Manisa Celal Bayar Üniversitesi

Hasan Ferdi Turgutlu Teknoloji Fakültesi

Yazılım Mühendisliği

**MAKİNE ÖĞRENMESİ DERSİ PROJESİ**

**2017-2018**

Proje Ekibi

152802052 Taner YILDIRIM

142802028 Orkun ÇELİKAĞ

## Projenin Amacı

Projemizde öğrenemsi algoritmalarını kullanarak restoran önerisi yapan bir uygulama geliştirmek.

## Girdiler

Kullanıcının restoranttan istediği özellikleri girerek öneride bulunacak.

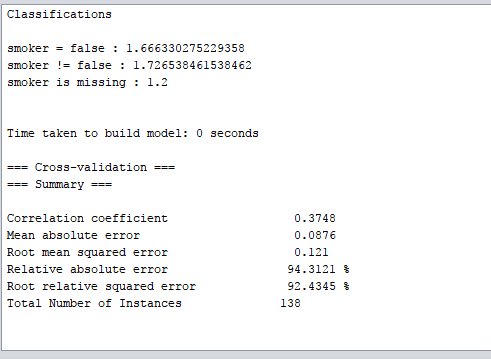
## Beklenen Çıktılar

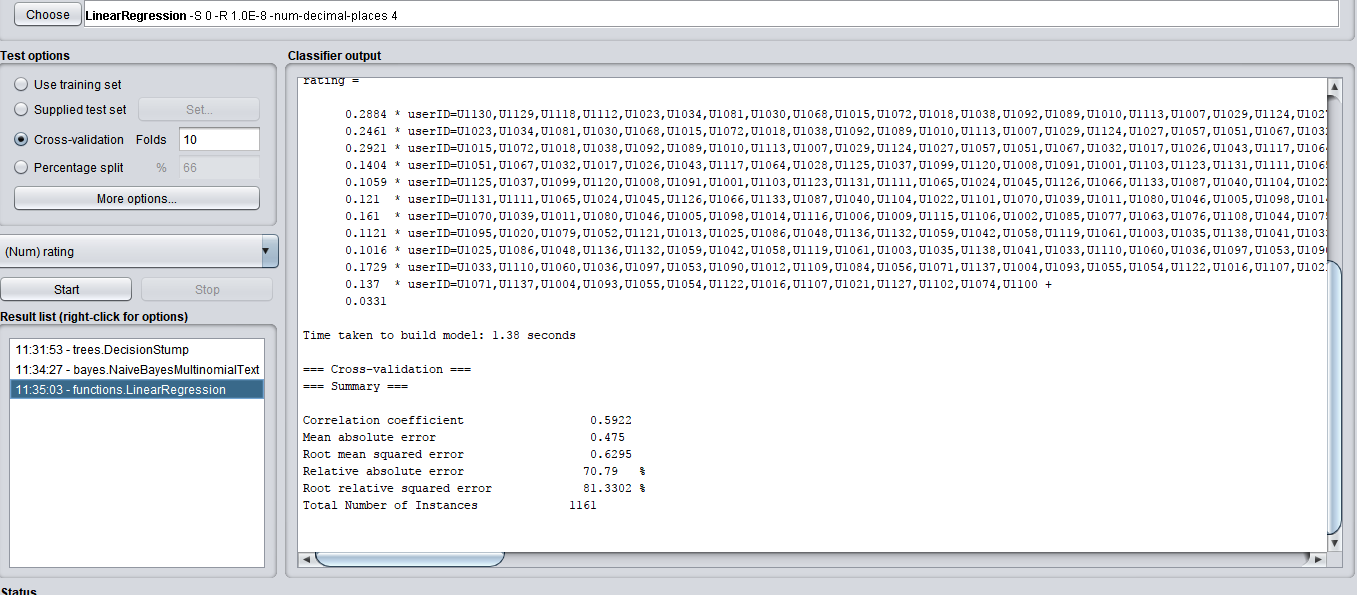
Kullanıcıya attirbutelardan ve azure machine learning kodu ile restorant önerilerini listelemesi.

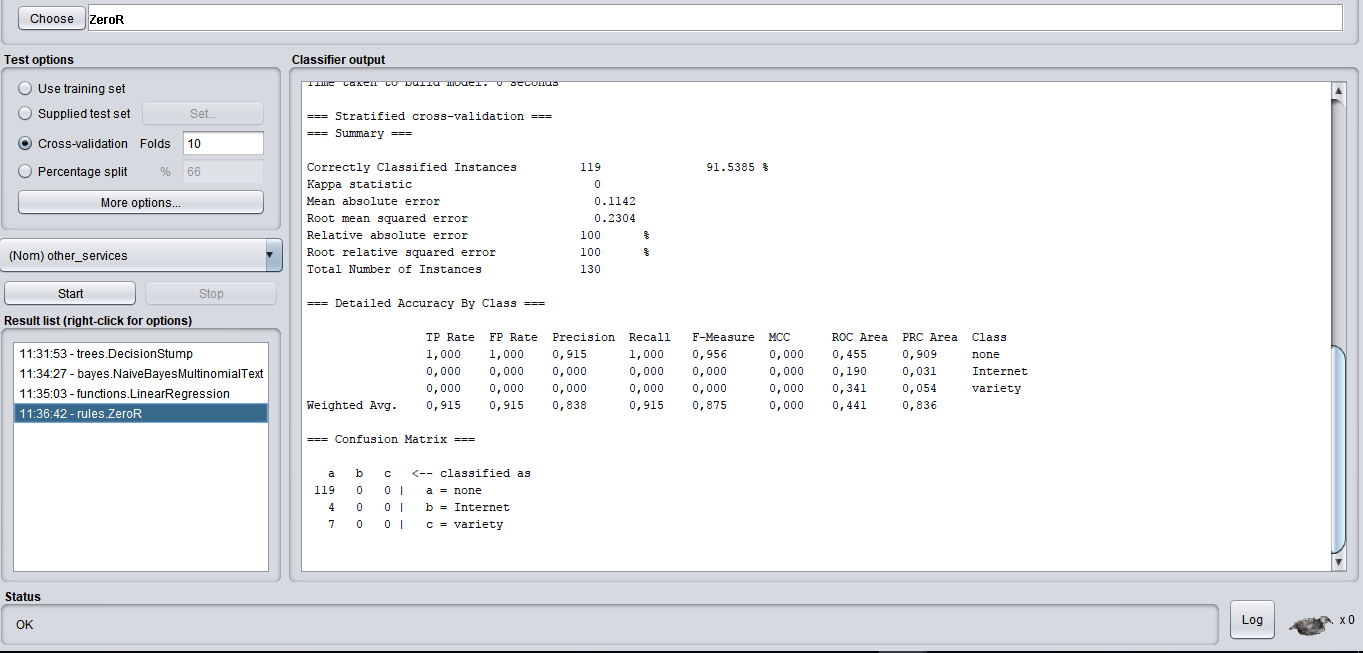
## Mantığı

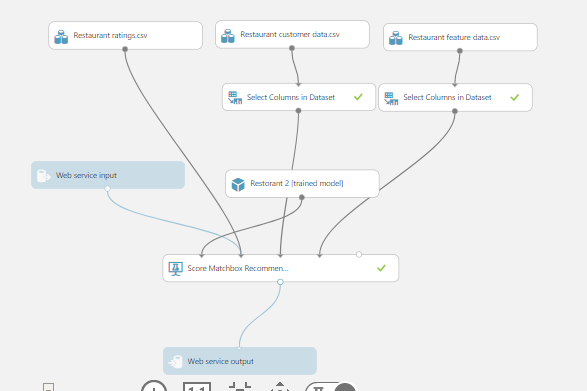
Gün geçtikçe daha çok kişinin tercih ettiği Azure Machine Learning çözümü olan Studio’ya hoş geldiniz. Tarayıcı tabanlı bir araç olan Studio, sürükleyip bırakma mantığıyla çalışan görsel bir yazma ortamıdır ve kodlama gerektirmeden kolayca kullanılabilir. Yalnızca birkaç tıklamayla fikir aşamasından dağıtım aşamasına geçin.Azure machine learning ile verilen kod bloğu ve azure kısmındaki veri analizlerine yönlendirerek kullanıcıya her defasında önermede bulunan bir uygulama.

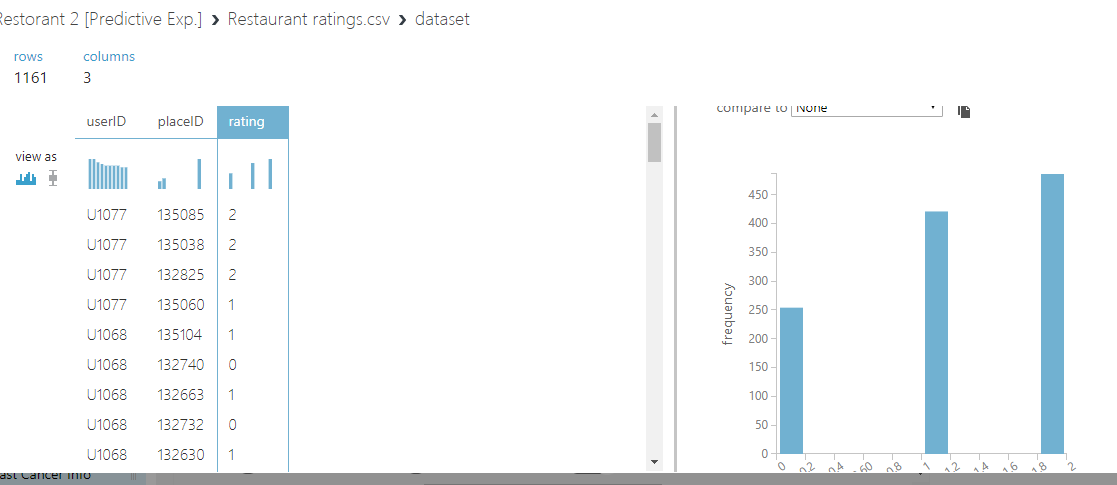
## Ekran görüntüsü

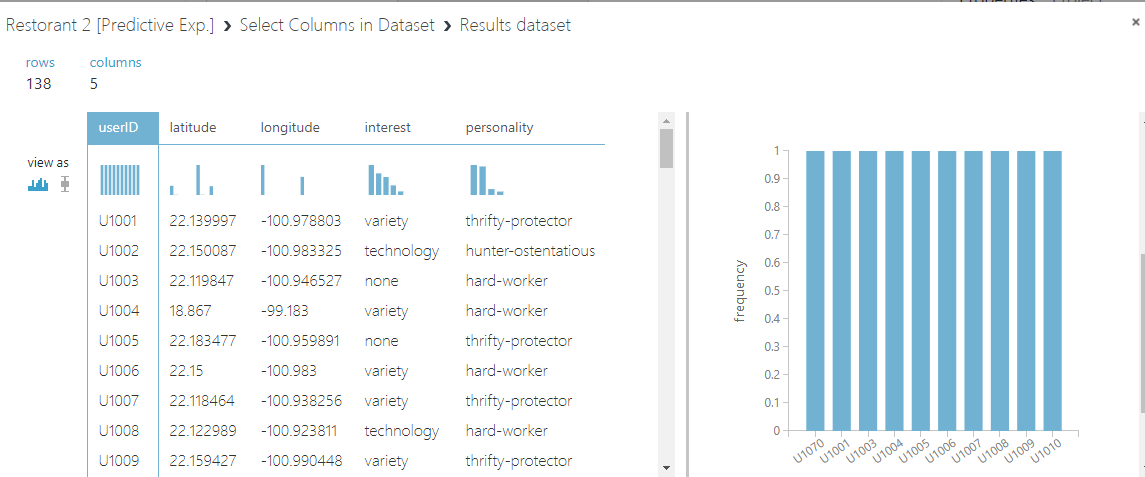


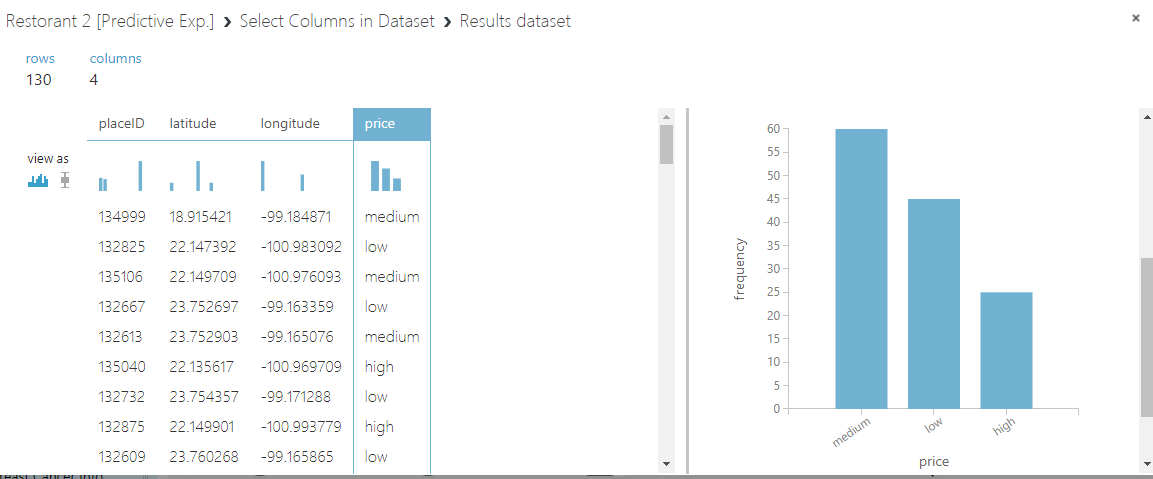


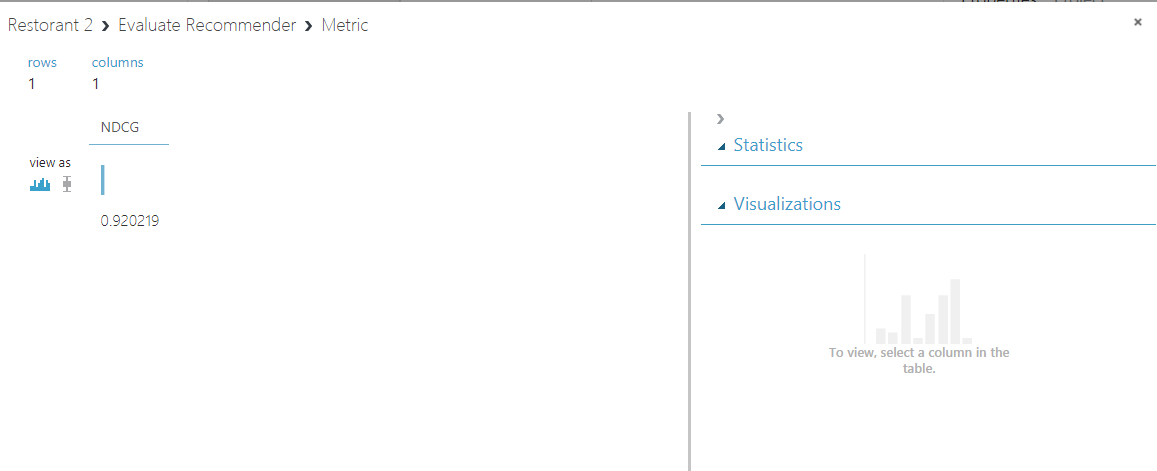












## Kod yapısı

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Globalization;

using System.IO;

using System.Linq;

using System.Net.Http;

using System.Net.Http.Formatting;

using System.Net.Http.Headers;

using System.Runtime.Serialization;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.WindowsAzure.Storage;

using Microsoft.WindowsAzure.Storage.Auth;

using Microsoft.WindowsAzure.Storage.Blob;

namespace CallBatchExecutionService

{

public class AzureBlobDataReference

{

// Storage connection string used for regular blobs. It has the following format:

// DefaultEndpointsProtocol=https;AccountName=ACCOUNT\_NAME;AccountKey=ACCOUNT\_KEY

// It's not used for shared access signature blobs.

public string ConnectionString { get; set; }

// Relative uri for the blob, used for regular blobs as well as shared access

// signature blobs.

public string RelativeLocation { get; set; }

// Base url, only used for shared access signature blobs.

public string BaseLocation { get; set; }

// Shared access signature, only used for shared access signature blobs.

public string SasBlobToken { get; set; }

}

public enum BatchScoreStatusCode

{

NotStarted,

Running,

Failed,

Cancelled,

Finished

}

public class BatchScoreStatus

{

// Status code for the batch scoring job

public BatchScoreStatusCode StatusCode { get; set; }

// Locations for the potential multiple batch scoring outputs

public IDictionary<string, AzureBlobDataReference> Results { get; set; }

// Error details, if any

public string Details { get; set; }

}

public class BatchExecutionRequest

{

public IDictionary<string, AzureBlobDataReference> Inputs { get; set; }

public IDictionary<string, string> GlobalParameters { get; set; }

// Locations for the potential multiple batch scoring outputs

public IDictionary<string, AzureBlobDataReference> Outputs { get; set; }

}

class Program

{

static void Main(string[] args)

{

InvokeBatchExecutionService().Wait();

}

static async Task WriteFailedResponse(HttpResponseMessage response)

{

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);

}

static void SaveBlobToFile(AzureBlobDataReference blobLocation, string resultsLabel)

{

const string OutputFileLocation = "myresults.csv"; // Replace this with the location you would like to use for your output file

var credentials = new StorageCredentials(blobLocation.SasBlobToken);

var blobUrl = new Uri(new Uri(blobLocation.BaseLocation), blobLocation.RelativeLocation);

var cloudBlob = new CloudBlockBlob(blobUrl, credentials);

Console.WriteLine(string.Format("Reading the result from {0}", blobUrl.ToString()));

cloudBlob.DownloadToFile(OutputFileLocation, FileMode.Create);

Console.WriteLine(string.Format("{0} have been written to the file {1}", resultsLabel, OutputFileLocation));

}

static void UploadFileToBlob(string inputFileLocation, string inputBlobName, string storageContainerName, string storageConnectionString)

{

// Make sure the file exists

if (!File.Exists(inputFileLocation))

{

throw new FileNotFoundException(

string.Format(

CultureInfo.InvariantCulture,

"File {0} doesn't exist on local computer.",

inputFileLocation));

}

Console.WriteLine("Uploading the input to blob storage...");

var blobClient = CloudStorageAccount.Parse(storageConnectionString).CreateCloudBlobClient();

var container = blobClient.GetContainerReference(storageContainerName);

container.CreateIfNotExists();

var blob = container.GetBlockBlobReference(inputBlobName);

blob.UploadFromFile(inputFileLocation, FileMode.Open);

}

static void ProcessResults(BatchScoreStatus status)

{

bool first = true;

foreach (var output in status.Results)

{

var blobLocation = output.Value;

Console.WriteLine(string.Format("The result '{0}' is available at the following Azure Storage location:", output.Key));

Console.WriteLine(string.Format("BaseLocation: {0}", blobLocation.BaseLocation));

Console.WriteLine(string.Format("RelativeLocation: {0}", blobLocation.RelativeLocation));

Console.WriteLine(string.Format("SasBlobToken: {0}", blobLocation.SasBlobToken));

Console.WriteLine();

// Save the first output to disk

if (first)

{

first = false;

SaveBlobToFile(blobLocation, string.Format("The results for {0}", output.Key));

}

}

}

static async Task InvokeBatchExecutionService()

{

// How this works:

//

// 1. Assume the input is present in a local file (if the web service accepts input)

// 2. Upload the file to an Azure blob - you'd need an Azure storage account

// 3. Call the Batch Execution Service to process the data in the blob. Any output is written to Azure blobs.

// 4. Download the output blob, if any, to local file

const string BaseUrl = "https://ussouthcentral.services.azureml.net/workspaces/5c36cfec60c14241afab0788a9c7f4aa/services/fff5e2c7ea414e1b9ac562e705dee417/jobs";

const string StorageAccountName = "mystorageacct"; // Replace this with your Azure Storage Account name

const string StorageAccountKey = "Dx9WbMIThAvXRQWap/aLnxT9LV5txxw=="; // Replace this with your Azure Storage Key

const string StorageContainerName = "mycontainer"; // Replace this with your Azure Storage Container name

string storageConnectionString = string.Format("DefaultEndpointsProtocol=https;AccountName={0};AccountKey={1}", StorageAccountName, StorageAccountKey);

const string apiKey = "abc123"; // Replace this with the API key for the web service

// set a time out for polling status

const int TimeOutInMilliseconds = 120 \* 1000; // Set a timeout of 2 minutes

UploadFileToBlob("input1data.csv" /\*Replace this with the location of your input file\*/,

"input1datablob.csv" /\*Replace this with the name you would like to use for your Azure blob; this needs to have the same extension as the input file \*/,

StorageContainerName, storageConnectionString);

using (HttpClient client = new HttpClient())

{

var request = new BatchExecutionRequest()

{

Inputs = new Dictionary<string, AzureBlobDataReference>()

{

{

"input1",

new AzureBlobDataReference()

{

ConnectionString = storageConnectionString,

RelativeLocation = string.Format("{0}/input1datablob.csv", StorageContainerName)

}

},

},

Outputs = new Dictionary<string, AzureBlobDataReference>()

{

{

"output1",

new AzureBlobDataReference()

{

ConnectionString = storageConnectionString,

RelativeLocation = string.Format("/{0}/output1results.csv", StorageContainerName)

}

},

},

GlobalParameters = new Dictionary<string, string>() {

}

};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", apiKey);

// 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)

Console.WriteLine("Submitting the job...");

// submit the job

var response = await client.PostAsJsonAsync(BaseUrl + "?api-version=2.0", request);

if (!response.IsSuccessStatusCode)

{

await WriteFailedResponse(response);

return;

}

string jobId = await response.Content.ReadAsAsync<string>();

Console.WriteLine(string.Format("Job ID: {0}", jobId));

// start the job

Console.WriteLine("Starting the job...");

response = await client.PostAsync(BaseUrl + "/" + jobId + "/start?api-version=2.0", null);

if (!response.IsSuccessStatusCode)

{

await WriteFailedResponse(response);

return;

}

string jobLocation = BaseUrl + "/" + jobId + "?api-version=2.0";

Stopwatch watch = Stopwatch.StartNew();

bool done = false;

while (!done)

{

Console.WriteLine("Checking the job status...");

response = await client.GetAsync(jobLocation);

if (!response.IsSuccessStatusCode)

{

await WriteFailedResponse(response);

return;

}

BatchScoreStatus status = await response.Content.ReadAsAsync<BatchScoreStatus>();

if (watch.ElapsedMilliseconds > TimeOutInMilliseconds)

{

done = true;

Console.WriteLine(string.Format("Timed out. Deleting job {0} ...", jobId));

await client.DeleteAsync(jobLocation);

}

switch (status.StatusCode) {

case BatchScoreStatusCode.NotStarted:

Console.WriteLine(string.Format("Job {0} not yet started...", jobId));

break;

case BatchScoreStatusCode.Running:

Console.WriteLine(string.Format("Job {0} running...", jobId));

break;

case BatchScoreStatusCode.Failed:

Console.WriteLine(string.Format("Job {0} failed!", jobId));

Console.WriteLine(string.Format("Error details: {0}", status.Details));

done = true;

break;

case BatchScoreStatusCode.Cancelled:

Console.WriteLine(string.Format("Job {0} cancelled!", jobId));

done = true;

break;

case BatchScoreStatusCode.Finished:

done = true;

Console.WriteLine(string.Format("Job {0} finished!", jobId));

ProcessResults(status);

break;

}

if (!done) {

Thread.Sleep(1000); // Wait one second

}

}

}

}

}

}

if(kontrol)// eğer kullanıcı aynı sayıyı girmişse bu ifin içine girer ve kullanıcı uyarılır. tekrardan aynı sayıyı girmesi içinde j bir azaltılır.

{

System.out.println("Bir kolonda aynı sayıdan iki tane olamaz.");

j--;

}

else// eğer kullanıcı var olan bir sayıyı girmediyse sayı gerekli yere atanır.

{

kolonlar[i][j] = sayi;

}

}

}

return kolonlar;

}

/\*Kullanıcının girdiği veya rastgele üretilen sayının tekrardan kolona yazılmaması için bu metod kullanılmıştır.\*/

public static boolean KontrolKolon(int kolon[], int kontrolSayi)

{

boolean kontrol = false;

for(int i = 0; i< kolon.length;i++)

{

if(kolon[i] == kontrolSayi)

kontrol = true;

}

return kontrol;

}

/\*çekilişin yapıldığı metod\*/

public static int[] CekilisYap(int secim)

{

Random randomNum = new Random();

int kolon[],sayi;

boolean kontrol;

if(secim == 1)//On numara çekilişinde 22 sayı çekiliyor. ve ona göre dizi oluşturuluyor

{

kolon = new int[22];

}

else

{

kolon = new int[6];

}

/\*Kontrolde sıkıtı yaşamamk için hepsi sıfıra atanıyor\*/

for(int i = 0; i<kolon.length; i++)

{

kolon[i] = 0;

}

int i = 0;

while(i < kolon.length) //Kullanıcın oynadığı oyuna göre çekiliş yapılıyor

{

if(secim == 1 )

{

sayi = randomNum.nextInt(80)+1;

}

else if(secim == 2 && i == 5)

{

sayi = randomNum.nextInt(14)+1;

}

else if(secim == 2)

{

sayi = randomNum.nextInt(34)+1;

}

else if(secim == 3)

{

sayi = randomNum.nextInt(54)+1;

}

else

{

sayi = randomNum.nextInt(49)+1;

}

if(!(secim == 2 && i == 5))/\*Şans topu oyunu için ayrıca bir else if eklenmiştir +1 sayıyı üretebilemk için.\*/

{

kontrol = KontrolKolon(kolon,sayi);

}

else

{

kontrol = false;

}

if(!kontrol)

{

kolon[i] = sayi;

i++;

}

}

return kolon;

}

/\*Çekiliş sonucunda hangi kolondan kaç tuttuğunun hesaplanıp gösterilmesi. \*/

public static void CekilisKontrol(int kolonlar[][], int cekilis [], int secim)

{

int tutanSayisi,artiBir;

System.out.println("Şansı Numaralar:");

for(int i = 0; i<cekilis.length; i++)

{

/\*Çekiliş sonucunda çekilen sayıları düzgün göstermek için yazılmıştır\*/

if(secim == 2 && i == 5)

{

System.out.print(" + " + cekilis[i]);

}

else

{

if(i != 0)

{

System.out.print("-" + cekilis[i]);

}

else{

System.out.print( cekilis[i]);

}

}

}

System.out.println("\n");

/\*Gerekli karşılaştırmaları yapıp sonucları gösterir\*/

for(int i = 0; i<kolonlar.length; i++)

{

tutanSayisi = 0;

artiBir = 0;

System.out.print(i+1 + ". Kolon = ");

for(int j = 0; j<kolonlar[0].length; j++)

{

System.out.print(kolonlar[i][j] + "-");

if(secim == 2 && j == 5)

{

if(kolonlar[i][j] == cekilis[j])

artiBir++;

}

else

{

for(int k = 0; k<cekilis.length; k++)

{

if(kolonlar[i][j] == cekilis[k])

tutanSayisi++;

}

}

}

System.out.println();

if(secim == 2)

{

System.out.println(i+1 + ". kolonunuzdan " + tutanSayisi + "+" + artiBir + "Tutmuştur.");

}

else

{

System.out.println(i+1 + ". kolonunuzdan " + tutanSayisi + " tutmuştur.");

}

System.out.println();

}

}

}