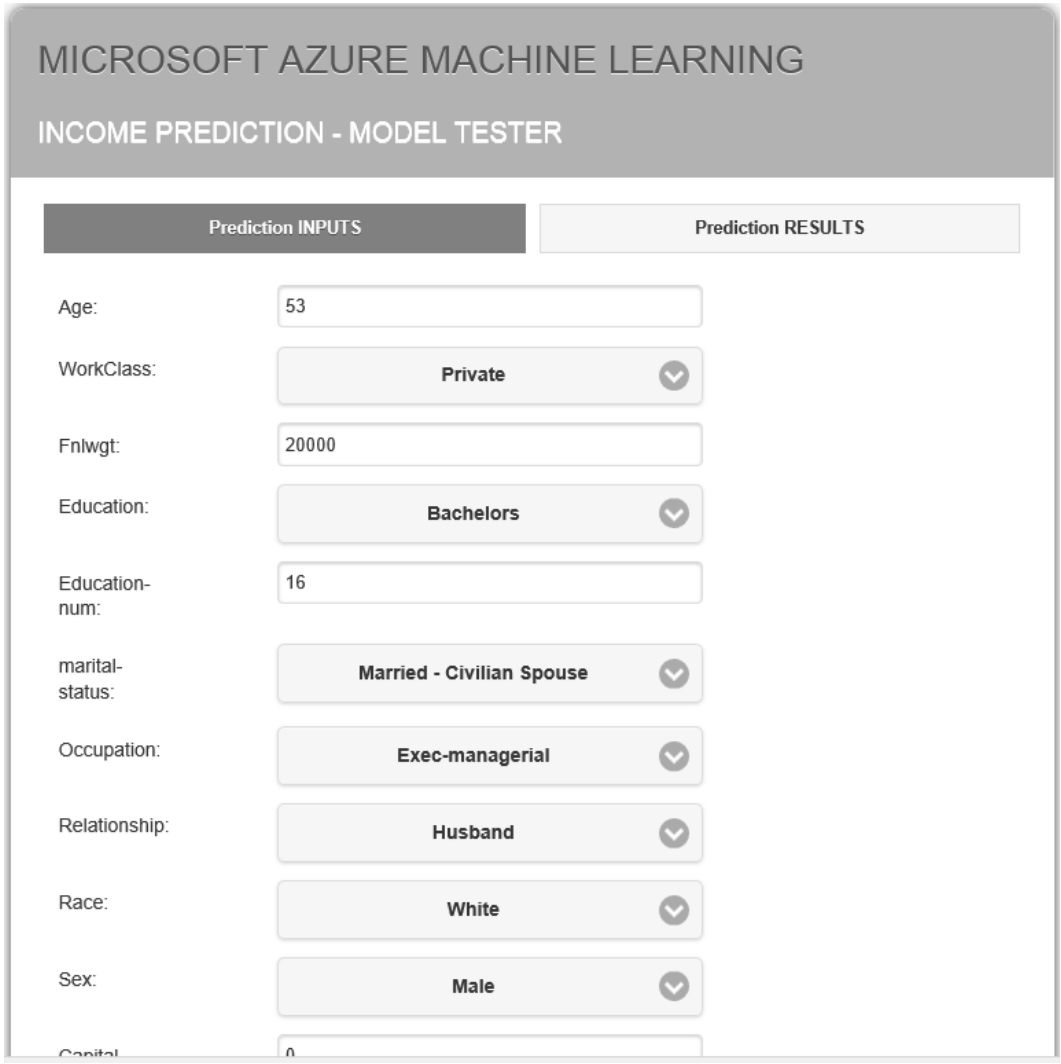


FIGURE 4-44 A jQuery Mobile web application to call our ASP.NET Web API web service.

In Figure 4-45, we can see the Prediction Results tab showing all the original web service call parameters along with the results of the call to our Azure Machine Learning web service for predicting a person's income level based on basic demographic data.

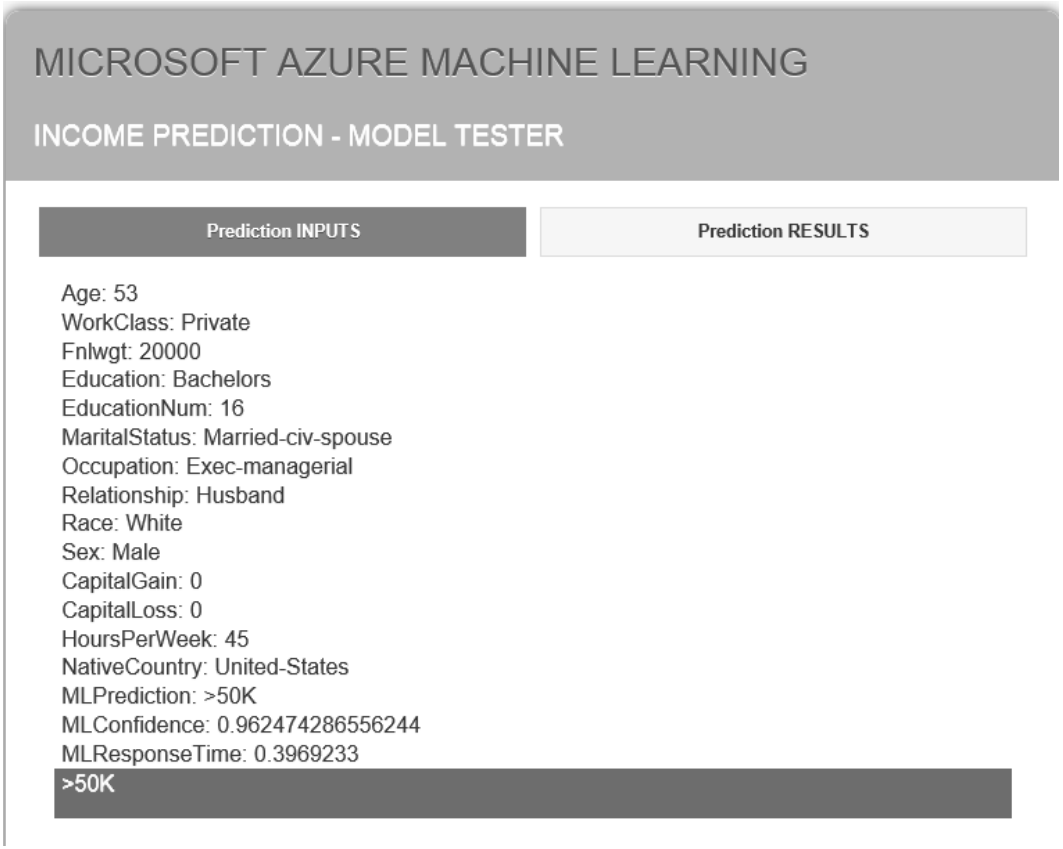


FIGURE 4-45 The results of a call to our new ASP.NET Web API web service displayed in a jQuery Mobile web application.

The following is the JavaScript code behind the Make Prediction button, which makes the call via AJAX to our exposed web service.

```
$("#btnMakePrediction").click(function (e) {
    e.preventDefault();
    jQuery.support.cors = true;

    //Get Input Variables
    var iAge = document.getElementById("Age").value.toString();
    var iWorkclass = document.getElementById("Workclass").value.toString();
    var iFnlwgt = document.getElementById("Fnlwgt").value.toString();
    var iEducation = document.getElementById("Education").value.toString();
    var iEducationnum = document.getElementById("Education-num").value.toString();
    var iMaritalstatus = document.getElementById("marital-status").value.toString();
    var iOccupation = document.getElementById("occupation").value.toString();
    var iRelationship = document.getElementById("relationship").value.toString();
    var iRace = document.getElementById("race").value.toString();
```

```
    var iSex = document.getElementById("sex").value.toString();
    var iCapitalgain = document.getElementById("capital-gain").value.toString();
    var iCapitalloss = document.getElementById("capital-loss").value.toString();
    var iHoursperweek = document.getElementById("hours-per-week").value.toString();
    var iNativecountry = document.getElementById("native-country").value.toString();
    // Make AJAX call to the Web Service
    var jqxhr = $.ajax(
        {
            url: 'http://mlwebservice.azurewebsites.net/api/IncomePrediction/GetPrediction',
            type: "GET",
            data: {
                age: iAge,
                workclass: iWorkclass,
                fnlwgt: iFnlwgt,
                education: iEducation,
                educationnum: iEducationnum,
                maritalstatus: iMaritalstatus,
                occupation: iOccupation,
                relationship: iRelationship,
                race: iRace,
                sex: iSex,
                capitalgain: iCapitalgain,
                capitalloss: iCapitalloss,
                hoursperweek: iHoursperweek,
                nativecountry: iNativecountry
            },
            dataType: "json",
            async: false
        }
        )
        // Process the results of the AJAX call
        .done(function (responseData) {
            var MLStatus = "";
            $("#MLRedResults").html('');
            $.each(responseData, function (index, item) {
                $("#MLRedResults").append("<li>" + index + ": " + item + "</li>");
                if (index == 'MLPrediction')
                {
                    MLStatus = item;
                }
            });
        //Render the results
            $("#MLRedResults").append("<label id='MLPredStatus' >" + MLStatus +
"</label><br/><br/><br/>");
            if (MLStatus == '>50K') {
                $('#MLPredStatus').css({ background: "green" })
            }
            if (MLStatus == '<=50K') {
                $('#MLPredStatus').css({ background: "red" })
            }
            //Display the Results Panel
            $("#tabs").tabs("option", "active", 1);
            $("#tabs").tabs({ active: 1 });
```

```
            return false;
        })
        .fail(function (xhr, status, error) {
            var err = eval("(" + xhr.responseText + ")");
            alert(xhr.statusText);
        })
        .always(function () {
            //alert("complete");
        });

    // Set another completion function for the request above
    jqxhr.always(function () {
        //alert("second complete");
    });
});
```

Summary

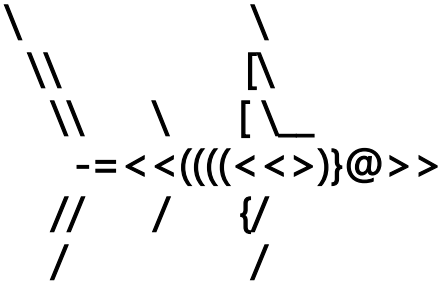
In this chapter, we explored how to interact with Azure Machine Learning web services for our income prediction model. We saw how the sample Azure Machine Learning web service provides some great starting code samples for C#, Python, and R for creating stand-alone clients. To that end, we demonstrated calling the Azure Machine Learning prediction web services from simple clients such as a C# console application and a stand-alone R program.

We learned how to further expose our Azure Machine Learning web service via an ASP.NET Web client to overcome the current Azure Machine Learning limitation with CORS requests, CORS being the limitation of a webpage loaded from one domain that is restricted from making AJAX requests to another domain.

Ultimately, by making use of the ASP.NET Web API, we were able to expose our own web service and overcome the existing Azure Machine Learning CORS restriction. This API enabled the widest variety of desktop, web, and mobile clients to consume and test our machine learning predictive model over the Web. In the end, developing an Azure Machine Learning solution is really only valuable if you quickly and easily expose it to a larger population of test or production environments. Only then will you be able to facilitate feedback loops that are crucial to allowing this technology to surpass simple machine training into the world of continuous learning for machines.

About the author

Jeff A. Barnes is a Cloud Solution Architect (CSA) on the Microsoft Partner Enterprise Architecture Team, where he engages with leading cloud architects and developers to present Microsoft's cloud vision. A 17-year Microsoft veteran, Jeff brings over 30 years of deep technical experience to the CSA role. He typically works with key ISVs and global partners to demonstrate how Microsoft Azure technologies can be best leveraged to meet the current and future demands of an organization transitioning to the cloud. Jeff has deep practical experience in the retail, financial, and manufacturing industries and he is a frequent speaker at Microsoft and third-party events. Jeff resides with his family in Miami, Florida, where his definition of "offshore development" usually equates to "fishing offshore."

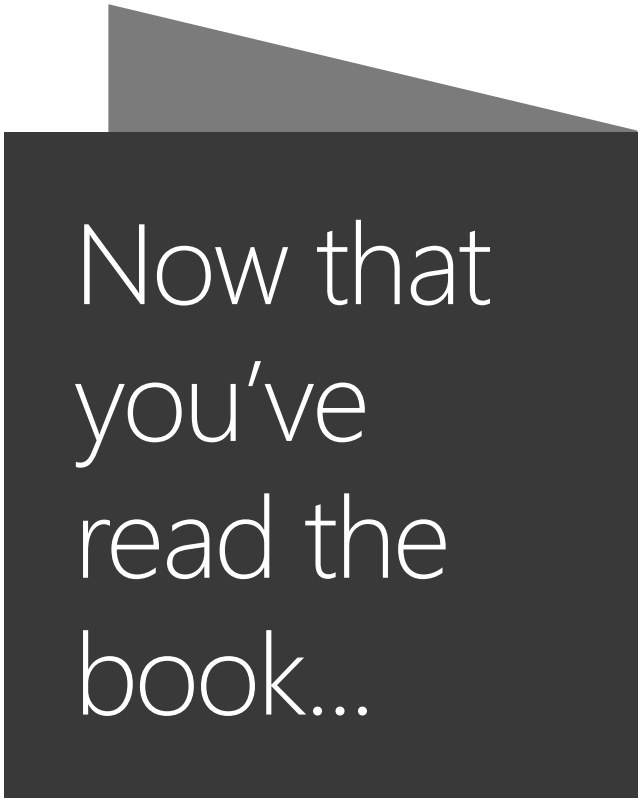


From technical overviews to drilldowns on special topics, get *free* ebooks from Microsoft Press at:

www.microsoftvirtualacademy.com/ebooks

Download your free ebooks in PDF, EPUB, and/or Mobi for Kindle formats.

Look for other great resources at Microsoft Virtual Academy, where you can learn new skills and help advance your career with free Microsoft training delivered by experts.



Tell us what you think!

Was it useful?
Did it teach you what you wanted to learn?
Was there room for improvement?

Let us know at http://aka.ms/tellpress

Your feedback goes directly to the staff at Microsoft Press,
and we read every one of your responses. Thanks in advance!

Microsoft