Міністерство освіти і науки, молоді та спорту України

Львівський національний університет ім. Івана Франка

факультет прикладної математики та інформатики

Кафедра програмування

Індивідуальне завдання №1

з курсу «Математична статистика»

Виконала:

студентка групи ПМі-31

Прокіп Олена

Львів – 2013

1. **Постановка задачі.**

На підставі наведених вибіркових даних:

1. побудувати частотну таблицю. При цьому область реалізацій розбити на дев'ять однакових інтервалів;
2. намалювати графік частот, полігон частот, гістограму частот;
3. обчислити медіану, моду, середнє арифметичне, варіансу, стандарт, варіацію, розмах, квантилі, другий, третій, четвертий центральні моменти, асиметрію, ексцес.
4. **Короткі теоретичні відомості.**

Нехай усі елементи вибірки x1,…xn з дискретної статистичної змінної належать множині х1,…хk , де , причому елемент хі трапляється ni разів. n1+…+nk = n, де n- обсяг вибірки. Числа ni називають частотами відповідних елементів. Таблицю називають **частотною таблицею** або статистичним розподілом дискретної варіанти Х.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| X |  |  |  |  |  | … |  |
| v | n1 | n2 | n3 | n4 | n5 | … | nk |

**Середнім арифметичн****им** статистичного ряду називають суму добутків усіх варіант на відповідні частоти, поділену на суму частот.

          (1.1)

**Медіаною**Ме статистичного ряду називають значення ознаки, що припадає на середину ранжованого ряду спостережень.

**Модою**Мо статистичного ряду називають значення ознаки, якій відповідає найбільша частота.

**Дисперсією** D статистичного ряду називають середню арифметичну квадратів відхилень варіант від їх середньої арифметичної.

**Варіансою** статистичної змінної називають суму квадратів відхилень елементів статистичного матеріалу від середнього цієї вибірки, поділену на обсяг вибірки без одного.

**Станадартом** називається арифметичне значення кореня квадрата з варіанси.

**Варіацією**  вибірки з додатної мінливої величини називають відношення стандарту цієї вибірки до її середнього.

**Квантилем** вибірки порядку якщо він існує, називають той елемент варіаційного ряду, до якого включно з ним є елементів статистичного матеріалу.

**Моментом**  або **статистичним моментом порядку k відносно константи с** називають середнє арифметичне k-их степенів усіх відхилень елементів цієї вибірки.

**Асиметрією** вибірки називають відношення її третього центрального моменту до другого центрального моменту в степені півтора.

**Ексцесом**  вибірки називають відношення четвертого центрального моменту до другого центрального моменту в квадраті мінус три.

1. **Програмна реалізація**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace TIMC

{

class Calculator

{

private List<double> \_Data;

public Calculator(List<double> data)

{

\_Data = data;

\_Data.Sort();

}

public double Mediana()

{

double result;

if (\_Data.Count() % 2 != 0)

{

result = \_Data[\_Data.Count() / 2];

}

else

{

result = (\_Data[(\_Data.Count() + 1) / 2] + \_Data[(\_Data.Count() - 1) / 2]) / 2;

}

return result;

}

public List<double> Moda()

{

int count = 1;

int max = 0;

List<double> res = new List<double>();

for (int i = 1; i < \_Data.Count(); ++i)

{

if (\_Data[i - 1] == \_Data[i])

{

count++;

}

else

{

if (count > max)

{

max = count;

}

count = 1;

}

}

if (count == max)

{

res.Add(\_Data[\_Data.Count() - 1]);

}

count = 0;

for (int i = 1; i < \_Data.Count(); ++i)

{

if (\_Data[i - 1] == \_Data[i])

{

count++;

}

else

{

if (count == max)

{

res.Add(\_Data[i - 1]);

}

count = 1;

}

}

return res;

}

public double Average()

{

double result = 0;

foreach (double i in \_Data)

{

result += i;

}

return result / \_Data.Count();

}

public double Variance()

{

double result = 0;

double average = this.Average();

foreach (double i in \_Data)

{

result += (i - average) \* (i - average);

}

double a = result / (\_Data.Count() - 1);

return a;

}

public double Standart()

{

return Math.Sqrt(this.Variance());

}

public double Variation()

{

return this.Standart() / this.Average();

}

public double Scope()

{

return (\_Data.Max() - \_Data.Min());

}

private List<double> Quantiles(int n)

{

if (\_Data.Count() % n != 0)

{

return null;

}

List<double> quantiles = new List<double>();

for (int i = 1; i < n; ++i)

quantiles.Add(\_Data[\_Data.Count() \* i / n]);

return quantiles;

}

public List<double> Quartiles()

{

return this.Quantiles(4);

}

public List<double> Octiles()

{

return this.Quantiles(8);

}

public List<double> Deciles()

{

return this.Quantiles(10);

}

public List<double> Centiles()

{

return this.Quantiles(100);

}

public List<double> Mililes()

{

return this.Quantiles(1000);

}

private double Moment(int k)

{

double result = 0;

double average = this.Average();

foreach (double i in \_Data)

{

result += Math.Pow(i - average, k);

}

return result / \_Data.Count();

}

public double Moment2()

{

return this.Moment(2);

}

public double Moment3()

{

return this.Moment(3);

}

public double Moment4()

{

return this.Moment(4);

}

public double Asymmetry()

{

return (this.Moment3() / Math.Pow(this.Moment2(), 1.5));

}

public double Excess()

{

return (this.Moment4() / Math.Pow(this.Moment2(), 2) - 3);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace TIMC

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public string FilePath

{

get { return (string)GetValue(FilePathProperty); }

set { SetValue(FilePathProperty, value); }

}

// Using a DependencyProperty as the backing store for FilePath. This enables animation, styling, binding, etc...

public static readonly DependencyProperty FilePathProperty =

DependencyProperty.Register("FilePath", typeof(string), typeof(MainWindow), new PropertyMetadata("Data.txt"));

private List<double> GetData()

{

List<double> data = new List<double>();

string[] s = System.IO.File.ReadAllLines(FilePath);

for (int i = 0; i < s.Length; ++i)

{

data.Add(double.Parse(s[i]));

}

return data;

}

private Table CreateTable(List<double> data)

{

double min = data.Min();

double max = data.Max();

double length = (max - min) / 9;

data.Sort();

List<double> intervals = new List<double>();

List<int> frequencies = new List<int>();

for (int i = 0; i <= 9; ++i)

{

intervals.Add(min + i \* length);

}

for (int i = 0; i < 9; ++i)

{

frequencies.Add(0);

}

int index = 0;

for (int i = 0; i < data.Count; ++i)

{

if (data[i] <= intervals[index + 1])

{

frequencies[index]++;

}

else

{

index++;

i--;

}

}

Table table = new Table(intervals, frequencies);

return table;

}

public MainWindow()

{

InitializeComponent();

grid.DataContext = this;

createTable\_Click\_1(null, null);

calculate\_Click\_1(null, null);

}

private void createTable\_Click\_1(object sender, RoutedEventArgs e)

{

tableDrawer.DrawTable(CreateTable(GetData()));

}

private void drawPlot\_Click\_1(object sender, RoutedEventArgs e)

{

plotDrawer.DrawFrequenciesPlot(GetData());

}

private void drawPolygon\_Click\_1(object sender, RoutedEventArgs e)

{

plotDrawer.DrawFrequenciesPolygon(GetData());

}

private void drawHistogram\_Click\_1(object sender, RoutedEventArgs e)

{

plotDrawer.DrawHistogram(CreateTable(GetData()));

}

private void calculate\_Click\_1(object sender, RoutedEventArgs e)

{

Calculator c = new Calculator(GetData());

median.Content = c.Mediana().ToString();

average.Content = c.Average().ToString("0.000");

variance.Content = c.Variance().ToString("0.000");

standart.Content = c.Standart().ToString("0.000");

variation.Content = c.Variation().ToString("0.000");

scope.Content = c.Scope().ToString();

moment2.Content = c.Moment2().ToString("0.000");

moment3.Content = c.Moment3().ToString("0.000");

moment4.Content = c.Moment4().ToString("0.000");

asymmetry.Content = c.Asymmetry().ToString("0.000");

excess.Content = c.Excess().ToString("0.000");

List<double> list = new List<double>();

string s = " ";

list = c.Moda();

foreach (double d in list)

{

s = s + d.ToString() + "; ";

}

moda.Content = s;

list = c.Quartiles();

if (list != null)

{

s = " ";

foreach (double d in list)

{

s = s + d.ToString() + "; ";

}

quartiles.Content = s;

}

else quartiles.Content = "нема";

list = c.Octiles();

if (list != null)

{

s = " ";

foreach (double d in list)

{

s = s + d.ToString() + "; ";

}

octiles.Content = s;

}

else octiles.Content = "нема";

list = c.Deciles();

if (list != null)

{

s = " ";

foreach (double d in list)

{

s = s + d.ToString() + "; ";

}

deciles.Content = s;

System.IO.File.WriteAllText(@"WriteText.txt", s);

}

else deciles.Content = "нема";

list = c.Centiles();

if (list != null)

{

s = " ";

for (int j = 1; j <= 10; ++j)

{

for (int i = 0; i < 10; ++i)

{

if (((j \* 10) + i - 10) != 99)

s = s + list[(j \* 10) + i - 10].ToString() + "; ";

}

if (j != 10)

{

s += "\n";

}

}

centiles.Content = s;

}

else centiles.Content = "нема";

list = c.Mililes();

if (list != null)

{

s = " ";

foreach (double d in list)

{

s = s + d.ToString() + "; ";

}

mililes.Content = s;

}

else mililes.Content = "нема";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Collections.ObjectModel;

namespace TIMC

{

/// <summary>

/// Interaction logic for PlotDrawer.xaml

/// </summary>

public partial class PlotDrawer : UserControl

{

public PlotDrawer()

{

InitializeComponent();

plot = new ObservableCollection<UIElement>();

Width = 800;

Height = 500;

grid.DataContext = this;

Stroke = Brushes.Black;

StrokeThickness=1;

}

private List<Tuple<double, int>> GetFrequencies(List<double> data)

{

data.Sort();

List<Tuple<double, int>> result = new List<Tuple<double, int>>();

int counter = 1;

for (int i = 1; i < data.Count; ++i)

{

if (data[i] == data[i - 1])

{

counter++;

}

else

{

result.Add(new Tuple<double, int>(data[i-1], counter));

counter = 1;

}

}

result.Add(new Tuple<double, int>(data[data.Count-1], counter));

return result;

}

public ObservableCollection<UIElement> plot

{

get { return (ObservableCollection<UIElement>)GetValue(plotProperty); }

set { SetValue(plotProperty, value); }

}

// Using a DependencyProperty as the backing store for plot. This enables animation, styling, binding, etc...

public static readonly DependencyProperty plotProperty =

DependencyProperty.Register("plot", typeof(ObservableCollection<UIElement>), typeof(PlotDrawer), new UIPropertyMetadata(new ObservableCollection<UIElement>()));

Brush Stroke { get; set; }

double StrokeThickness { get; set; }

double dx, dy;

double sx, sy;

double Sx, Sy;

double minx, maxx, miny, maxy;

private Point Transform(double x, double y)

{

return new Point(x \* sx \* Sx + dx, y \* sy \* Sy + dy);

}

private Line GetDashedLine(double x1, double y1, double x2, double y2)

{

Point A = Transform(x1,y1);

Point B = Transform(x2,y2);

Line line = new Line()

{

X1 = A.X,

Y1 = A.Y,

X2 = B.X,

Y2 = B.Y,

Stroke = Stroke,

StrokeThickness = StrokeThickness/2,

StrokeDashArray = { 4, 4 }

};

return line;

}

private Line GetWideLine(double x1, double y1, double x2, double y2)

{

Point A = Transform(x1, y1);

Point B = Transform(x2, y2);

Line line = new Line()

{

X1 = A.X,

Y1 = A.Y,

X2 = B.X,

Y2 = B.Y,

Stroke = Stroke,

StrokeThickness = StrokeThickness \* 3

};

return line;

}

private Line GetColoredLine(double x1, double y1, double x2, double y2)

{

Point A = Transform(x1, y1);

Point B = Transform(x2, y2);

Line line = new Line()

{

X1 = A.X,

Y1 = A.Y,

X2 = B.X,

Y2 = B.Y,

Stroke = Brushes.Green,

StrokeThickness = StrokeThickness \* 2

};

return line;

}

private Line GetColoredDashedLine(double x1, double y1, double x2, double y2)

{

Point A = Transform(x1, y1);

Point B = Transform(x2, y2);

Line line = new Line()

{

X1 = A.X,

Y1 = A.Y,

X2 = B.X,

Y2 = B.Y,

Stroke = Brushes.Green,

StrokeDashArray = { 4, 4 },

StrokeThickness = StrokeThickness \* 2

};

return line;

}

private Line GetLine(double x1, double y1, double x2, double y2)

{

Point A = Transform(x1,y1);

Point B = Transform(x2,y2);

Line line = new Line()

{

X1 = A.X,

Y1 = A.Y,

X2 = B.X,

Y2 = B.Y,

Stroke = Stroke,

StrokeThickness = StrokeThickness

};

return line;

}

private Rectangle GetRectangle(double x, double y)

{

Point a = Transform(x, y);

Point b = Transform(x, 0);

Rectangle rectangle = new Rectangle()

{

Margin = new Thickness(a.X - 2, a.Y, 0, 0),

Width = 4,

Height = b.Y - a.Y,

Stroke = Brushes.Blue,

StrokeThickness = 2,

Fill = Brushes.LightBlue

};

return rectangle;

}

private bool IsInside(double x, double y)

{

Point a = Transform(x, y);

if (a.X < 0 || a.Y < 0 || a.X >= Width || a.Y >= Height) return false;

return true;

}

private void DrawAxes()

{

plot.Add(GetWideLine(0, miny, 0, maxy));

plot.Add(GetWideLine(minx, 0, maxx, 0));

double d = (maxx - minx) / 10;

for (int i = -10; i <= 10; i++)

{

if (IsInside(d \* i, 0))

{

plot.Add(GetDashedLine(d \* i, miny, d \* i, maxy));

Label t = new Label();

t.Content = (i \* d).ToString("0.0");

Point p = Transform(d\*i,0);

t.Margin = new Thickness(p.X-1, p.Y-1, 0, 0);

plot.Add(t);

}

}

d = (maxy - miny) / 10;

for (int i = -10; i <= 10; i++)

{

if (IsInside(0, d \* i))

{

plot.Add(GetDashedLine(minx, d \* i, maxx, d \* i));

Label t = new Label();

t.Content = (i \* d).ToString("0.0");

Point p = Transform(0, d \* i);

t.Margin = new Thickness(p.X - 1, p.Y - 1, 0, 0);

plot.Add(t);

}

}

}

private List<Tuple<double, int>> Prepare(List<double> data)

{

plot.Clear();

List<Tuple<double, int>> frequencies = GetFrequencies(data);

maxy = 0;

miny = 0;

for (int i = 0; i < frequencies.Count; i++)

{

maxy = Math.Max(maxy, frequencies[i].Item2);

}

maxx = Math.Max(data.Max(), 0);

minx = Math.Min(data.Min(), 0);

minx--;

maxx++;

miny--;

maxy++;

while ((maxy - miny) % 10 != 0) maxy++;

int min = (int)minx;

int max = (int)maxx;

while ((max - min) % 10 != 0)

{

max++;

if ((max - min) % 10 != 0) min--;

}

maxx = max;

minx = min;

sx = 1;

sy = -1;

dx = -minx \* Width / (maxx - minx);

dy = maxy \* Height / (maxy - miny);

Sx = Math.Min((Width - dx) / (sx \* maxx), -dx / (sx \* minx));

Sy = Math.Min((Height - dy) / (sy \* miny), -dy / (sy \* maxy));

DrawAxes();

return frequencies;

}

private void Prepare(Table data)

{

plot.Clear();

maxy = 0;

miny = 0;

for (int i = 0; i < data.Frequencies.Count; i++)

{

maxy = Math.Max(maxy, data.Frequencies[i]);

}

maxx = 0;

minx = 0;

for (int i = 0; i < data.Intervals.Count; i++)

{

maxx = Math.Max(maxx, data.Intervals[i]);

minx = Math.Min(minx, data.Intervals[i]);

}

minx--;

maxx++;

miny--;

maxy++;

while ((maxy - miny) % 10 != 0) maxy++;

int min = (int)minx;

int max = (int)maxx;

while ((max - min) % 10 != 0)

{

max++;

if ((max - min) % 10 != 0) min--;

}

maxx = max;

minx = min;

sx = 1;

sy = -1;

dx = -minx \* Width / (maxx - minx);

dy = maxy \* Height / (maxy - miny);

Sx = Math.Min((Width - dx) / (sx \* maxx), -dx / (sx \* minx));

Sy = Math.Min((Height - dy) / (sy \* miny), -dy / (sy \* maxy));

DrawAxes();

}

public void DrawFrequenciesPlot(List<double> data)

{

List<Tuple<double, int>> frequencies = Prepare(data);

for (int i = 0; i < frequencies.Count; i++)

{

plot.Add(GetColoredLine(frequencies[i].Item1, frequencies[i].Item2, frequencies[i].Item1, 0));

}

}

public void DrawFrequenciesPolygon(List<double> data)

{

List<Tuple<double, int>> frequencies = Prepare(data);

for (int i = 1; i < frequencies.Count; i++)

{

plot.Add(GetColoredLine(frequencies[i - 1].Item1, frequencies[i - 1].Item2, frequencies[i].Item1, frequencies[i].Item2));

}

}

public void DrawHistogram(Table data)

{

Prepare(data);

for (int i = 0; i < data.Frequencies.Count; i++)

{

plot.Add(GetColoredLine(data.Intervals[i], data.Frequencies[i], data.Intervals[i + 1], data.Frequencies[i]));

if (i != data.Frequencies.Count - 1)

{

plot.Add(GetColoredDashedLine(data.Intervals[i + 1], data.Frequencies[i], data.Intervals[i + 1], data.Frequencies[i + 1]));

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace TIMC

{

/// <summary>

/// Interaction logic for Table.xaml

/// </summary>

public partial class TableDrawer : UserControl

{

public TableDrawer()

{

InitializeComponent();

}

private string GetInterval(double a, double b)

{

return "(" + a.ToString("0.0") + " ; " + b.ToString("0.0") + "]";

}

private void CreateTable(int n)

{

grid.ColumnDefinitions.Clear();

grid.RowDefinitions.Clear();

grid.Children.Clear();

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

{

ColumnDefinition c = new ColumnDefinition();

c.Width = new GridLength(74);

grid.ColumnDefinitions.Add(c);

}

for (int i = 0; i < 2; ++i)

{

RowDefinition c = new RowDefinition();

c.Height = new GridLength(47);

grid.RowDefinitions.Add(c);

}

}

public void DrawTable(Table table)

{

int n = table.Frequencies.Count;

CreateTable(n);

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

{

TextBlock tb1 = new TextBlock();

tb1.VerticalAlignment = System.Windows.VerticalAlignment.Center;

tb1.HorizontalAlignment = System.Windows.HorizontalAlignment.Center;

tb1.Text = GetInterval(table.Intervals[i], table.Intervals[i + 1]);

Grid.SetColumn(tb1, i);

Grid.SetRow(tb1, 0);

grid.Children.Add(tb1);

TextBlock tb2 = new TextBlock();

tb2.VerticalAlignment = System.Windows.VerticalAlignment.Center;

tb2.HorizontalAlignment = System.Windows.HorizontalAlignment.Center;

tb2.Text = table.Frequencies[i].ToString();

Grid.SetColumn(tb2, i);

Grid.SetRow(tb2, 1);

grid.Children.Add(tb2);

}

}

}

}

1. **Аналіз отриманих результатів.**

Отримали такі результати обчислень:

**Медіана : -0.75**

**Мода : -4.1**

**Середнє арифметичне : -0.148**

**Варіанса : 35.166**

**Стандарт : 5.930**

**Варіація : -40.068**

**Квартилі: -4.5; -0.7; 5.2;**

**Октилі: нема**

**Децилі:** -8,1; -5,2; -4,2; -2,5; -0,7; 1,8; 3,8; 6,2; 8,2;

**Центилі:** -9,8; -9,8; -9,6; -9,2; -9,2; -9; -9; -9; -8,5; -8,1; -8,1; -8,1; -8; -8; -6,9; -6,8; -6,4; -6; -5,6; -5,2; -5,2; -4,9; -4,8; -4,7; -4,5; -4,5; -4,4; -4,3; -4,3; -4,2; -4,1; -4,1; -4,1; -4,1; -3,6; -3; -2,8; -2,6; -2,5; -2,5; -2,3; -2,1; -2; -1,9; -1,8; -1,5; -1,5; -0,9; -0,8; -0,7; 0,2; 0,4; 0,6; 0,6; 0,8; 1,4; 1,5; 1,6; 1,7; 1,8; 2,4; 2,5; 2,5; 2,6; 2,8; 2,9; 3,4; 3,5; 3,8; 3,8; 4; 4,1; 4,2; 5,1; 5,2; 5,4; 5,6; 5,7; 5,9; 6,2; 6,4; 6,5; 6,6; 7,2; 7,3; 8; 8,1; 8,1; 8,2; 8,2; 8,3; 8,9; 8,9; 9,1; 9,5; 9,5; 9,5; 9,6; 9,9;

**Мілілі: нема**

**Другий центральний момент : 34.814**

**Третій центральний момент : 11.716**

**Четевертий центральний момент : 2201.926**

**Асиметрія : 0.057 ,** є близькою до нуля, отже статистичний матеріал розташований симетрично відносно середини інтервалу.

**Ексцес : -1.183 ,** є меншим за нуль, отже статистичний матеріал низьковершинний.

Це також доводить нам результат частотної таблиці.

1. **Висновки.**

Виконуючи це завдання, я навчилася обчислювати числові характеристики статистичного матеріалу.

Отримані числові характеристики є зручними та корисними для використання.

Щоб охарактеризувати вибірку нам більше не потрібно звертатись до статистичного матеріалу, адже це досить велика робота, достатньо проаналізувати отримані результати.