[1356] Earthquake Histogram

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;

namespace LWTech.ChipAnderson.EarthQuakeHistogram
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Earthquakes by Magnitude:");
         Console.WriteLine("==========================");

         List<string> quakeMagnitudes = new List<string>();

         WebClient client = new WebClient();

         try
         {
            Stream stream = client.OpenRead("https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_day.geojson");

            using (StreamReader reader = new StreamReader(stream))
            {
               while (!reader.EndOfStream)
               {
                  string line = reader.ReadLine();

                  if (line.Contains("\"mag\":"))
                  {
                     int start = line.IndexOf("\"mag\":", StringComparison.Ordinal) + 6;
                     string magString = line.Substring(start, 1);
                     quakeMagnitudes.Add(magString);
                  }
               }
            }
         }
         catch (IOException ex)
         {
            Console.WriteLine("A network error occurred.  " + ex.Message);
            Console.WriteLine("Unable to continue.");
            return;
         }

         Histogram quakeLocationHistogram = new Histogram(quakeMagnitudes, width: 100, maxLabelWidth: 5);
         quakeLocationHistogram.Sort((x,y)=>x.Value.CompareTo(y.Value));
         Console.WriteLine(quakeLocationHistogram);
      }
   }
```

```
class Histogram
{
   private int width;
   private int maxBarWidth;
   private int maxLabelWidth;
   private int minValue;
   private List<KeyValuePair<string, int>> bars;

   public Histogram(List<string> data, int width = 80, int maxLabelWidth = 10, int minValue = 0)
   {
      this.width = width;
      this.maxLabelWidth = maxLabelWidth;
      this.minValue = minValue;
      this.maxBarWidth = width - maxLabelWidth - 2;   // -2 for the space and pipe separator

      var barCounts = new Dictionary<string, int>();

      foreach (string item in data)
      {
         if (barCounts.ContainsKey(item))
            barCounts[item]++;
         else
            barCounts.Add(item, 1);
      }

      this.bars = new List<KeyValuePair<string, int>>(barCounts);

   }

   public void Sort(Comparison<KeyValuePair<string, int>> f)
   {
      bars.Sort(f);
   }

   public override string ToString()
   {
      string s = "";
      string blankLabel = "".PadRight(maxLabelWidth);

      int maxValue = 0;
      foreach (KeyValuePair<string, int> bar in bars)
      {
         if (bar.Value > maxValue)
            maxValue = bar.Value;
      }

      foreach (KeyValuePair<string, int> bar in bars)
      {
         string key = bar.Key;
         int value = bar.Value;

         if (value >= minValue)
         {
            string label;
            if (key.Length < maxLabelWidth)
               label = key.PadLeft(maxLabelWidth);
```

```
                else
                    label = key.Substring(0, maxLabelWidth);

                int barSize = (int)(((double)value / maxValue) * maxBarWidth);
                string barStars = "".PadRight(barSize, '*');

                s += label + " |" + barStars + " " + value + "\n";
            }
        }

        string axis = blankLabel + " +".PadRight(maxBarWidth + 2, '-') + "\n";
        s += axis;

        return s;
    }
  }
}
```