**Practice "Heat Map"**

In the same project, in the HeatmapTask class, write the code to prepare data for a fertility heatmap based on the day and month.

Prepare the data for building a intensity map, in which the X-axis is the day of the month, the Y-axis is the number of the month, and the point intensity (it is displayed in color and size) indicates the number of people born on this day of this month. Similar to the previous task, do not count people born on the first of any month.

For labels on X, use an array of possible numbers of the month, except for the first: from 2 to 31. For labels on Y, use an array of numbers for months from 1 (January) to 12 (December).

Thus, the data for the intensity map should be in the form of a two-dimensional array of 30 by 12 - from 2 to 31 and from January to December.

In this task, your code must work correctly on any data, not just on the set contained in the archive with the task. This is verified by secret tests.

// Paste the final content of the HeatmapTask.cs file here

**Contents of the Program.cs file**

using System;

using System.IO;

using System.Linq;

namespace Names

{

public static class Program

{

private static readonly string dataFilePath = "names.txt";

private static void Main(string[] args)

{

var namesData = ReadData();

Charts.ShowHeatmap(HeatmapTask.GetBirthsPerDateHeatmap(namesData));

// Charts.ShowHistogram(HistogramSample.GetHistogramBirthsByYear(namesData));

Charts.ShowHistogram(HistogramTask.GetBirthsPerDayHistogram(namesData, "юрий"));

Charts.ShowHistogram(HistogramTask.GetBirthsPerDayHistogram(namesData, "владимир"));

// CreativityTask.ShowYourStatistics(namesData);

Console.WriteLine();

}

private static NameData[] ReadData()

{

var lines = File.ReadAllLines(dataFilePath);

var names = new NameData[lines.Length];

for (var i = 0; i < lines.Length; i++)

names[i] = NameData.ParseFrom(lines[i]);

return names;

}

// А это более короткая версия ReadData(). Она использует механизм языка под названием Linq

// Вы можете познакомиться с ней самостоятельно: https://ulearn.azurewebsites.net/Course/Linq

// Освоив LINQ решать задачи подобные NamesTask становится гораздо проще и приятнее.

// Но это уже совсем другая история.

private static NameData[] ReadData2()

{

return File

.ReadLines(dataFilePath)

.Select(NameData.ParseFrom)

.ToArray();

}

}

}

**Contents of the Charts.cs file**

using System;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using ZedGraph;

namespace Names

{

internal static class Charts

{

public static void ShowHistogram(HistogramData stats)

{

// Графики строятся сторонней библиотекой ZedGraph. Документацию можно найти тут http://zedgraph.sourceforge.net/samples.html

// Не бойтесь экспериментировать с кодом самостоятельно!

var chart = new ZedGraphControl

{

Dock = DockStyle.Fill

};

chart.GraphPane.Title.Text = stats.Title;

chart.GraphPane.YAxis.Title.Text = "Y";

chart.GraphPane.AddBar("", Enumerable.Range(0, stats.YValues.Length).Select(i => (double) i).ToArray(),

stats.YValues, Color.Blue);

chart.GraphPane.YAxis.Scale.MaxAuto = true;

chart.GraphPane.YAxis.Scale.MinAuto = true;

chart.GraphPane.XAxis.Type = AxisType.Text;

chart.GraphPane.XAxis.Scale.TextLabels = stats.XLabels;

chart.AxisChange();

// Form — это привычное нам окно программы.

// Это одна из главных частей подсистемы под названием Windows Forms http://msdn.microsoft.com/ru-ru/library/ms229601.aspx

var form = new Form

{

Text = stats.Title,

Size = new Size(800, 600)

};

form.Controls.Add(chart);

form.ShowDialog();

}

public static void ShowHeatmap(HeatmapData stats)

{

var form = new Form

{

Text = stats.Title,

Size = new Size(800, 600)

};

form.Paint += (s, e) => DrawHeatmap(form.ClientRectangle, e.Graphics, stats);

form.ShowDialog();

}

private static void DrawHeatmap(Rectangle clientRect, Graphics g, HeatmapData data)

{

var values = data.Heat.Cast<double>().ToList();

var avgHeat = values.Average();

var sigma = Math.Sqrt(values.Average(x => (x - avgHeat) \* (x - avgHeat)));

var cellWidth = clientRect.Width / (data.XLabels.Length + 1);

var cellHeight = clientRect.Height / (data.YLabels.Length + 1);

for (var x = 0; x < data.XLabels.Length; x++)

for (var y = 0; y < data.YLabels.Length; y++)

{

var color = GetColor(data.Heat[x, y], avgHeat, sigma);

var cellRect = new Rectangle(

clientRect.Left + cellWidth \* (1 + x),

clientRect.Top + cellHeight \* y,

cellWidth,

cellHeight

);

g.FillRectangle(new SolidBrush(color), cellRect);

}

DrawLabels(g, data, cellWidth, cellHeight);

}

private static void DrawLabels(Graphics g, HeatmapData data, int cellWidth, int cellHeight)

{

var font = new Font(FontFamily.GenericMonospace, 10);

for (var x = 0; x < data.XLabels.Length; x++)

{

var text = data.XLabels[x];

var labelRect = new RectangleF(cellWidth \* (1 + x), data.YLabels.Length \* cellHeight, cellWidth,

cellHeight);

var format = new StringFormat();

format.LineAlignment = StringAlignment.Near;

format.Alignment = StringAlignment.Center;

g.DrawString(text, font, new SolidBrush(Color.Black), labelRect, format);

}

for (var y = 0; y < data.YLabels.Length; y++)

{

var text = data.YLabels[y];

var labelRect = new RectangleF(0, y \* cellHeight, cellWidth, cellHeight);

var format = new StringFormat();

format.LineAlignment = StringAlignment.Center;

format.Alignment = StringAlignment.Far;

g.DrawString(text, font, new SolidBrush(Color.Black), labelRect, format);

}

}

private static Color GetColor(double value, double avgHeat, double sigma)

{

var sigmaValue = (value - avgHeat) / sigma;

var colorValue = Math.Min(255, (int) (200 \* Math.Abs(sigmaValue)));

var color = sigmaValue >= 0

? Color.FromArgb(255, 255 - colorValue, 255, 255 - colorValue)

: Color.FromArgb(255, 255, 255 - colorValue, 255 - colorValue);

return color;

}

}

}

**Contents of the HeatmapData.cs file**

using System.Linq;

namespace Names

{

public class HeatmapData

{

public HeatmapData(string title, double[,] heat, string[] xLabels, string[] yLabels)

{

XLabels = xLabels;

YLabels = yLabels;

Title = title;

Heat = heat;

}

public string[] XLabels { get; }

public string[] YLabels { get; }

public string Title { get; }

public double[,] Heat { get; }

public bool Equals(HeatmapData other)

{

return Enumerable.Range(0, 2)

.All(dimension =>

Heat.GetLength(dimension) == other.Heat.GetLength(dimension))

&& Heat

.Cast<double>()

.SequenceEqual(other.Heat

.Cast<double>());

}

}

}

**Contents of the HeatmapTask.cs file**

using System;

namespace Names

{

internal static class HeatmapTask

{

public static HeatmapData GetBirthsPerDateHeatmap(NameData[] names)

{

return new HeatmapData(

"Пример карты интенсивностей",

new double[,] { { 1, 2, 3 }, { 2, 3, 4 }, { 3, 4, 4 }, { 4, 4, 4 } },

new[] { "a", "b", "c", "d" },

new[] { "X", "Y", "Z" });

}

}

}

**Contents of the HistogramData.cs file**

using System.Linq;

namespace Names

{

public class HistogramData

{

public HistogramData(string title, string[] xLabels, double[] yValues)

{

Title = title;

XLabels = xLabels;

YValues = yValues;

}

public string Title { get; }

public string[] XLabels { get; }

public double[] YValues { get; }

public bool Equals(HistogramData other)

{

return other.XLabels.SequenceEqual(XLabels) && other.YValues.SequenceEqual(YValues);

}

}

}

**Contents of the HistogramSample.cs file**

using System;

namespace Names

{

internal static class HistogramSample

{

// Пример подготовки данных для построения графиков:

public static HistogramData GetHistogramBirthsByYear(NameData[] names)

{

/\*

Подготовка данных для построения гистограммы

— количества людей в выборке в зависимости от года их рождения.

\*/

Console.WriteLine("Статистика рождаемости по годам");

var minYear = int.MaxValue;

var maxYear = int.MinValue;

foreach (var name in names)

{

minYear = Math.Min(minYear, name.BirthDate.Year);

maxYear = Math.Max(maxYear, name.BirthDate.Year);

}

var years = new string[maxYear - minYear + 1];

for (var y = 0; y < years.Length; y++)

years[y] = (y + minYear).ToString();

var birthsCounts = new double[maxYear - minYear + 1];

foreach (var name in names)

birthsCounts[name.BirthDate.Year - minYear]++;

return new HistogramData("Рождаемость по годам", years, birthsCounts);

}

}

}

**Contents of the HistogramTask.cs file**

using System;

using System.Linq;

namespace Names

{

internal static class HistogramTask

{

public static HistogramData GetBirthsPerDayHistogram(NameData[] names, string name)

{

var minDay = 1;

var maxDay = int.MinValue;

foreach (var day in names)

maxDay = Math.Max(maxDay, day.BirthDate.Day);

var days = new string[maxDay - minDay + 1];

for (var i = 0; i < days.Length; i++)

{

days[i] = (i + minDay).ToString();

}

var birthCounts = new double[maxDay - minDay + 1];

foreach (var day in names)

{

if (day.Name == name && day.BirthDate.Day > 1)

birthCounts[day.BirthDate.Day - minDay]++;

}

return new HistogramData(String.Format("Рождаемость людей с именем '{0}'", name), days, birthCounts);

}

}

}

**Contents of the NameData.cs file**

using System;

using System.Globalization;

namespace Names

{

public class NameData

{

/// <summary>Дата рождения</summary>

public DateTime BirthDate;

/// <summary>Имя</summary>

public string Name;

public NameData(int year, int month, int day, string name)

: this(new DateTime(year, month, day), name)

{

}

public NameData(DateTime birthDate, string name)

{

BirthDate = birthDate;

Name = name;

}

public static NameData ParseFrom(string textLine)

{

var parts = textLine.Split('\t');

const string format = "dd.MM.yyyy";

var date = DateTime.ParseExact(parts[0], format, CultureInfo.InvariantCulture);

return new NameData(date, parts[1]);

}

public override string ToString()

{

return string.Format("{0} {1}", BirthDate.ToString("dd.MM.yyyy"), Name);

}

}

}

**Code:**

using System;

namespace Names

{

internal static class HeatmapTask

{

public static string[] NomerDayGet(NameData[] names)

{

var nomerDayArr = new string[30];

for (int i = 0; i < 30; i++)

nomerDayArr[i] = (i + 2).ToString();

return nomerDayArr;

}

public static string[] NomerMounthGet(NameData[] names)

{

var nomerMounthArr = new string[12];

for (int i = 0; i < 12; i++)

{ nomerMounthArr[i] = (i + 1).ToString(); }

return nomerMounthArr;

}

public static HeatmapData GetBirthsPerDateHeatmap(NameData[] names)

{

string[] mounth = NomerMounthGet(names);

string[] day = NomerDayGet(names);

double[,] mounthDay = new double[30, 12];

foreach (var mname in names)

if (mname.BirthDate.Day != 1)

mounthDay[mname.BirthDate.Day - 2, mname.BirthDate.Month - 1]++;

return new HeatmapData("Пример карты интенсивностей", mounthDay, day, mounth);

}

}

}