

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

дисциплина: Компьютерный практикум
по статистическому данным анализ

Студент: Доре Стевенсон Эдгар

Группа: НКН-бд-01-19

МОСКВА

2023 г.

Лабораторная работа №7. Введение в работу с данными

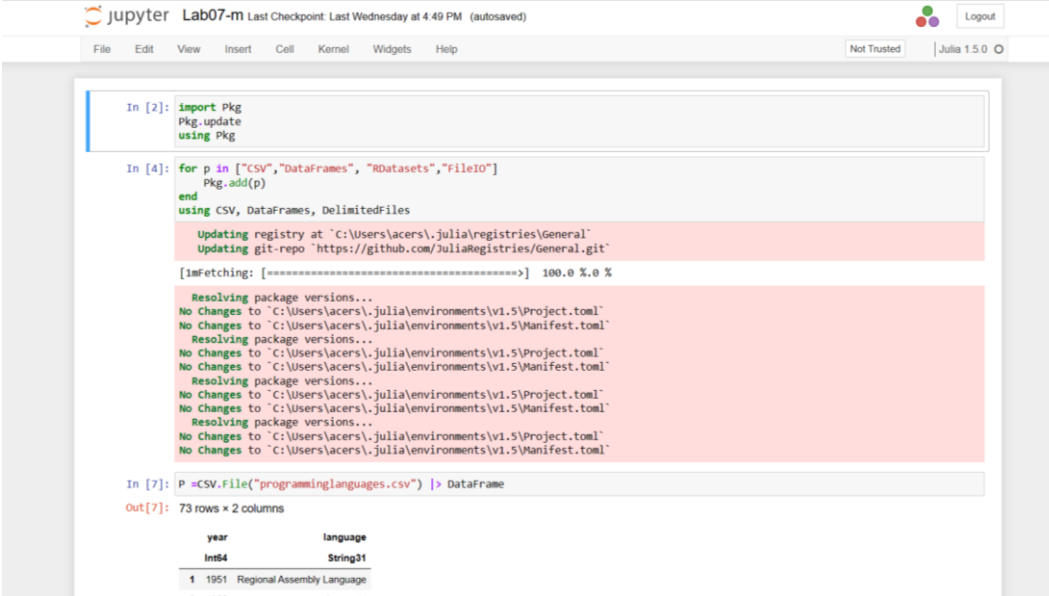
Цель работы:

Основной целью работы является специализированных пакетов Julia для обработки данных.

Ход работы:

7.2.1. Julia для науки о данных

7.2.1.1. Считывание данных



The screenshot shows a JupyterLab environment with the following components:

- Header:** "jupyter Lab07-m Last Checkpoint: Last Wednesday at 4:49 PM (autosaved)" and a "Logout" button.
- Menu Bar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help.
- Right Panel:** "Not Trusted" and "Julia 1.5.0".
- Code Cells:**
 - In [2]:** `import Pkg; Pkg.update; using Pkg`
 - In [4]:** `for p in ["CSV", "DataFrames", "RDatasets", "FileIO"]; Pkg.add(p); end; using CSV, DataFrames, DelimitedFiles`. This cell shows output for updating the registry and resolving package versions.
 - In [7]:** `P = CSV.File("programminglanguages.csv") > DataFrame`
- Output [7]:** "73 rows × 2 columns" followed by a table:

	year	language
	Int64	String31
1	1951	Regional Assembly Language

```
In [7]: P = CSV.File("programminglanguages.csv") |> DataFrame
```

Out[7]: 73 rows × 2 columns

	year	language
	Int64	String31
1	1951	Regional Assembly Language
2	1952	Autocode
3	1954	IPL
4	1955	FLOW-MATIC
5	1957	FORTRAN
6	1957	COMTRAN
7	1958	LISP
8	1958	ALGOL 58
9	1959	FACT
10	1959	COBOL
11	1959	RPG
12	1962	APL
13	1962	Simula
14	1962	SNOBOL
15	1963	CPL
16	1964	Speakeasy
17	1964	BASIC
18	1964	PL/I
19	1966	JOSS
20	1967	BCPL
21	1968	Logo

```
In [5]: 1 # Функция определения по названию языка программирования года его создания:
2 function language_created_year(P, language::String)
3     loc = findfirst(P[:,2].==language)
4     return P[loc,1]
5 end
```

Out[5]: language_created_year (generic function with 1 method)

```
In [6]: 1 # Пример вызова функции и определение даты создания языка Python:
2 language_created_year(P, "Python")
```

Out[6]: 1991

```
In [7]: 1 # Пример вызова функции и определение даты создания языка Julia:
2 language_created_year(P, "Julia")
```

Out[7]: 2012

```
In [8]: 1 #В следующем примере при вызове функции, в качестве аргумента которой указано
2 #слово julia, написанное со строчной буквы:
3 language_created_year(P, "julia")
4 #поэтому выходит ошибка
```

```
MethodError: no method matching getindex(::DataFrame, ::Nothing, ::Int64)
Closest candidates are:
  getindex(::DataFrame, !Matched::Colon, ::Union{AbstractString, Signed, Symbol, Unsigned}) at C:\Users\Admin\.julia\packages\DataFrames\GtZ11\src\dataframe\dataframe.jl:420
  getindex(::DataFrame, !Matched::typeof(!), ::Union{Signed, Unsigned}) at C:\Users\Admin\.julia\packages\DataFrames\GtZ11\src\dataframe\dataframe.jl:426
  getindex(::DataFrame, !Matched::Integer, ::Union{Signed, Unsigned}) at C:\Users\Admin\.julia\packages\DataFrames\GtZ11\src\dataframe\dataframe.jl:420
```

```
In [9]: 1 # Функция определения по названию языка программирования
        2 # года его создания (без учёта регистра):
        3 function language_created_year_v2(P,language::String)
        4     loc = findfirst(lowercase.(P[:,2]).==lowercase.(language))
        5     return P[loc,1]
        6 end
        7 language_created_year_v2(P,"julia")
```

Out[9]: 2012

```
In [10]: 1 # Пример вызова функции и определение даты создания языка julia:
        2 language_created_year_v2(P,"julia")
        3
```

Out[10]: 2012

```
In [11]: 1 # Построчное считывание данных с указанием разделителя:
        2 Tx = readlm("programminglanguages.csv", ',')
```

```
Out[11]: 74x2 Array{Any,2}:
          "year"  "language"
1951      "Regional Assembly Language"
1952      "Autocode"
1954      "IPL"
1955      "FLOW-MATIC"
1957      "FORTRAN"
1957      "COMTRAN"
1958      "LISP"
1958      "ALGOL 58"
1959      "FACT"
1959      "COBOL "
1959      "RPG"
```

7.2.1.2. Запись данных в файл

7.2.1.2. Запись данных в файл

```
In [12]: 1 # Запись данных в CSV-файл:
        2 CSV.write("programming_languages_data2.csv", P)
```

Out[12]: "programming_languages_data2.csv"

```
In [13]: 1 # Пример записи данных в текстовый файл с разделителем ',':
        2 writedlm("programming_languages_data.txt", Tx, ',')
```

```
In [14]: 1 # Пример записи данных в текстовый файл с разделителем '-':
        2 writedlm("programming_languages_data2.txt", Tx, '-')
```

```
In [15]: 1 # Построчное считывание данных с указанием разделителя:
        2 P_new_delim = readldm("programming_languages_data2.txt", '-')
```

```
Out[15]: 74x2 Array{Any,2}:
          "year"  "language"
1951      "Regional Assembly Language"
1952      "Autocode"
1954      "IPL"
1955      "FLOW-MATIC"
1957      "FORTRAN"
1957      "COMTRAN"
1958      "LISP"
1958      "ALGOL 58"
1959      "FACT"
1959      "COBOL"
1959      "RPG"
1962      "APL"
          .
```

7.2.1.3. Словари

7.2.1.3. Словари

```
In [16]: 1 # Инициализация словаря:
        2 dict = Dict{Integer,Vector{String}}{}
```

Out[16]: Dict{Integer,Array{String,1}}{}

```
In [17]: 1 # Инициализация словаря:
        2 dict2 = Dict{}
```

Out[17]: Dict{Any,Any}()

```
In [18]: 1 # Заполнение словаря данными:
        2 for i = 1:size(P,1)
        3     year,lang = P[i,:]
        4
        5     if year in keys(dict)
        6         dict[year] = push!(dict[year],lang)
        7     else
        8         dict[year] = [lang]
        9     end
       10 end
```

```
In [19]: 1 # Пример определения в словаре языков программирования, созданных в 2003 году:
        2 dict[2003]
```

Out[19]: 2-element Array{String,1}:
"Groovy"
"Scala"

7.2.1.4. DataFrames

7.2.1.4. DataFrames

```
In [20]: 1 # Подгружаем пакет DataFrames:
          2 using DataFrames
```

```
In [21]: 1 # Задаём переменную со структурой DataFrame:
          2 df = DataFrame(year = P[:,1], language = P[:,2])
```

Out[21]: 73 rows × 2 columns

	year	language
	Int64	String
1	1951	Regional Assembly Language
2	1952	Autocode
3	1954	IPL
4	1955	FLOW-MATIC
5	1957	FORTRAN
6	1957	COMTRAN
7	1958	LISP
8	1958	ALGOL 58
9	1959	FACT
10	1959	COBOL
11	1959	RPG
12	1962	APL

```
In [22]: 1 # Вывод всех значения столбца year:
          2 df[:,year]
```

Out[22]: 73-element Array{Int64,1}:
1951
1952
1954
1955
1957
1957
1958
1958
1959
1959
1959
1962
1962
:
2003
2005
2006
2007
2009
2010
2011
2011
2011
2011
2012
2014

```
In [23]: 1 # Получение статистических сведений о фрейме:
          2 describe(df)
```

Out[23]: 2 rows × 8 columns

	variable	mean	min	median	max	nunique	nmissing	eltype
	Symbol	Union...	Any	Union...	Any	Union...	Nothing	DataType
1	year	1982.99	1951	1986.0	2014			Int64
2	language		ALGOL 58		dBase III	73		String

7.2.1.5. RDatasets

```
In [24]: 1 # Подгружаем пакет RDatasets:
          2 using RDatasets
```

```
In [25]: 1 # Задаём структуру данных в виде набора данных:
          2 iris = dataset("datasets", "iris")
```

Out[25]: 150 rows × 5 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
	Float64	Float64	Float64	Float64	Cat...
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa

```
In [26]: 1 # Определения типа переменной:
        2 typeof(iris)
```

Out[26]: DataFrame

```
In [27]: 1 #Пакет RDatasets также предоставляет возможность с помощью description получить
        2 #основные статистические сведения о каждом столбце в наборе данных:
        3 describe(iris)
```

Out[27]: 5 rows × 8 columns

	variable	mean	min	median	max	nunique	nmissing	eltype
	Symbol	Union...	Any	Union...	Any	Union...	Nothing	DataType
1	SepalLength	5.84333	4.3	5.8	7.9			Float64
2	SepalWidth	3.05733	2.0	3.0	4.4			Float64
3	PetalLength	3.758	1.0	4.35	6.9			Float64
4	PetalWidth	1.19933	0.1	1.3	2.5			Float64
5	Species		setosa		virginica	3		CategoricalValue{String, UInt8}

7.2.1.6. Работа с переменными отсутствующего типа (Missing Values)

7.2.1.6. Работа с переменными отсутствующего типа (Missing Values)

```
In [28]: 1 # Отсутствующий тип:
        2 a = missing
        3 typeof(a)
```

Out[28]: Missing

```
In [29]: 1 # Пример операции с переменной отсутствующего типа:
        2 a + 1
```

Out[29]: missing

```
In [30]: 1 # Определение перечня продуктов:
        2 foods = ["apple", "cucumber", "tomato", "banana"]
```

Out[30]: 4-element Array{String,1}:
"apple"
"cucumber"
"tomato"
"banana"

```
In [31]: 1 # Определение калорий:
        2 calories = [missing, 47, 22, 105]
```

Out[31]: 4-element Array{Union{Missing, Int64},1}:
missing
47
22
105


```
In [32]: 1 # Определение типа переменной:
         2 typeof(calories)
```

Out[32]: Array{Union{Missing, Int64},1}

```
In [33]: 1 # Подключаем пакет Statistics:
         2 using Statistics
```

```
In [34]: 1 # Определение среднего значения:
         2 mean(calories)
```

Out[34]: missing

```
In [35]: 1 # Определение среднего значения без значений с отсутствующим типом:
         2 mean(skipmissing(calories))
```

Out[35]: 58.0

```
In [36]: 1 # Задание сведений о ценах:
         2 prices = [0.85,1.6,0.8,0.6]
```

Out[36]: 4-element Array{Float64,1}:
0.85
1.6
0.8
0.6

```
In [37]: 1 # Формирование данных о калориях:
          2 dataframe_calories = DataFrame(item=foods,calories=calories)
```

Out[37]: 4 rows × 2 columns

	item	calories
	String	Int64?
1	apple	missing
2	cucumber	47
3	tomato	22
4	banana	105

```
In [38]: 1 # Формирование данных о ценах:
          2 dataframe_prices = DataFrame(item=foods,price=prices)
```

Out[38]: 4 rows × 2 columns

	item	price
	String	Float64
1	apple	0.85
2	cucumber	1.6
3	tomato	0.8
4	banana	0.6

```
In [39]: 1 # Объединение данных о калориях и ценах:
          2 DF = join(dataframe_calories,dataframe_prices,on=:item)
```

Out[39]: 4 rows × 3 columns

	item	calories	price
	String	Int64?	Float64
1	apple	missing	0.85
2	cucumber	47	1.6
3	tomato	22	0.8
4	banana	105	0.6

7.2.1.7. FileIO

7.2.1.7. FileIO

```
In [40]: 1 # Подключаем пакет FileIO:
        2 using FileIO
```

```
In [41]: 1 julialogo = download("https://avatars0.githubusercontent.com/u/743164?s=200&v=4", "julialogo.png")
```

```
Out[41]: "julialogo.png"
```

```
In [42]: 1 # Подключаем пакет ImageIO:
        2 import Pkg
        3 Pkg.add("ImageIO")
```

```
Updating registry at `C:\Users\Admin\.julia\registries\General`
Resolving package versions...
No Changes to `C:\Users\Admin\.julia\environments\v1.5\Project.toml`
No Changes to `C:\Users\Admin\.julia\environments\v1.5\Manifest.toml`
```

```
In [43]: 1 # Загрузка изображения:
        2 X1 = load("julialogo.png")
```

```
Out[43]: 200x200 Array{RGBA{N0f8},2} with eltype ColorTypes.RGBA{FixedPointNumbers.Normed{UInt8,8}}:
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
RGBA{N0f8}(0.0,0.0,0.0,0.0) ... RGBA{N0f8}(0.0,0.0,0.0,0.0)
```

```
In [44]: 1 # Определение типа и размера данных:
        2 @show typeof(X1);
        3 @show size(X1);
```

```
typeof(X1) = Array{ColorTypes.RGBA{FixedPointNumbers.Normed{UInt8,8}},2}
size(X1) = (200, 200)
```

7.2.2. Обработка данных: стандартные алгоритмы машинного обучения в Julia

7.2.2.1. Кластеризация данных. Метод k-средних

7.2.2. Обработка данных: стандартные алгоритмы машинного обучения в Julia

7.2.2.1. Кластеризация данных. Метод k-средних

```
In [1]: 1 # Загрузка пакетов:  
2 #import Pkg  
3 #Pkg.add("DataFrames")  
4 #Pkg.add("Statistics")  
5 using DataFrames  
6 using CSV  
7 #import Pkg  
8 #Pkg.add("Plots")
```

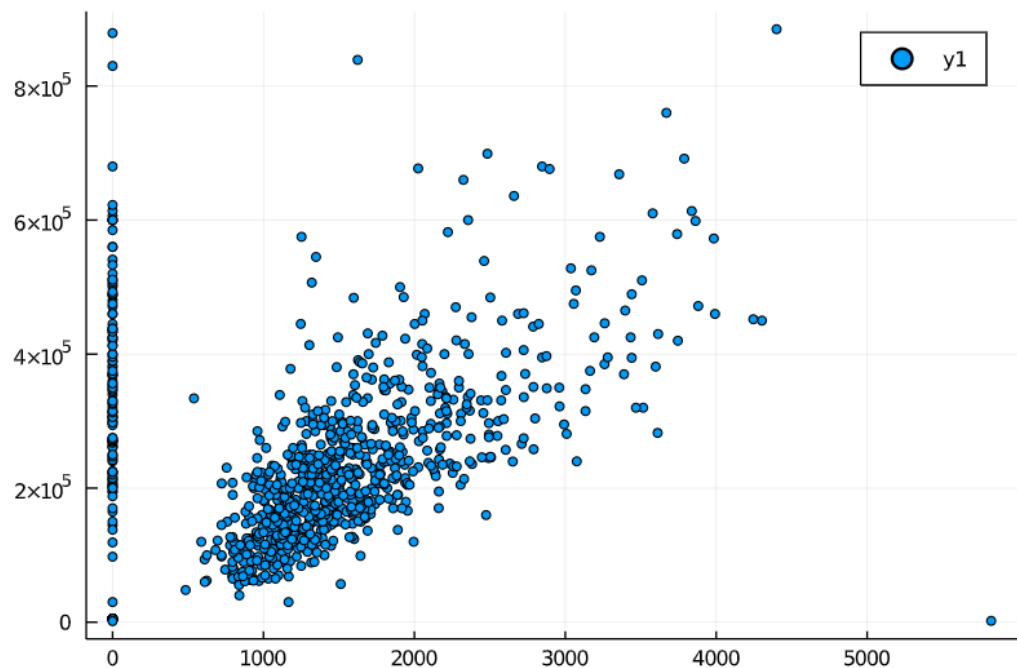
```
In [2]: 1 # Загрузка данных:  
2 houses = CSV.File("houses.csv") |> DataFrame
```

Out[2]: 985 rows × 12 columns (omitted printing of 5 columns)

	street	city	zip	state	beds	baths	sq__ft
	String	String	Int64	String	Int64	Int64	Int64
1	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836
2	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167
3	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796
4	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852
5	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797
6	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122

```
In [3]: 1 # Построение графика:  
2 using Plots  
3 plot(size=(500,500),leg=false)  
4 x = houses[:sq__ft]  
5 y = houses[:price]  
6 scatter(x,y,markersize=3)
```

Out[3]:



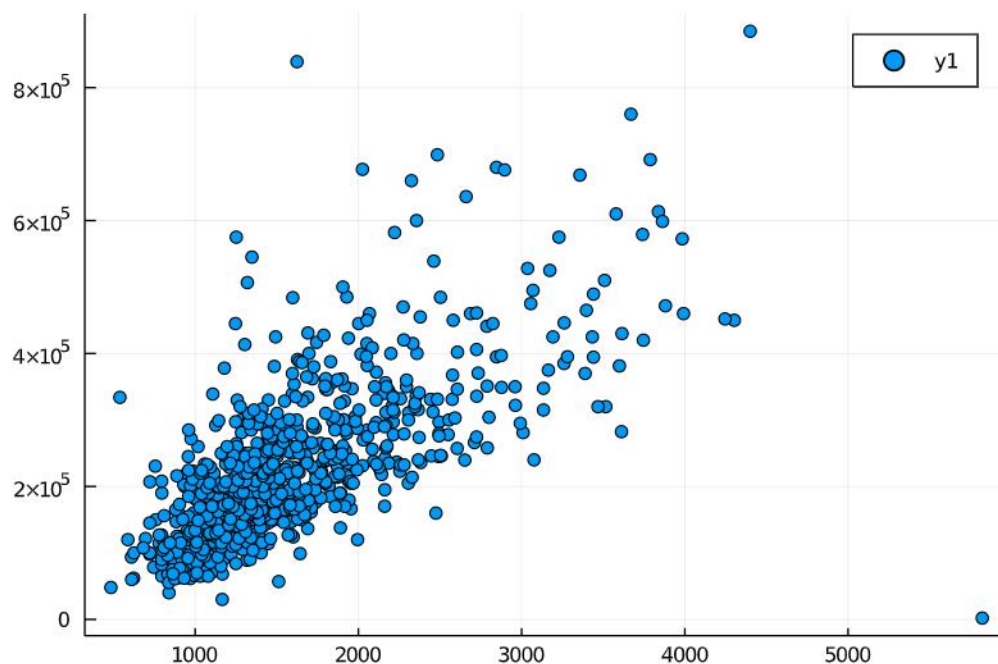
```
In [4]: 1 # Фильтрация данных по заданному условию:
        2 filter_houses = houses[houses[:sq_ft].>0,:]
```

Out[4]: 814 rows × 12 columns (omitted printing of 5 columns)

	street	city	zip	state	beds	baths	sq_ft
	String	String	Int64	String	Int64	Int64	Int64
1	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836
2	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167
3	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796
4	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852
5	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797
6	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122
7	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104
8	2561 19TH AVE	SACRAMENTO	95820	CA	3	1	1177
9	11150 TRINITY RIVER DR Unit 114	RANCHO CORDOVA	95670	CA	2	2	941
10	7325 10TH ST	RIO LINDA	95673	CA	3	2	1146
11	645 MORRISON AVE	SACRAMENTO	95838	CA	3	2	909
12	4085 FAWN CIR	SACRAMENTO	95823	CA	3	2	1289
13	2930 LA ROSA RD	SACRAMENTO	95815	CA	1	1	871
14	2113 KIRK WAY	SACRAMENTO	95822	CA	3	1	1020
15	4533 LOCH HAVEN WAY	SACRAMENTO	95842	CA	2	2	1022

```
In [5]: 1 # Построение графика:
        2 x = filter_houses[:sq_ft]
        3 y = filter_houses[:price]
        4 scatter(x,y)
```

Out[5]:



```
In [6]: 1 # Подключение пакета Statistics:
        2 using Statistics
```

```
In [7]: 1 # Определение средней цены для определённого типа домов:
        2 by(filter_houses, :type, filter_houses -> mean(filter_houses[:price]))
```

Out[7]: 3 rows × 2 columns

	type	x1
	String	Float64
1	Residential	2.34802e5
2	Condo	1.34213e5
3	Multi-Family	2.24535e5

```
In [8]: 1 # Подключение пакета Clustering:
        2 #import Pkg
        3 #Pkg.add("Clustering")
        4 using Clustering
```

```
In [9]: 1 # Добавление данных :latitude и :longitude в новый фрейм:
        2 X = filter_houses[:, :latitude, :longitude]
```

Out[9]: 814 rows × 2 columns

	latitude	longitude
	Float64	Float64
1	38.6319	-121.435
2	38.4789	-121.431

```
In [10]: 1 # Конвертация данных в матричный вид:
        2 X = convert(Matrix{Float64}, X)
```

```
Out[10]: 814x2 Array{Float64,2}:
```

```
38.6319 -121.435
38.4789 -121.431
38.6183 -121.444
38.6168 -121.439
38.5195 -121.436
38.6626 -121.328
38.6817 -121.352
38.5351 -121.481
38.6212 -121.271
38.7009 -121.443
38.6377 -121.452
38.4707 -121.459
38.6187 -121.436
⋮
38.7035 -121.375
38.7031 -121.235
38.3898 -121.446
38.8978 -121.325
38.4679 -121.445
38.4453 -121.442
38.4174 -121.484
38.4577 -121.36
38.4999 -121.459
38.7088 -121.257
38.417 -121.397
38.6552 -121.076
```

```
In [11]: 1 # Транспонирование матрицы с данными:
        2 X = X'
```

```
Out[11]: 2x814 LinearAlgebra.Adjoint{Float64,Array{Float64,2}}:
38.6319 38.4789 38.6183 ... 38.7088 38.417 38.6552
-121.435 -121.431 -121.444 -121.257 -121.397 -121.076
```

```
In [12]: 1 # Задание количества кластеров:
        2 k = length(unique(filter_houses[:zip]))
```

```
Out[12]: 66
```

```
In [13]: 1 # Определение k-среднего:
        2 C = kmeans(X,k) # попробуйте поменять k
```

```
Out[13]: KmeansResult{Array{Float64,2},Float64,Int64}([38.680849300000006 38.284105999999994 ... 38.51819435714285 38.251295999999996; -12
1.36293816666664 -121.29519644444443 ... -121.49720271428573 -121.31266899999999], [35, 47, 35, 35, 10, 41, 1, 46, 60, 45 ... 27,
36, 47, 62, 19, 61, 47, 23, 7, 28], [6.273316466831602e-5, 0.0005054545472376049, 7.037215982563794e-5, 6.84302649460733e-5, 0.
00012161941776867025, 0.0001970666489796713, 0.00012683964814641513, 0.00023977080854820088, 1.144897032645531e-5, 0.0001168665
8399412408 ... 3.845047467621043e-5, 5.2413251978578046e-5, 0.000216632652154658, 0.00023574619626742788, 0.0001341583774774335
3, 0.00012178530232631601, 0.0004134149457968306, 0.0005266233965812717, 7.109141370165162e-5, 0.0014500179277092684], [30, 9,
5, 25, 1, 1, 10, 13, 5, 22 ... 15, 18, 2, 9, 34, 36, 1, 1, 14, 6], [30, 9, 5, 25, 1, 1, 10, 13, 5, 22 ... 15, 18, 2, 9, 34, 36,
1, 1, 14, 6], 0.2148637686877919, 15, true)
```

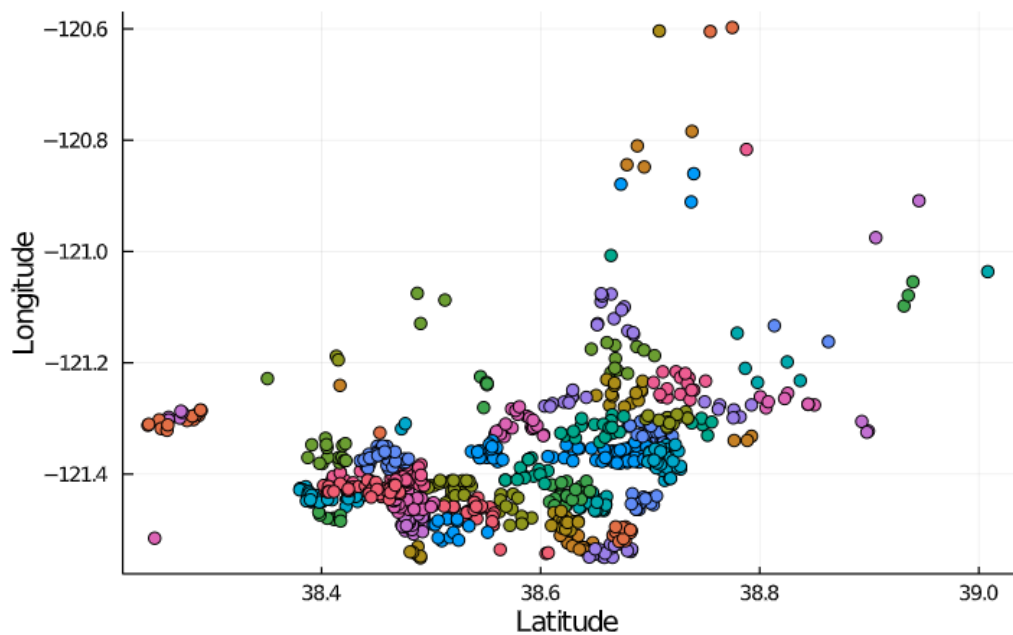
```
In [14]: 1 # Формирование фрейма данных:
2 df = DataFrame(cluster = C.assignments,city = filter_houses[:city],
3               latitude = filter_houses[:latitude],longitude = filter_houses[:longitude],zip = filter_houses[:zip])
```

Out[14]: 814 rows × 5 columns

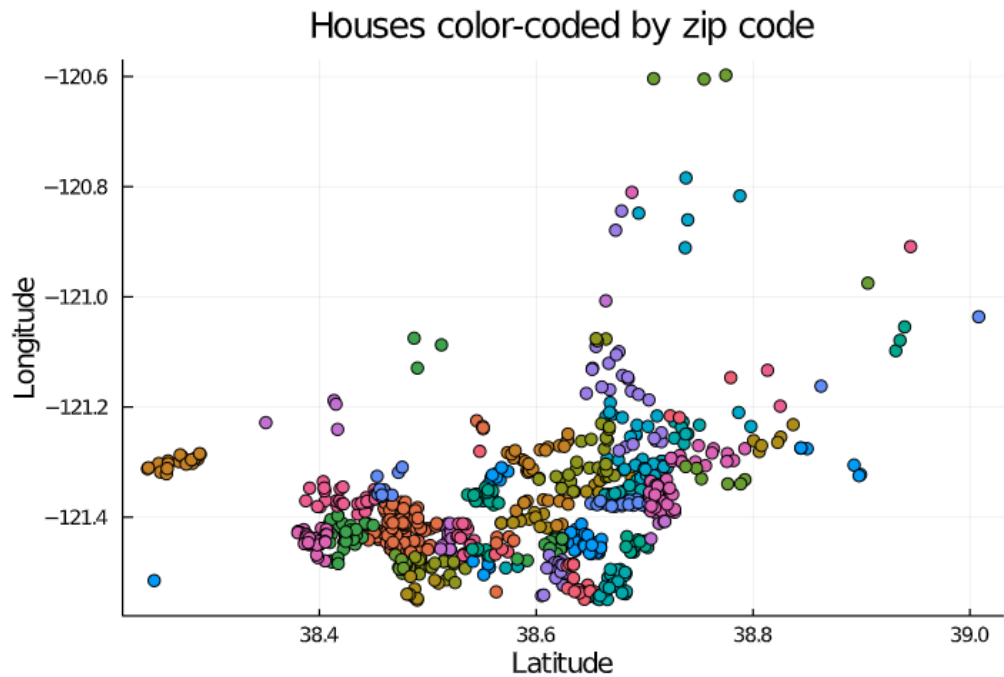
	cluster	city	latitude	longitude	zip
	Int64	String	Float64	Float64	Int64
1	35	SACRAMENTO	38.6319	-121.435	95838
2	47	SACRAMENTO	38.4789	-121.431	95823
3	35	SACRAMENTO	38.6183	-121.444	95815
4	35	SACRAMENTO	38.6168	-121.439	95815
5	10	SACRAMENTO	38.5195	-121.436	95824
6	41	SACRAMENTO	38.6626	-121.328	95841
7	1	SACRAMENTO	38.6817	-121.352	95842
8	46	SACRAMENTO	38.5351	-121.481	95820
9	60	RANCHO CORDOVA	38.6212	-121.271	95670
10	45	RIO LINDA	38.7009	-121.443	95673
11	22	SACRAMENTO	38.6377	-121.452	95838
12	47	SACRAMENTO	38.4707	-121.459	95823
13	35	SACRAMENTO	38.6187	-121.436	95815
14	4	SACRAMENTO	38.4822	-121.493	95822

```
In [15]: 1 #Построим график, обозначив каждый кластер отдельным цветом:
2 clusters_figure = plot(legend = false)
3 for i = 1:k
4     clustered_houses = df[df[:cluster].== i,:]
5     xvals = clustered_houses[:latitude]
6     yvals = clustered_houses[:longitude]
7     scatter!(clusters_figure,xvals,yvals,markersize=4)
8 end
9 xlabel!("Latitude")
10 ylabel!("Longitude")
11 title!("Houses color-coded by cluster")
12 display(clusters_figure)
```

Houses color-coded by cluster




```
In [16]: 1 #Построим график, раскрасив кластеры по почтовому индексу:
2 unique_zips = unique(filter_houses[:zip])
3 zips_figure = plot(legend = false)
4 for uzip in unique_zips
5     subs = filter_houses[filter_houses[:zip].==uzip,:]
6     x = subs[:latitude]
7     y = subs[:longitude]
8     scatter!(zips_figure,x,y)
9 end
10 xlabel!("Latitude")
11 ylabel!("Longitude")
12 title!("Houses color-coded by zip code")
13 display(zips_figure)
```



7.2.2.2. Кластеризация данных. Метод k ближайших соседей

7.2.2.2. Кластеризация данных. Метод k ближайших соседей

```
In [17]: 1 # Подключение пакета NearestNeighbors:
2 #import Pkg
3 #Pkg.add("NearestNeighbors")
4 using NearestNeighbors
```

```
In [18]: 1 #Найдём k-среднее одного из объектов недвижимости:
2 knearest = 10
3 id = 70
4 point = X[:,id]
```

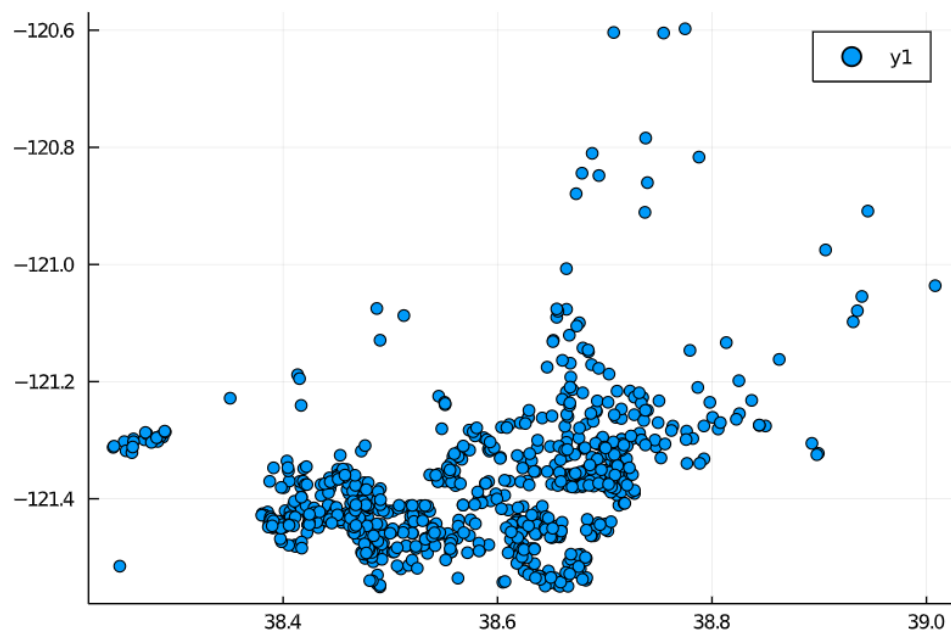
```
Out[18]: 2-element Array{Float64,1}:
 38.44004
-121.421012
```

```
In [19]: 1 # Поиск ближайших соседей:
2 kdtree = KDTree(X)
3 idxs, dists = knn(kdtree, point, knearest, true)
```

```
Out[19]: ([70, 764, 196, 125, 557, 368, 415, 92, 112, 683], [0.0, 0.006264891539364138, 0.00825320259050462, 0.008473585132630057, 0.009164073548370188, 0.009405065124697706, 0.009921759722950759, 0.009941028618812013, 0.010332637707777167, 0.011168993911721985])
```

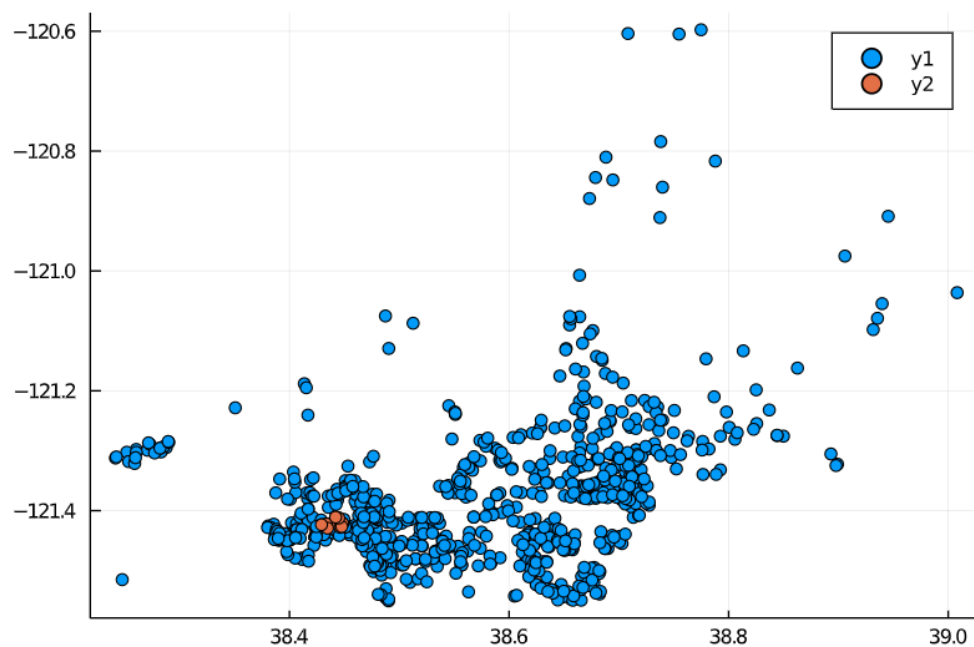
```
In [20]: 1 # Все объекты недвижимости:
2 x = filter_houses[:, :latitude];
3 y = filter_houses[:, :longitude];
4 scatter(x,y)
```

Out[20]:



```
In [21]: 1 # Соседи:
2 x = filter_houses[idxs, :latitude];
3 y = filter_houses[idxs, :longitude];
4 scatter!(x,y)
```

Out[21]:



```
In [22]: 1 # Фильтрация по районам соседних домов:
         2 cities = filter_houses[idxs,:city]
```

```
Out[22]: 10-element Array{String,1}:
"SACRAMENTO"
"ELK GROVE"
"SACRAMENTO"
"SACRAMENTO"
"SACRAMENTO"
"SACRAMENTO"
"ELK GROVE"
"ELK GROVE"
"ELK GROVE"
"ELK GROVE"
```

7.2.2.3. Обработка данных. Метод главных компонент

7.2.2.3. Обработка данных. Метод главных компонент

```
In [23]: 1 # Фрейм с указанием площади и цены недвижимости:
         2 F = filter_houses[:, :sq__ft, :price]
```

```
Out[23]: 814 rows × 2 columns
```

	sq__ft	price
	Int64	Int64
1	836	59222
2	1167	68212
3	796	68880
4	852	69307
5	797	81900
6	1122	89921
7	1104	90895
8	1177	91002
9	941	94905
10	1146	98937
11	909	100309
12	1289	106250
13	871	106852

```
In [24]: 1 # Конвертация данных в массив:
        2 F = convert(Array{Float64,2},F)'
```

```
Out[24]: 2x814 LinearAlgebra.Adjoint{Float64,Array{Float64,2}}:
          836.0  1167.0   796.0   852.0   797.0  ...  1216.0   1685.0   1362.0
        59222.0  68212.0  68880.0  69307.0  81900.0    235000.0  235301.0  235738.0
```

```
In [25]: 1 # Подключение пакета MultivariateStats:
        2 #import Pkg
        3 #Pkg.add("MultivariateStats")
        4 using MultivariateStats
```

```
In [26]: 1 # Приведение типов данных к распределению для PCA:
        2 M = fit(PCA, F)
```

```
Out[26]: PCA(indim = 2, outdim = 1, principalratio = 0.9999840784692097)
```

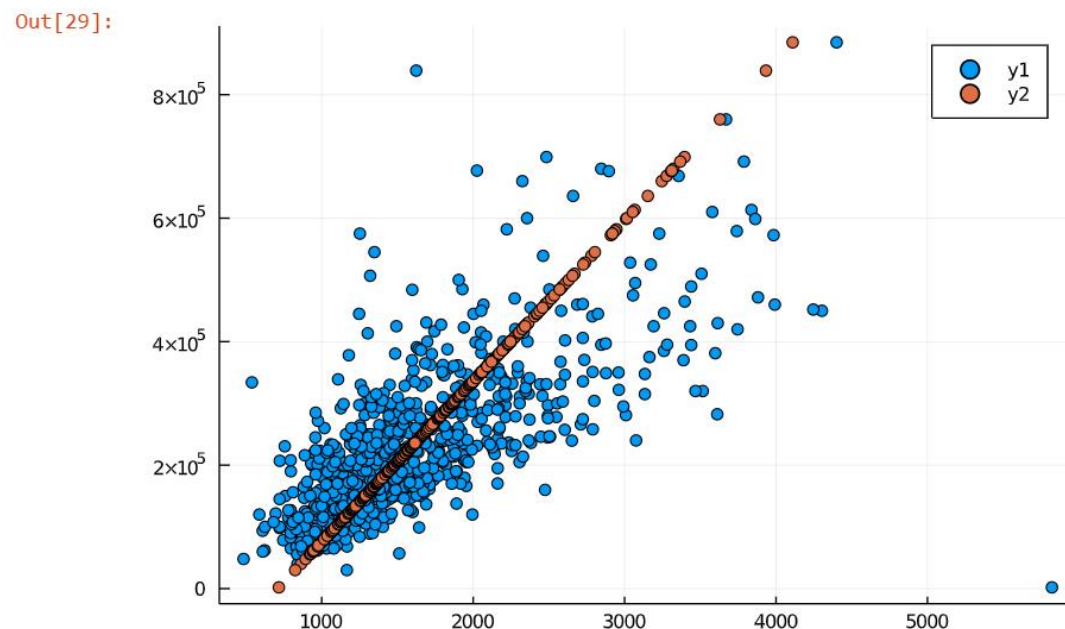
```
In [27]: 1 y = MultivariateStats.transform(M, F)
```

```
Out[27]: 1x814 Array{Float64,2}:
        -170228.0  -1.61237e5  -1.6057e5  ...  4551.16  5550.15  5852.95  6288.7
```

```
In [28]: 1 # Выделение значений главных компонент в отдельную переменную:
        2 Xr = reconstruct(M, y)
```

```
Out[28]: 2x814 Array{Float64,2}:
          936.922   971.477   974.039   975.681  ...  1613.64   1615.32
        59221.6   68212.8   68879.3   69306.5    2.35301e5  235737.0
```

```
In [29]: 1 # Построение графика с выделением главных компонент:
        2 scatter(F[1,:],F[2,:])
        3 scatter!(Xr[1,:],Xr[2,:])
```

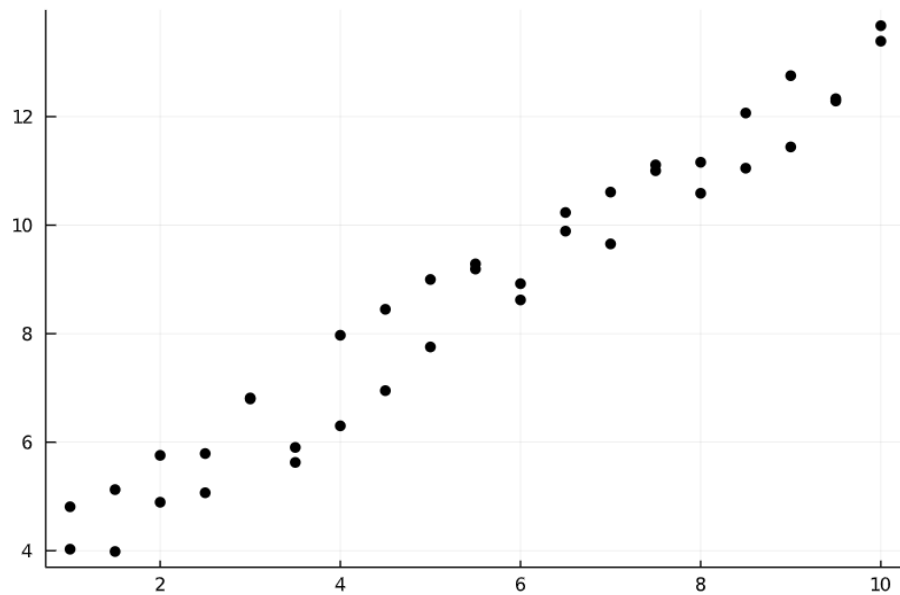


7.2.2.4. Обработка данных. Линейная регрессия

7.2.2.4. Обработка данных. Линейная регрессия

```
In [30]: 1 #Зададим случайный набор данных. Попробуем найти для данных лучшее соответствие:
2 xvals = repeat(1:0.5:10,inner=2)
3 yvals = 3 .+ xvals + 2*rand(length(xvals)) .- 1
4 scatter(xvals,yvals,color=:black,leg=false)
```

Out[30]:



```
In [31]: 1 #Определим функцию линейной регрессии:
2 function find_best_fit(xvals,yvals)
3     meanx = mean(xvals)
4     meany = mean(yvals)
5     stdx = std(xvals)
6     stdy = std(yvals)
7     r = cor(xvals,yvals)
8     a = r*stdy/stdx
9     b = meany - a*meanx
10    return a,b
11 end
```

Out[31]: find_best_fit (generic function with 1 method)

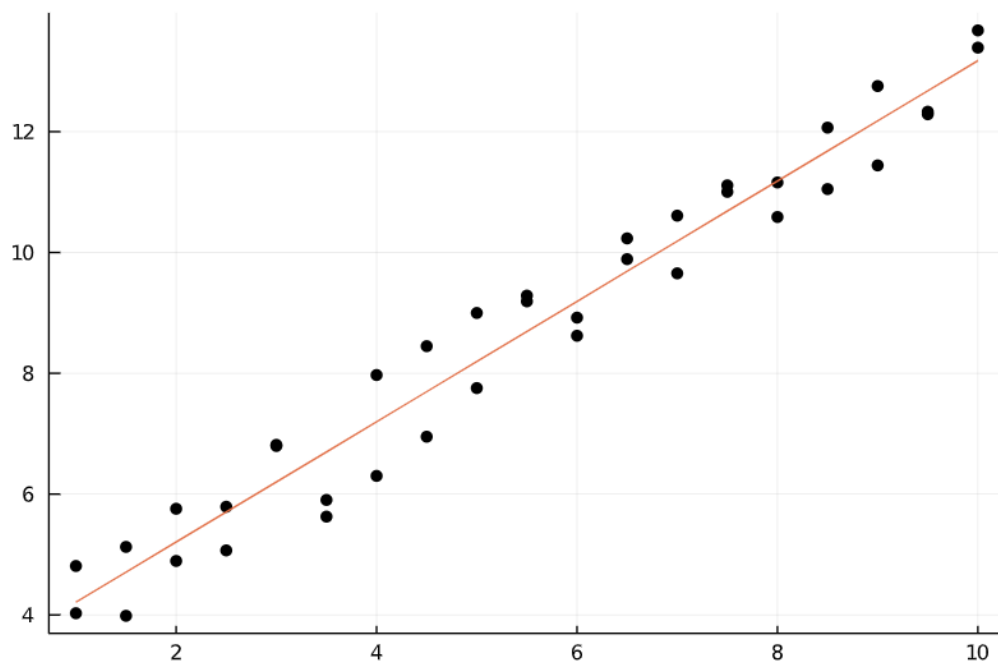
```
In [32]: 1 #Применим функцию линейной регрессии для построения соответствующего графика значений:
2 a,b = find_best_fit(xvals,yvals)
3 ynew = a * xvals .+ b
```

Out[32]: 38-element Array{Float64,1}:

- 4.210197911424078
- 4.210197911424078
- 4.708165370535329
- 4.708165370535329
- 5.206132829646579
- 5.206132829646579
- 5.7041002887578305
- 5.7041002887578305
- 6.20206774786908
- 6.20206774786908
- 6.700035206980331
- 6.700035206980331
- 7.198002666091581

```
In [33]: 1 plot!(xvals,ynew)
```

Out[33]:



```
In [34]: 1 #Сгенерируем большой набор данных:
2 xvals = 1:100000;
3 xvals = repeat(xvals,inner=3);
4 yvals = 3 .* xvals + 2*rand(length(xvals)) .- 1;
```

```
In [35]: 1 @show size(xvals)
2 @show size(yvals)

size(xvals) = (300000,)
size(yvals) = (300000,)
```

Out[35]: (300000,)

```
In [36]: 1 #Определим, сколько времени потребуется, чтобы найти соответствие этим данным:
2 @time a,b = find_best_fit(xvals,yvals)

0.132372 seconds (323.96 k allocations: 16.933 MiB)
```

Out[36]: (1.0000000566451486, 2.996018700163404)

```
In [37]: 1 #Для сравнения реализуем подобный код на языке Python:
2 #import Pkg
3 #Pkg.add("PyCall")
4 #Pkg.add("Conda")
5 using PyCall
6 using Conda
```

```
In [39]: 1 find_best_fit_python = py"find_best_fit_python"
```

```
Out[39]: PyObject <function find_best_fit_python at 0x0000000061344A60>
```

```
In [40]: 1 xpy = PyObject(xvals)
2 ypy = PyObject(yvals)
3 @time a,b = find_best_fit_python(xpy,ypy)
```

```
0.096835 seconds (195.44 k allocations: 10.155 MiB)
```

```
Out[40]: (1.00000000566451517, 2.996018700010609)
```

```
In [41]: 1 #Используем пакет для анализа производительности, чтобы провести сравнение:
2 #import Pkg
3 #Pkg.add("BenchmarkTools")
4 using BenchmarkTools
```

```
In [42]: 1 @btime a,b = find_best_fit_python(xvals,yvals)
```

```
8.559 ms (27 allocations: 1.02 KiB)
```

```
Out[42]: (1.00000000566451517, 2.996018700010609)
```

```
In [43]: 1 @btime a,b = find_best_fit(xvals,yvals)
```

```
1.175 ms (1 allocation: 32 bytes)
```

```
Out[43]: (1.00000000566451486, 2.996018700163404)
```
