1.Amaç

- Özelliklerini girdiğimiz otomobilin fiyatının tahmin edilmesi

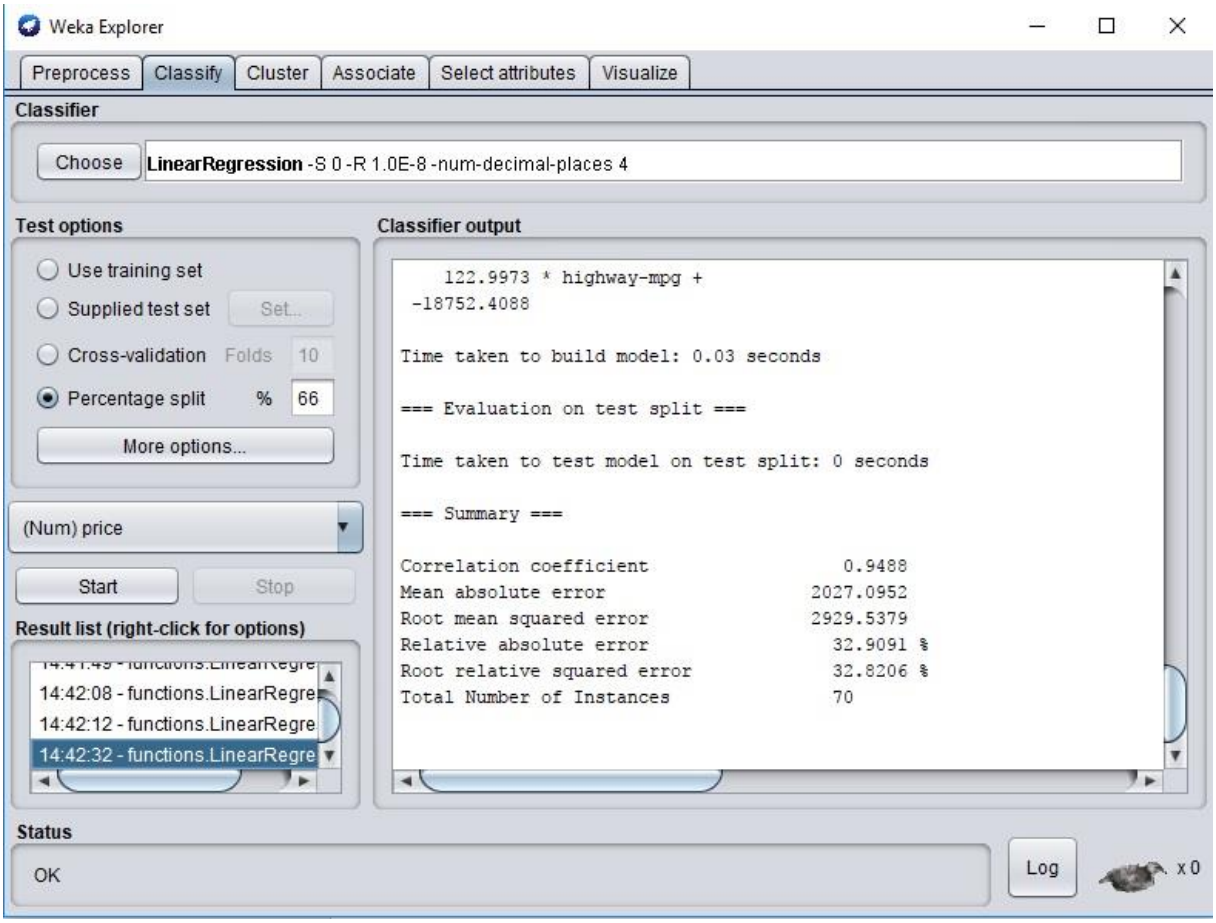
2.Veri Seti

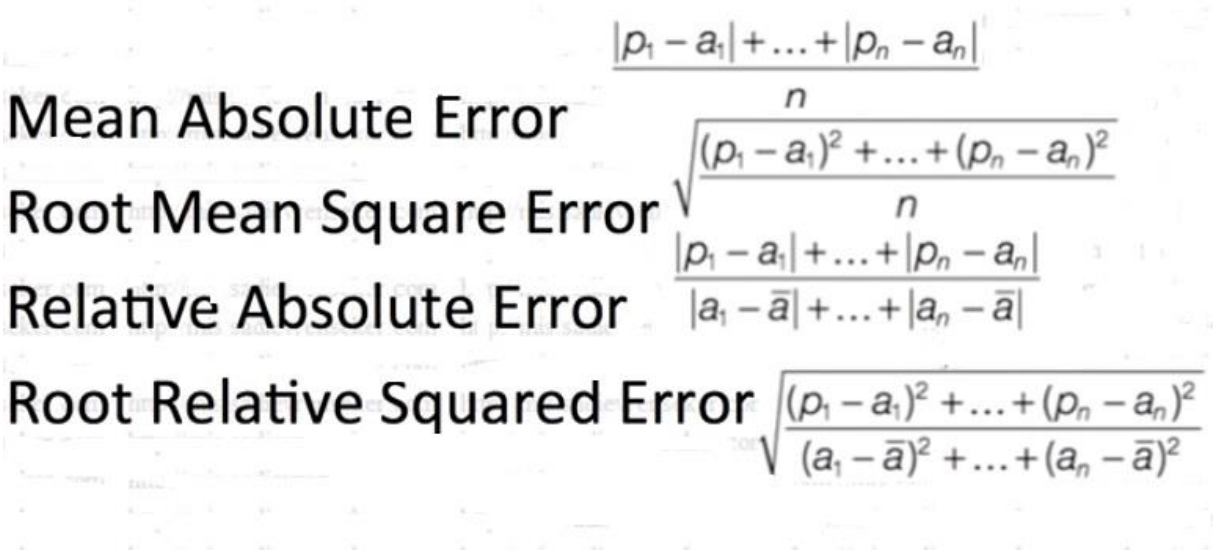
- Veriseti Automobile Data Set. Jeffrey C. Schimmer tarafından UCI a yüklenmiştir. Verisetinde 26 tane feature, 205 tane instance vardır.
- Feature anlamları

Symbol:insurance +3risky -3safe
Normalized-loses: risk/car
Brand:markalar
Fuel:yakıt tipi
Aspiration:motor tipi
No-of-doors:kapı sayısı
Body-style:arabanın şekli
Engine-to-whell: motorun çalıştığı teker
Engine-location:motorun konumu
No-of-cylinder :silindir sayısı
Engine-size:motor boyutu
Fuel-system: yakıt sistemi
Bore:kalibre
Stroke:strok(verilen basınç)
Compression ratio : sıkıştırma katsayısı
Hp:beygir güçü
Peak-rpm: maksimum çevirme kuvveti
City-mpg: şehiriçi hızı
Highway-mpg:otoyol hızı
Price:fiyat

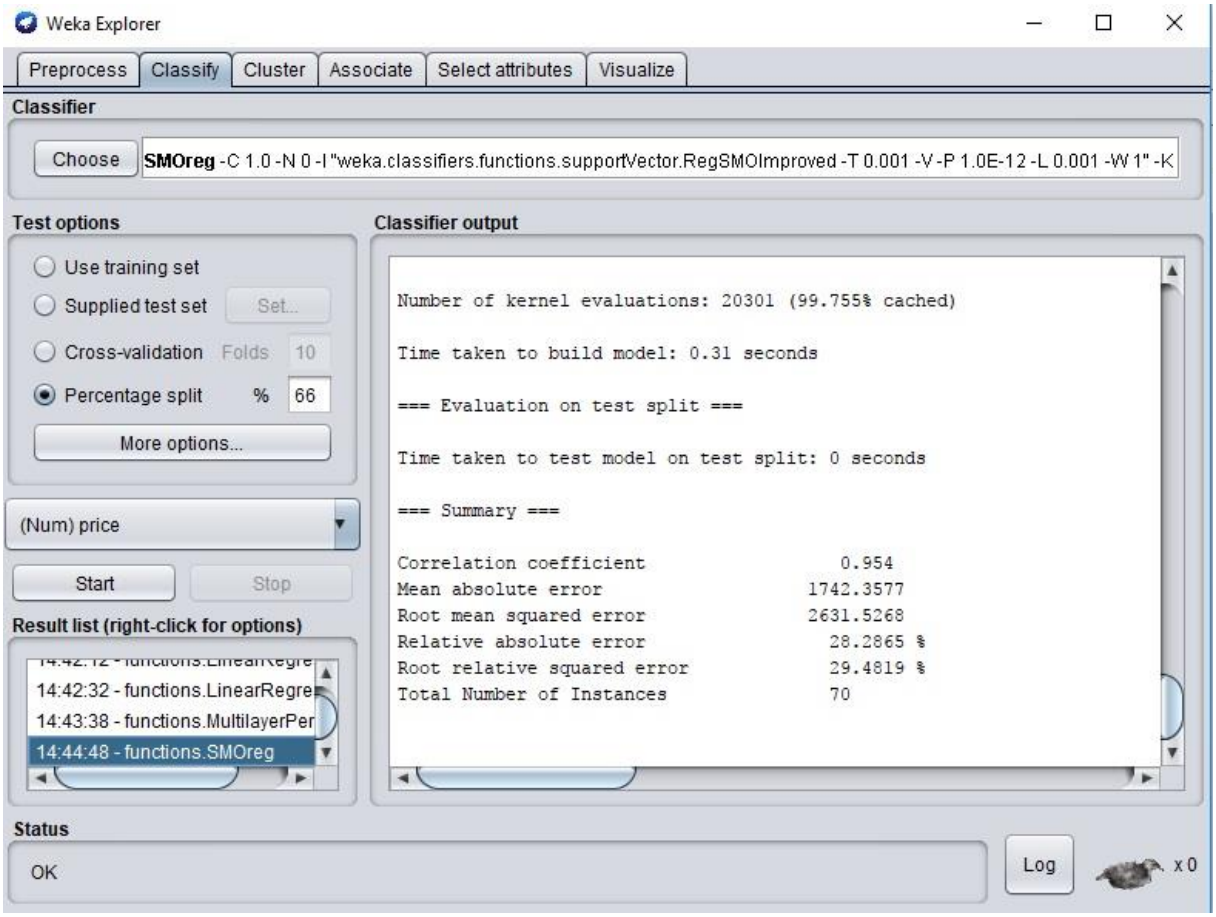
3.Weka ile model oluşturma

- Denenen classification modelleri



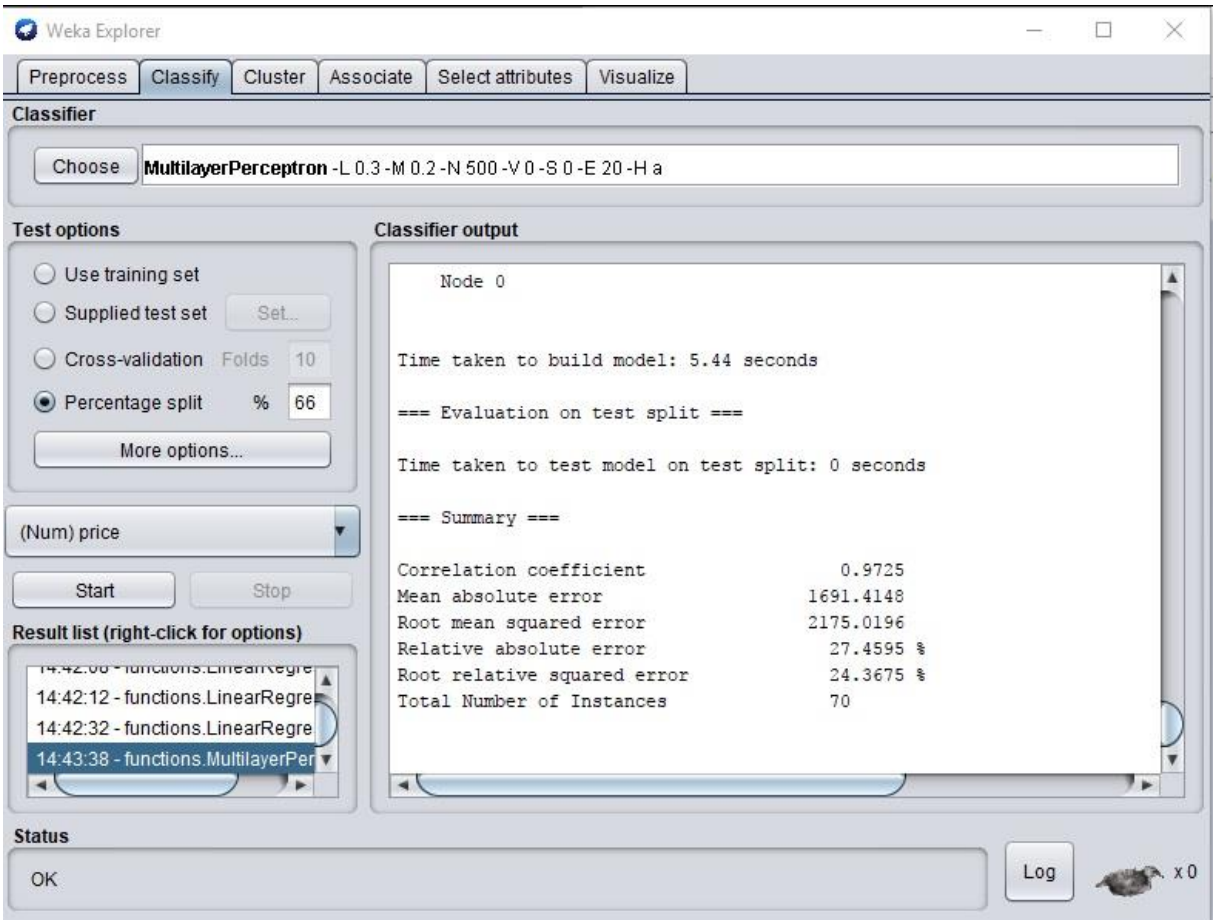


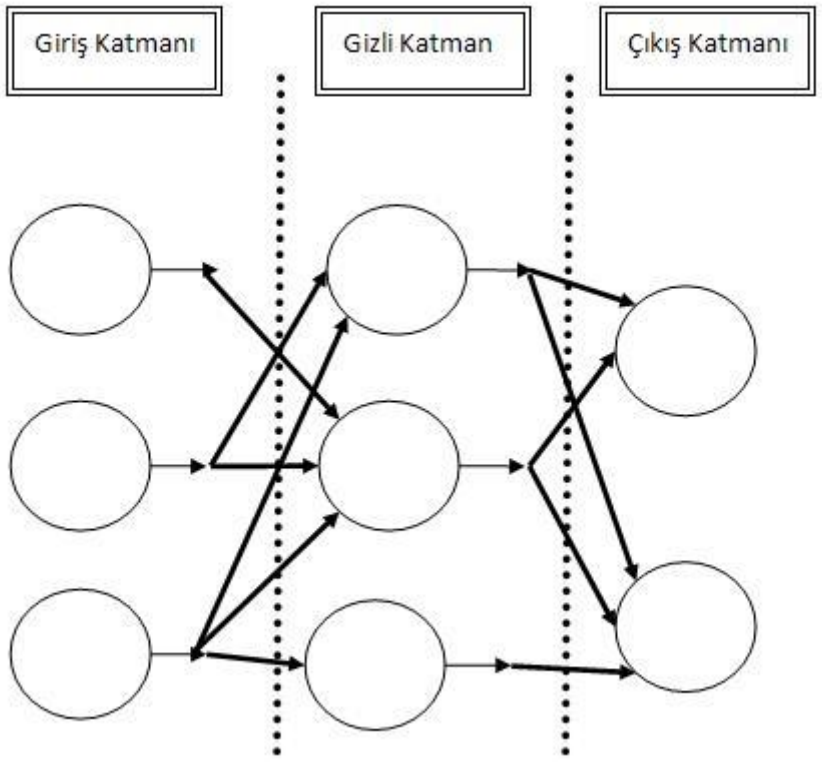
Mean Absolute Error: TahminEdilen-gerçek/n
Relative Absolute Error: tahmindeki sapmalar/gerçekleşen değerlerin ortalamadan sapması



Support Vector Machine, Wekada SMO olarak geçer. Sınıflandırma için bir düzlemde bulunan iki grup arasında bir sınır çizilerek iki gruba ayırmak mümkündür. Bu sınırın çizileceği yer ise iki grubun da üyelerine en uzak yer olmalıdır. SVM bu sınırlarının yapılacağını belirler.

İşlem yapılması için iki gruba da yakın ve birbirine paralel iki sınır çizilir ve bu sınır çizgileri birbirine yaklaştırılarak ortak sınır çizgisi üretilir.

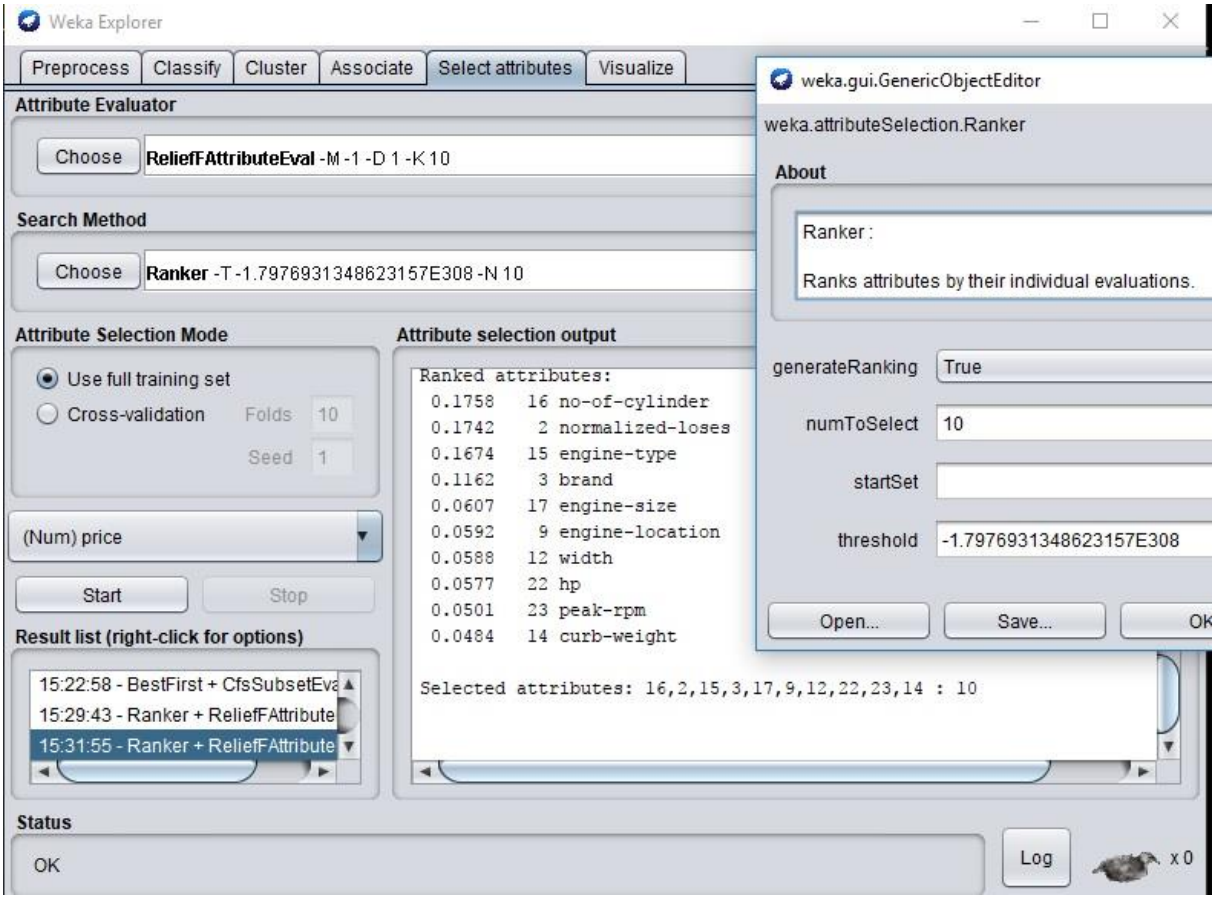




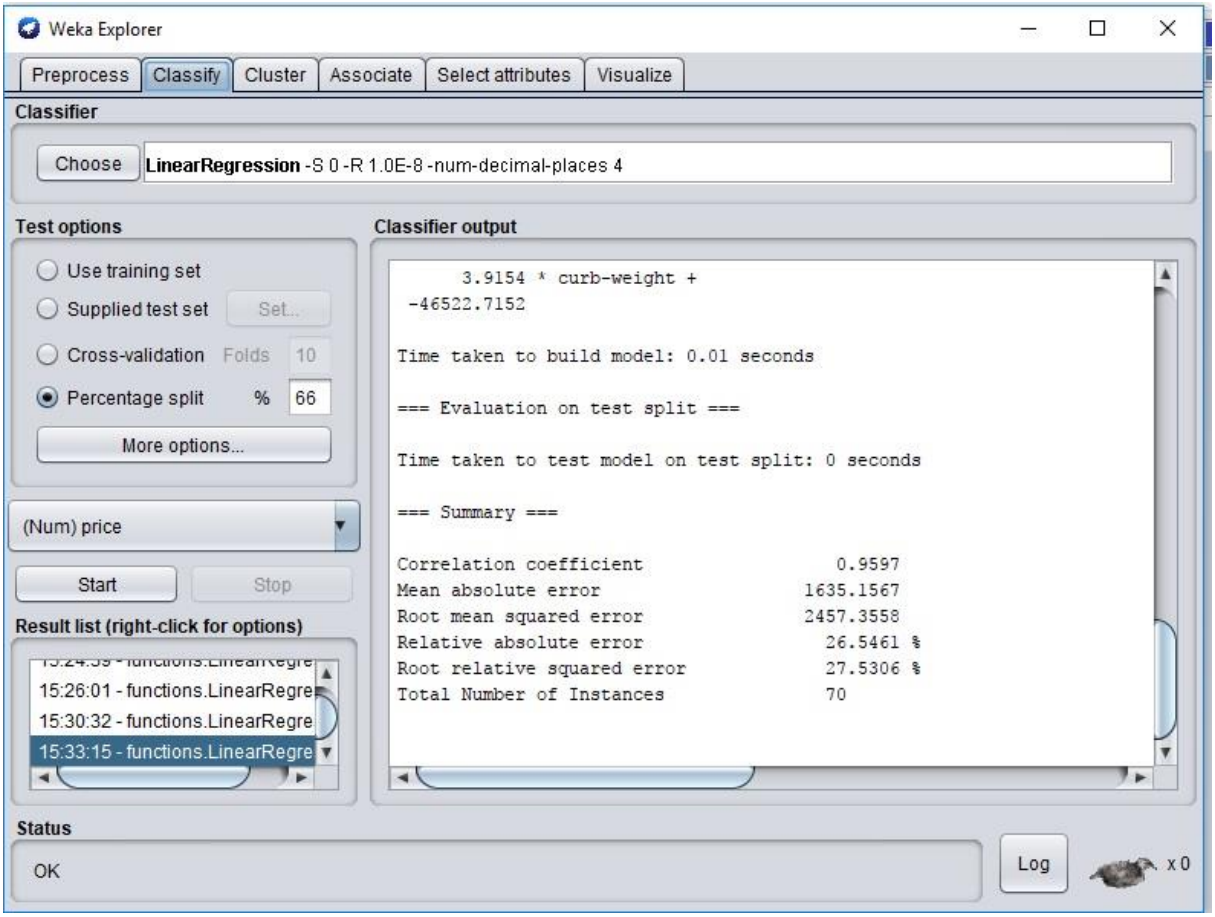
Bu şekilde görüldüğü üzere nöronların bir kısmı giriş, çıkış için kullanılmıştır. Giriş ve çıkışta bulunan bu nöronların ana amacı sistemin dışarısı ile ilgili olan etkileşimini sağlamaktır. Bütün nöronların işlem yapma yeteneği vardır.

- Verisetimiz ile beklediğimiz sonuç regresyon, kesikli değer olduğundan dolayı regresyon bulmanın çeşitli varyanslarını denedik.

- Seçtiğimiz sonuçlar arasında çok büyük bir fark yoktur(0.9488,0.954 ve0.9725). Bu farklar farklı yöntemler içeren farklı regresyon tekniklerinden dolayı olmuştur.
- Verisetimizde fazlalık featurelar vardır. Bunun için ReliefAttributeEval seçip ranker ı 10 yaptım.



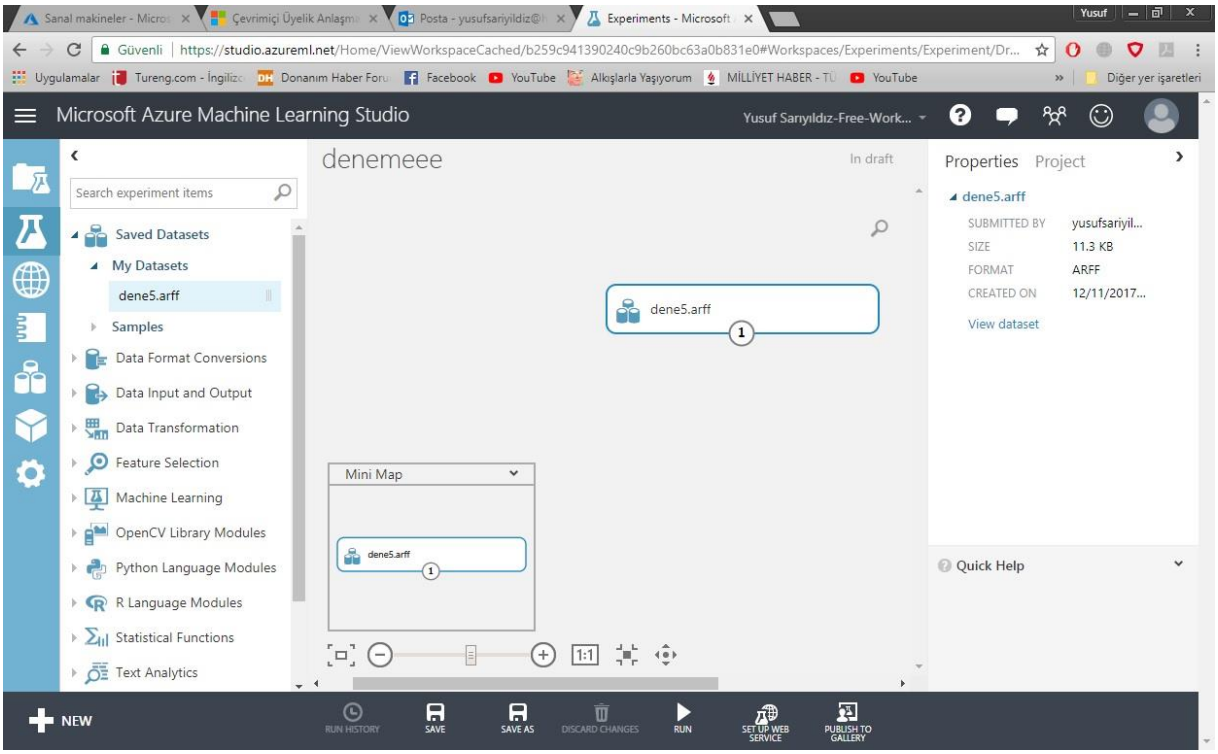
Ardından tekrar lineerRegression yaptım ve değişim.



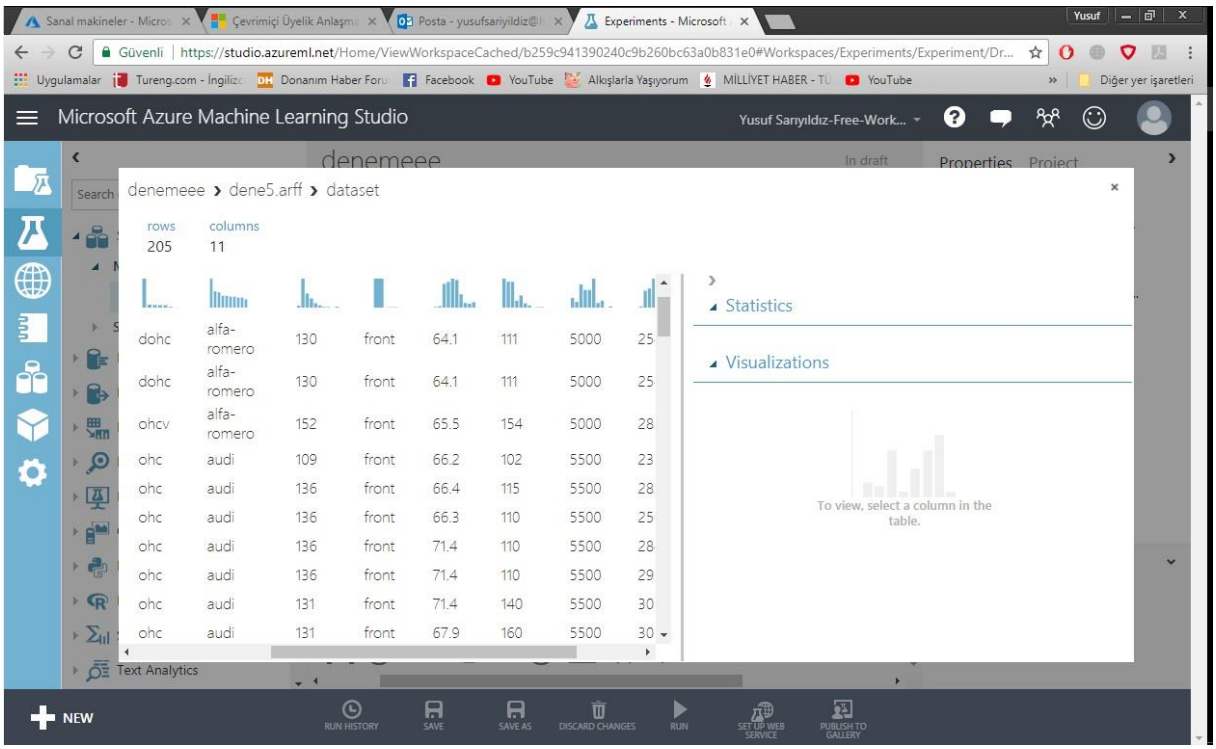
4. Azure Machine Learning üzerinde kullanımı.

azureml.net üzerinde hesap alındıktan sonra. Adım adım proje oluşturacağız.

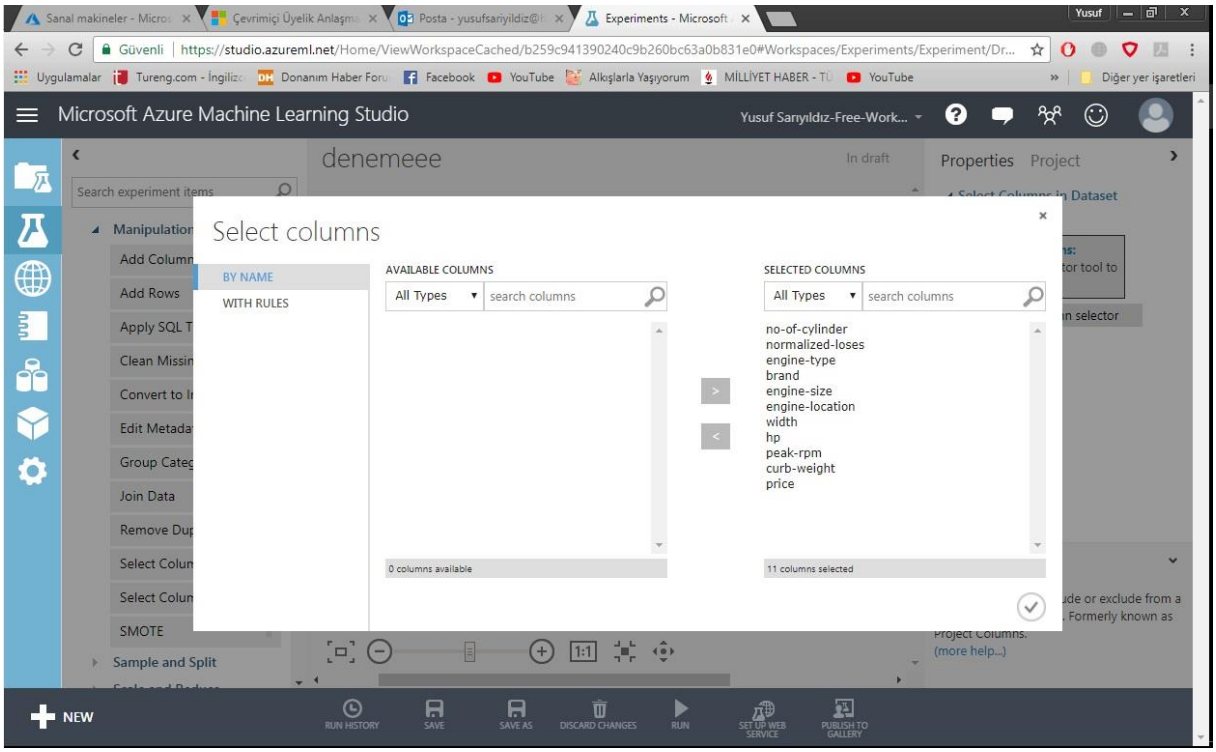
1. Kullanacağımız arff dosyası sisteme eklenir.



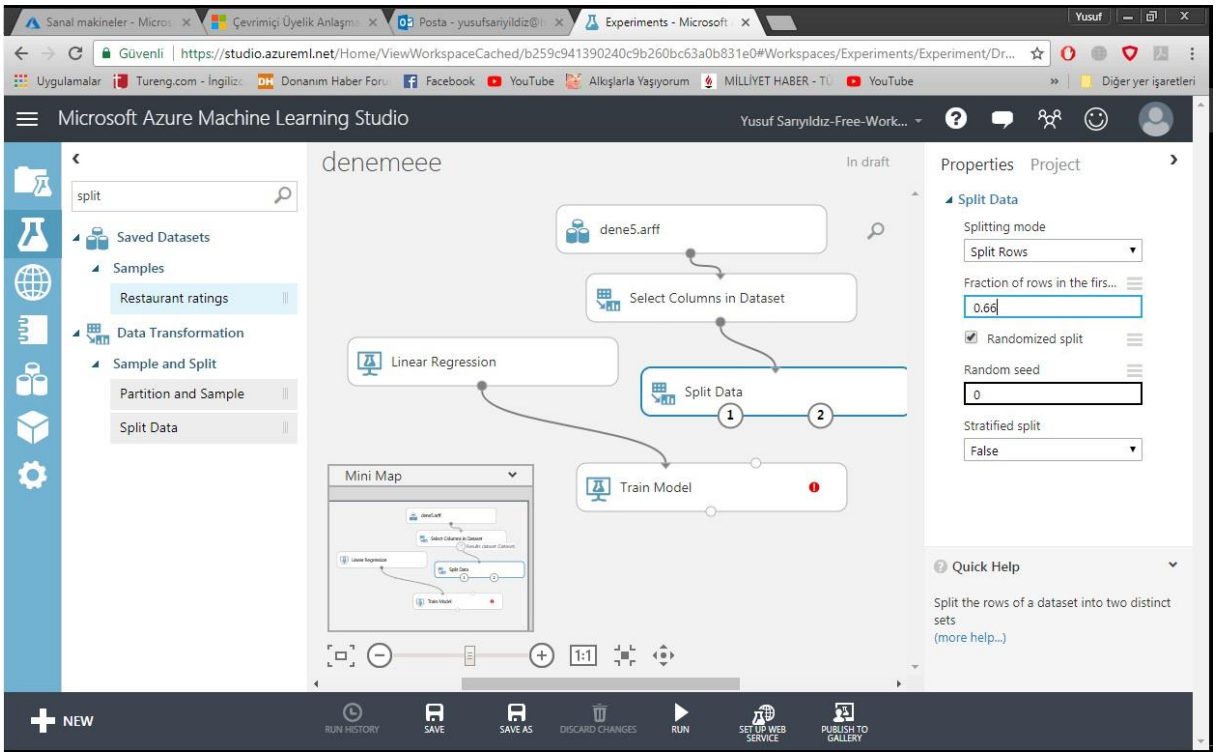
2. Sağ klik yaparak dosyamızın içine bakabiliriz.



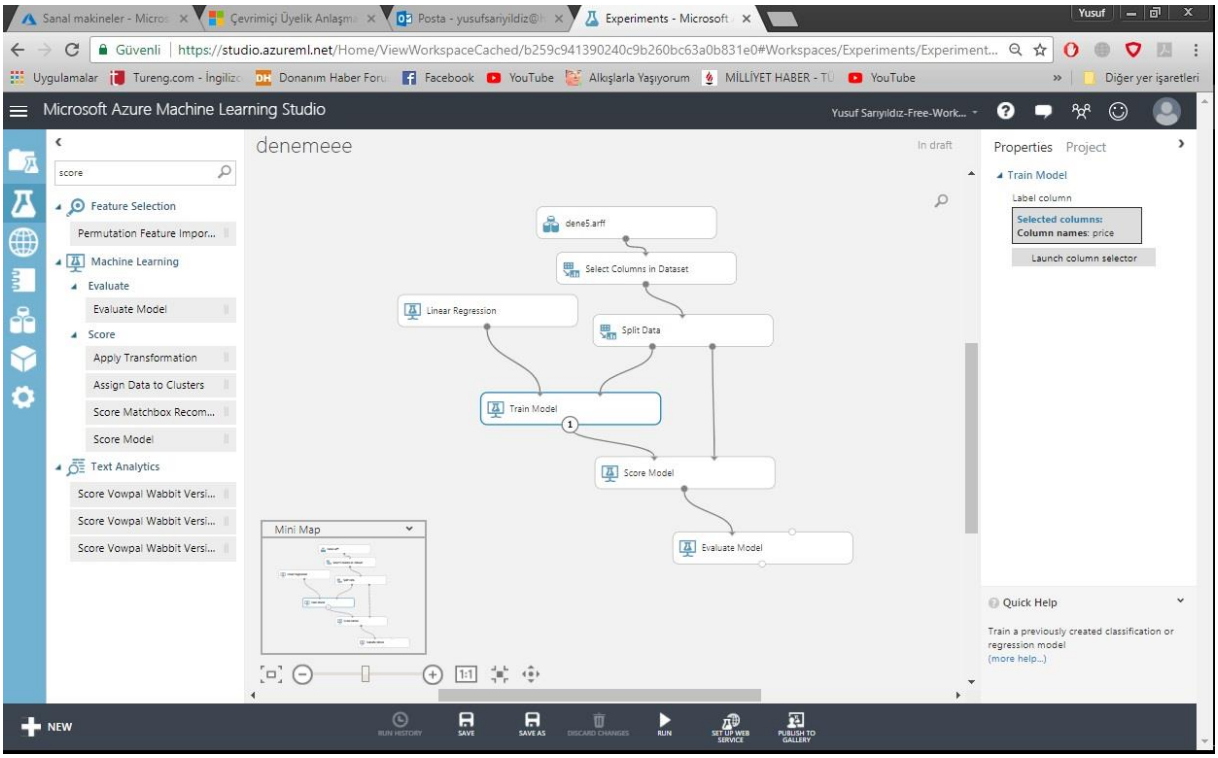
3. Kolonları seçmek için select column in dataseti tıkladık. Sağda çıkan yere launch column selector u seçtik.



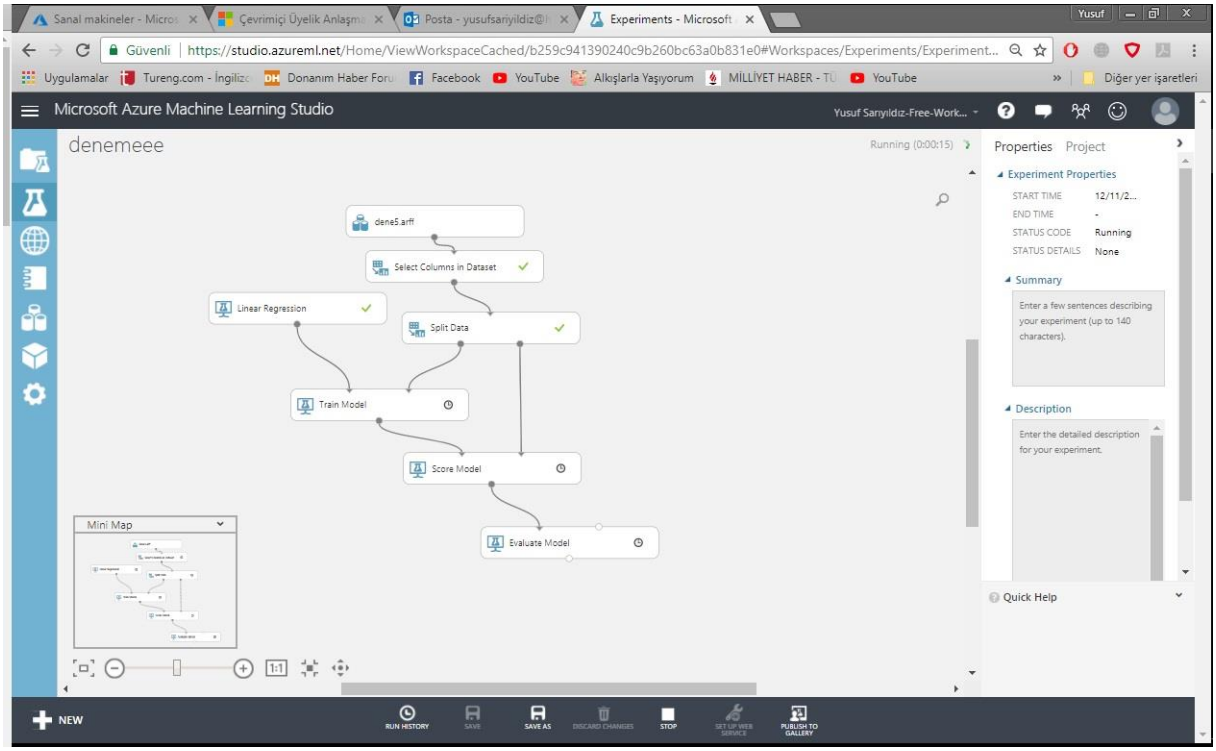
4. Veriyi split ile böldük. %66 olacak şekilde.



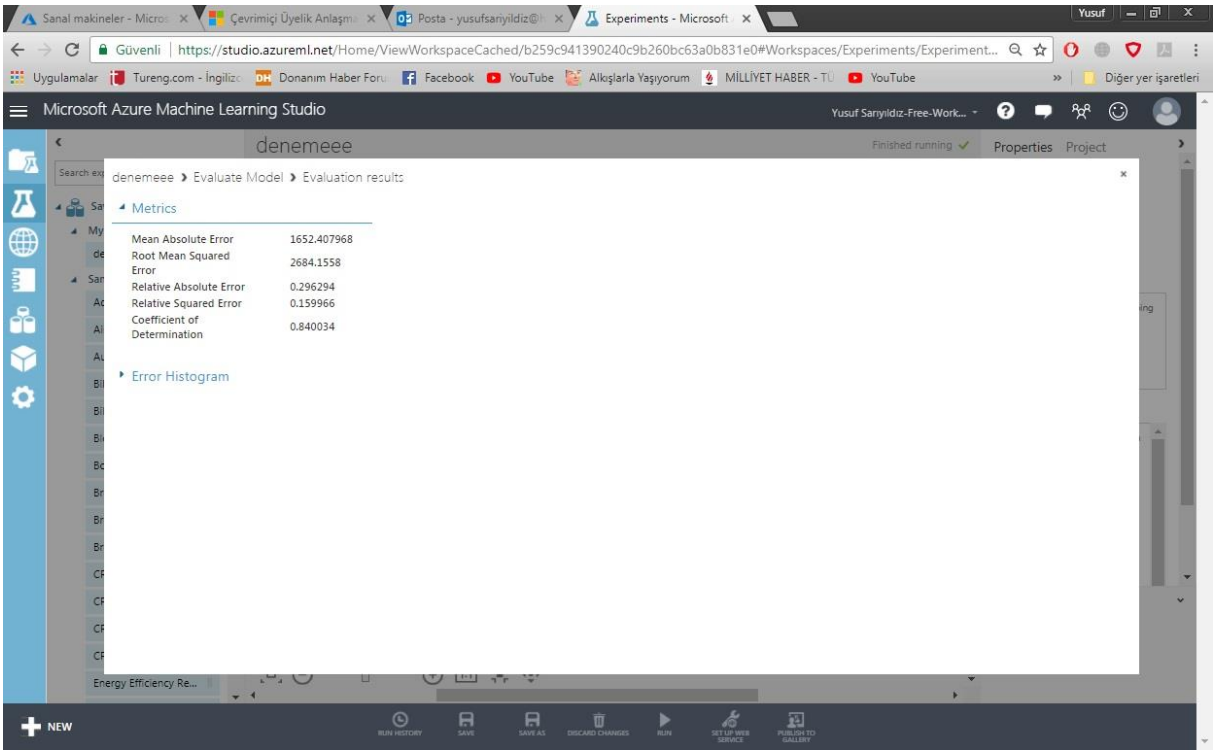
5. Score model ve evaluate modeli birleştirdik. Scorun 2. Kısmını split datadan çektik.



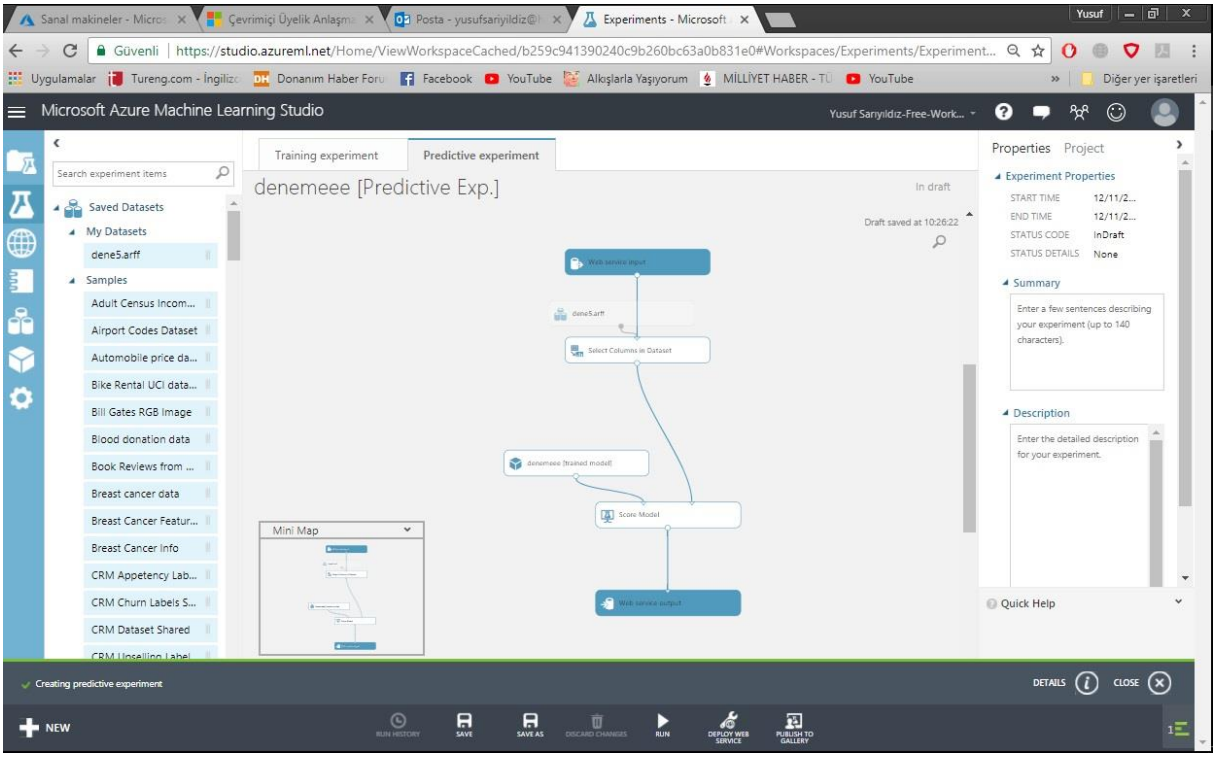
6.Save ve run a tıkladık.



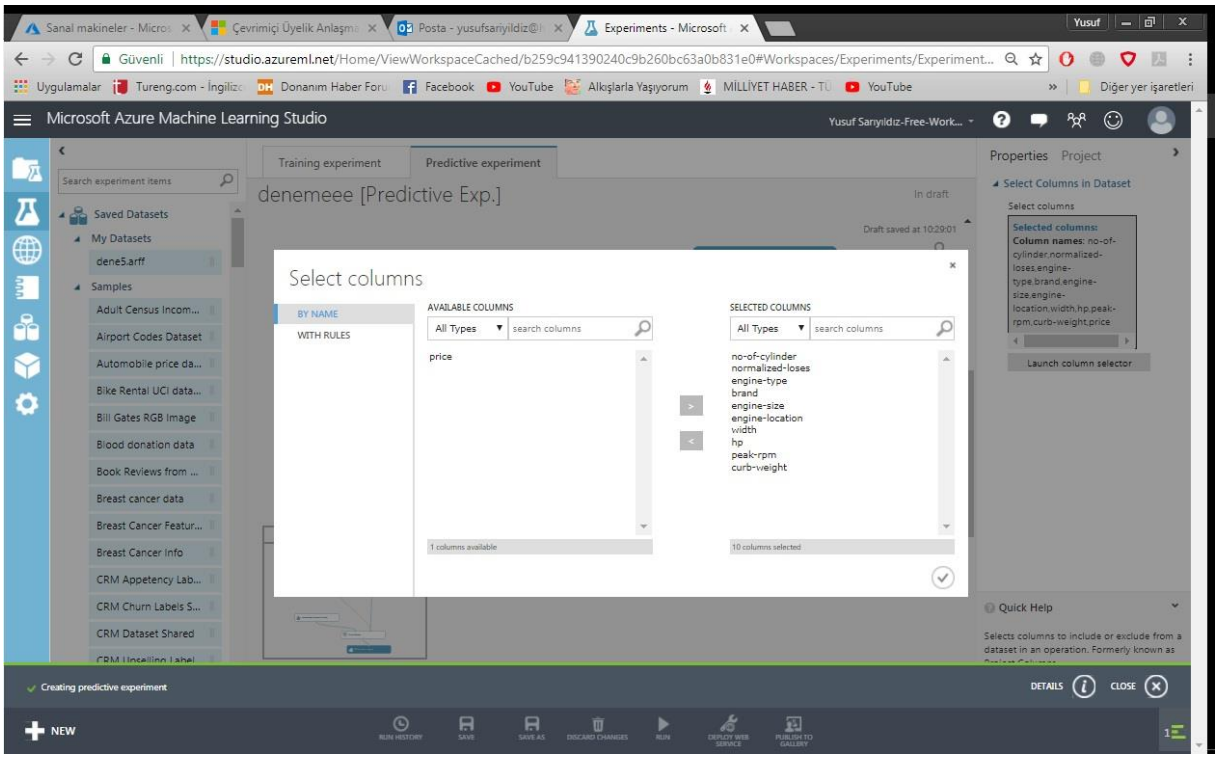
7. Evaluate e sağ tıkladıktan sonra visualize dersek sonuçları verir.



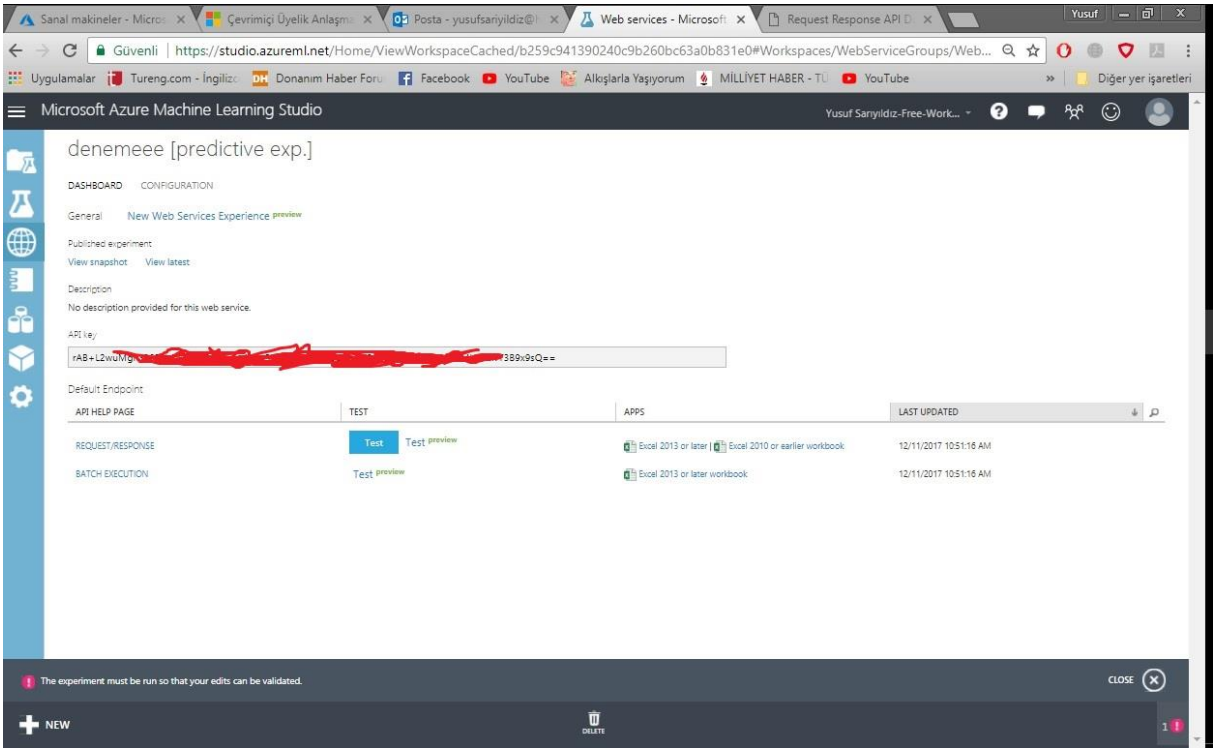
8. Set up web service e tıkladık. Trained model hazırlanmış oldu.



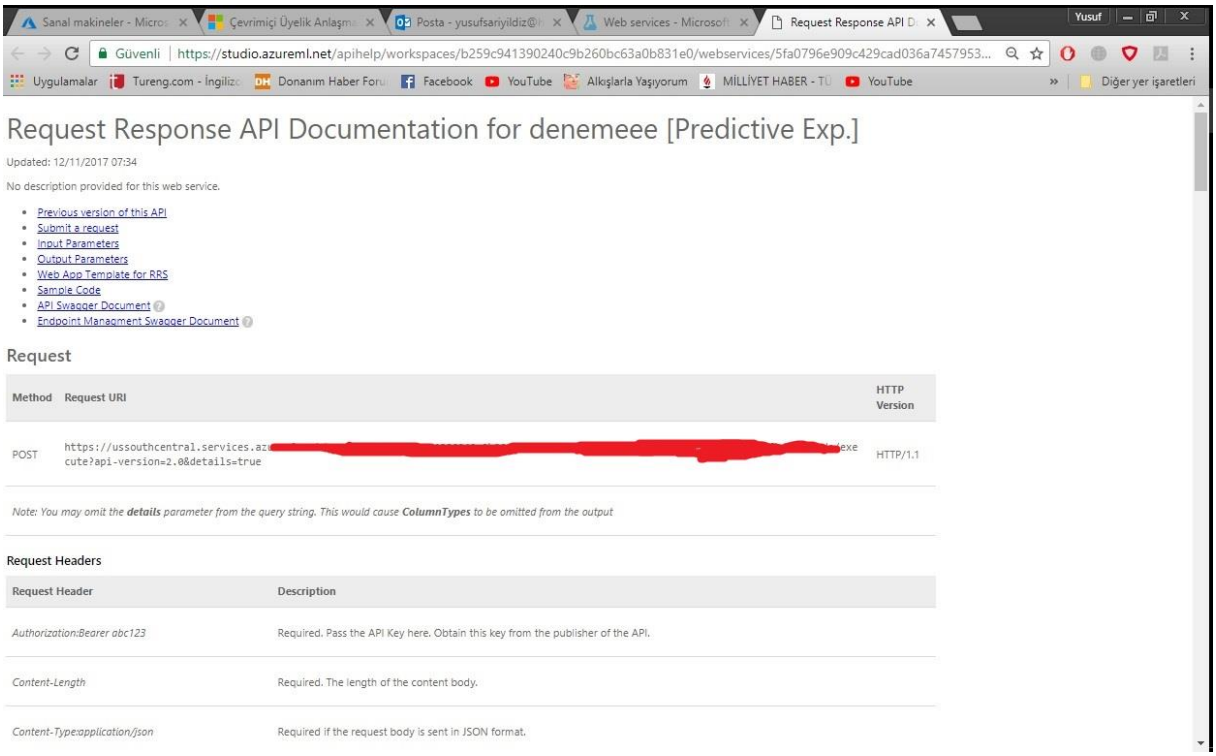
9. Modelimiz price ı verdiğine göre artık kaldırabiliriz. Select column kısmından.



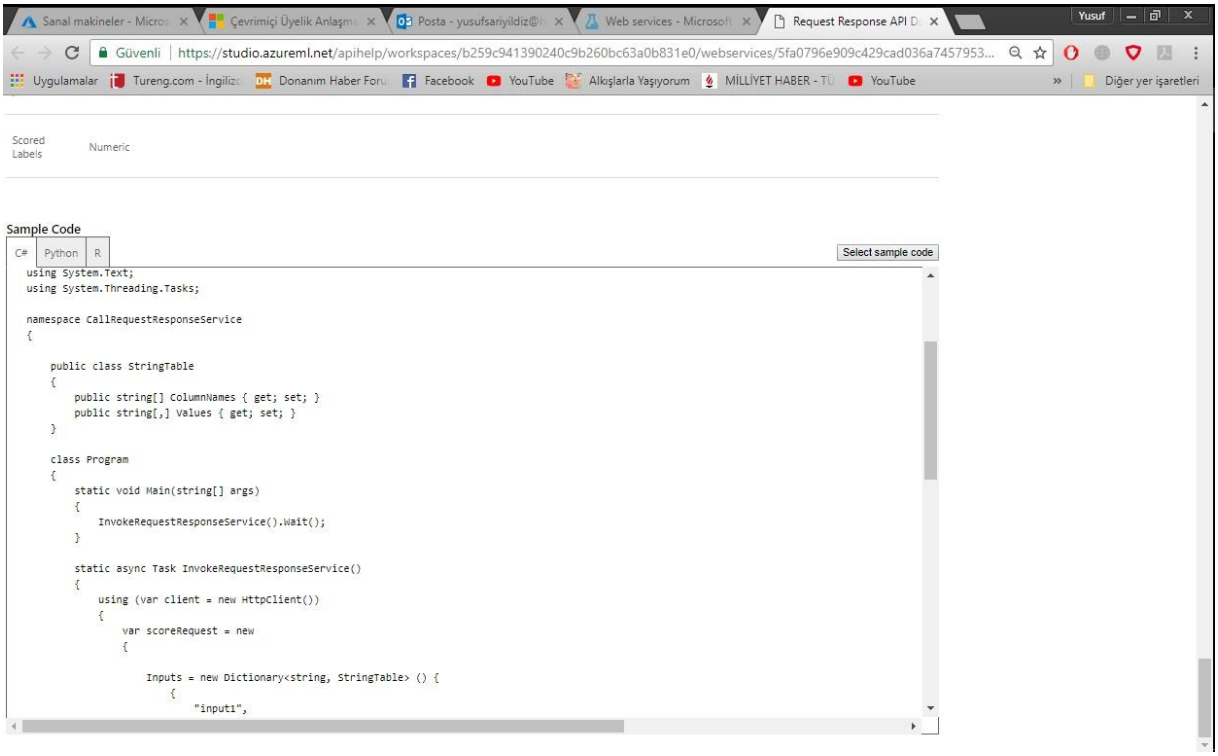
10. Deploy dedik. Karşımıza sayfa çıktı. Burada apikey görünebilir.



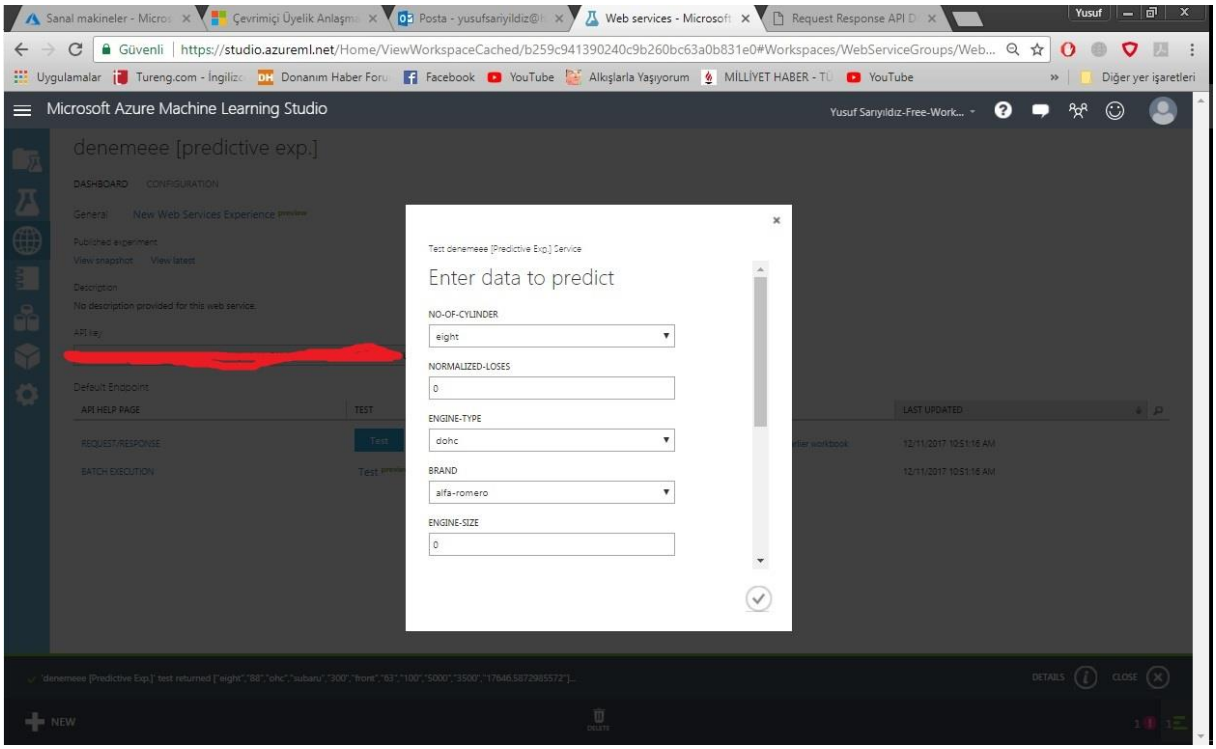
11. Request responda tıklayınca nasıl kullanacağımız hakkında bilgi verir.



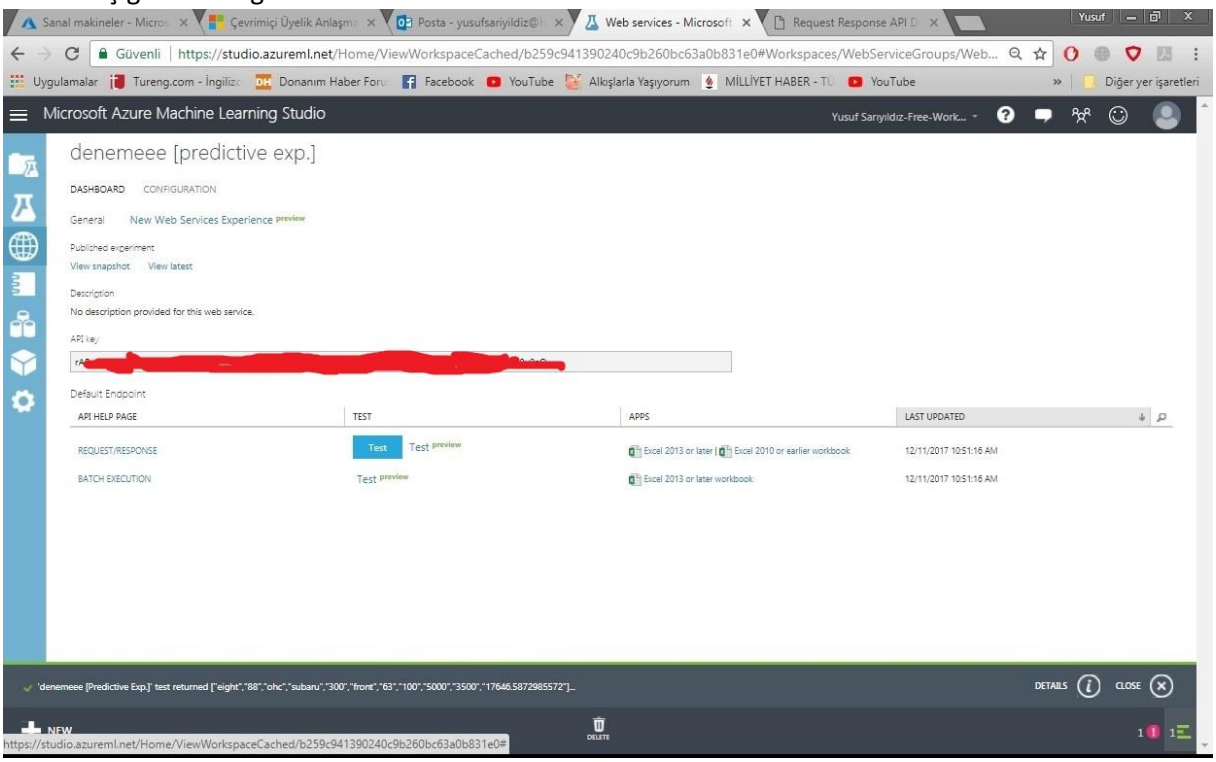
12. Örnek c# kullanımı bu sayfada mevcuttur.



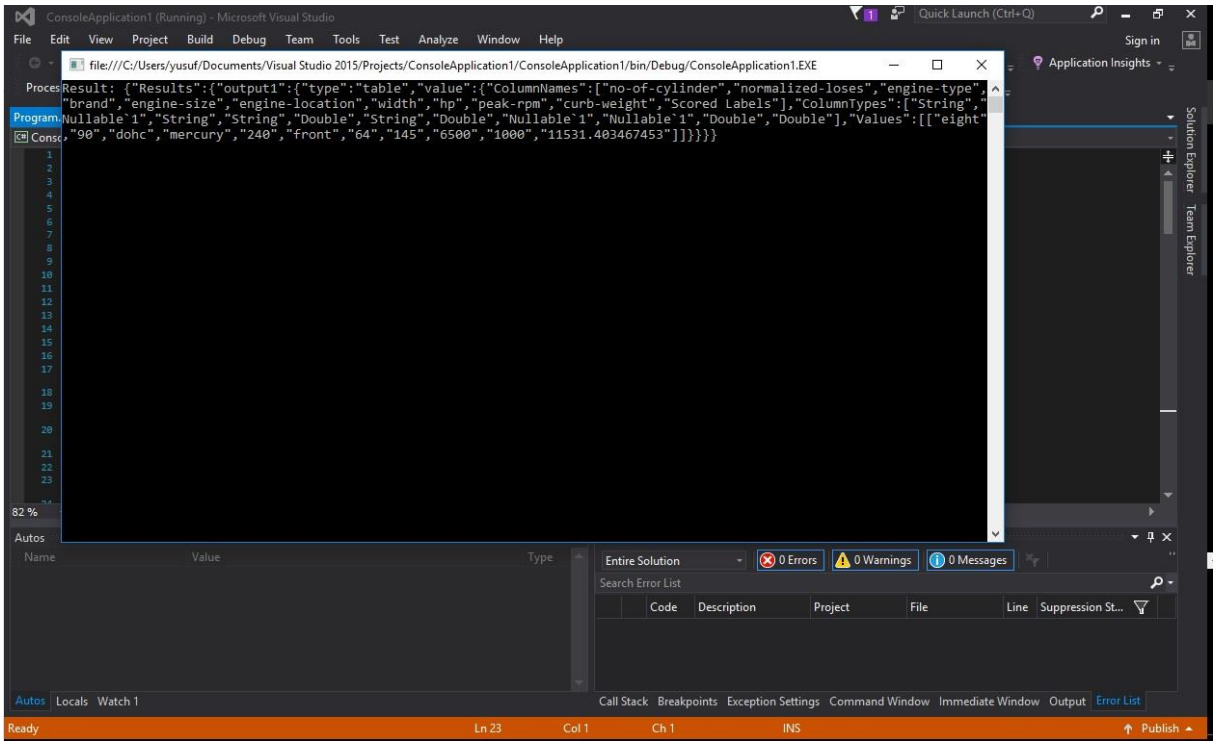
13. Test e tıkladık .



14. Sonuç gri alanda görülebilir.



15. C# üstünde console application sonucu



Kodlar

```
// 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)
        {
            InvokeRequestResponseService().Wait();
        }
```

```
        static async Task InvokeRequestResponseService()
        {
            using (var client = new HttpClient())
            {
                var scoreRequest = new
                {

                    Inputs = new Dictionary<string, StringTable>() {
                        {
                            "input1",
                            new StringTable()
                            {
                                ColumnNames = new string[] {"no-of-cylinder",
"normalized-loses", "engine-type", "brand", "engine-size", "engine-location", "width",
"hp", "peak-rpm", "curb-weight"},
                                // Values = new string[,] {  { "eight", "0", "dohc",
"alfa-romero", "0", "front", "0", "0", "0", "0" },  { "eight", "0", "dohc", "alfa-
romero", "0", "rear", "0", "0", "0", "0" },  }
                               // Values = new string[,] {  { "six", "200", "ohc",
"subaru", "200", "rear", "64", "145", "6500", "1000" } }
                               Values = new string[,] {  { "eight", "90", "dohc",
"mercury", "240", "front", "64", "145", "6500", "1000" } }

                                //4073
                                //11531
                                 /* ----num-of-cylinders:         eight, five, four,
six, three, twelve, two.----
                                ---normalized-losses:        continuous from 65 to
256.------
                                 ----engine-type:              dohc, dohcv, l, ohc,
ohcf, ohcv, rotor.----
                                 ----brand:        alfa-romero, audi, bmw, chevrolet,
dodge, honda,
                                                   isuzu, jaguar, mazda, mercedes-benz,
mercury,
                                                  mitsubishi, nissan, peugot, plymouth,
porsche,
                                                  renault, saab, subaru, toyota,
volkswagen, volvo----
                                   ------ engine-size:              continuous from 61
to 326.---------
                                     -------engine-location:          front, rear.----
--
                                 -------  hp:               continuous from 48 to 288.-
-----
                                      --------width:                    continuous from
60.3 to 72.3.------------
                                  ------horsepower:               continuous from 48 to
288.-------
                                     ----------peak-rpm:                 continuous
from 4150 to 6600.-----------
                                     -------curb-weight:              continuous from
1488 to 4066.---------------- */

                            }
                        },
                    },
                    GlobalParameters = new Dictionary<string, string>()
                    {
                    }
                };
```

```
                const string apiKey =
"rAB+L2aaaaaaaaaaaaaaaaaaaaBDDb7l/sQaaaaaaaaa/cKaaaaaaaaaaaaaaa=="; // 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/workaaaaaaaaspaces/b25aaaaaaaaaaaaaa
aaae0/services/0daaaaaaaaaaaaaaaaac1/execute?api-version=2.0&details=true");

                // WARNING: The 'await' statement below can result in a deadlock if
you are calling this code from the UI thread of an ASP.Net application.
                // One way to address this would be to call ConfigureAwait(false) so
that the execution does not attempt to resume on the original context.
                // For instance, replace code such as:
                //      result = await DoSomeTask()
                // with the following:
                //      result = await DoSomeTask().ConfigureAwait(false)


                HttpResponseMessage response = await client.PostAsJsonAsync("",
scoreRequest);

                if (response.IsSuccessStatusCode)
                {
                    string result = await response.Content.ReadAsStringAsync();
                    Console.WriteLine("Result: {0}", result);
                }
                else
                {
                    Console.WriteLine(string.Format("The request failed with status
code: {0}", response.StatusCode));

                    // Print the headers - they include the requert ID and the
timestamp, which are useful for debugging the failure
                    Console.WriteLine(response.Headers.ToString());

                    string responseContent = await
response.Content.ReadAsStringAsync();
                    Console.WriteLine(responseContent);
                }
                System.Threading.Thread.Sleep(30000000);//görmek için beklettik
            }
        }
    }
}
```