

Повторение примеров

Julia для науки о данных

Считывание данных

```
In [1]: using CSV, DataFrames, DelimitedFiles

# Считывание данных и их запись в структуру:
P = CSV.File("programminglanguages.csv") |> DataFrame
# Функция определения по названию языка программирования года его создания:
function language_created_year(P,language::String)
    loc = findfirst(P[:,2].==language)
    return P[loc,1]
end
# Пример вызова функции и определение даты создания языка Python:
language_created_year(P,"Python")
# Пример вызова функции и определение даты создания языка Julia:
language_created_year(P,"Julia")
```

Out[1]: 2012

```
In [2]: language_created_year(P,"julia")
```

```
MethodError: no method matching getindex(::DataFrame, ::Nothing, ::Int64)

Closest candidates are:
  getindex(::DataFrame, ::typeof(!), ::Union{Signed, Unsigned})
    @ DataFrames C:\Users\vemanaeva\.julia\packages\DataFrames\58MUJ\src\dataframe\dataframe.jl:548
  getindex(::DataFrame, ::Colon, ::Union{AbstractString, Signed, Symbol, Unsigned})
    @ DataFrames C:\Users\vemanaeva\.julia\packages\DataFrames\58MUJ\src\dataframe\dataframe.jl:542
  getindex(::DataFrame, ::InvertedIndex, ::Union{AbstractString, Signed, Symbol, Unsigned})
    @ DataFrames C:\Users\vemanaeva\.julia\packages\DataFrames\58MUJ\src\dataframe\dataframe.jl:538
  ...
  ...

Stacktrace:
 [1] language_created_year(P::DataFrame, language::String)
   @ Main .\In[1]:8
 [2] top-level scope
   @ In[2]:1
```

```
In [3]: # Функция определения по названию языка программирования года его создания (без учё
function language_created_year_v2(P,language::String)
loc = findfirst(lowercase.(P[:,2]).==lowercase.(language))
return P[loc,1]
```

```
end  
# Пример вызова функции и определение даты создания языка julia:  
language_created_year_v2(P,"julia")
```

Out[3]: 2012

```
In [4]: # Построчное считывание данных с указанием разделителя:  
Tx = readdlm("programminglanguages.csv", ',', )
```

```
Out[4]: 74x2 Matrix{Any}:  
    "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"  
    :  
2003      "Scala"  
2005      "F#"  
2006      "PowerShell"  
2007      "Clojure"  
2009      "Go"  
2010      "Rust"  
2011      "Dart"  
2011      "Kotlin"  
2011      "Red"  
2011      "Elixir"  
2012      "Julia"  
2014      "Swift"
```

Запись данных в файл

```
In [5]: # Запись данных в CSV-файл:  
CSV.write("programming_languages_data2.csv", P)
```

Out[5]: "programming_languages_data2.csv"

```
In [6]: # Пример записи данных в текстовый файл с разделителем ',':  
writedlm("programming_languages_data.txt", Tx, ',')
```

```
In [7]: # Пример записи данных в текстовый файл с разделителем '-':  
writedlm("programming_languages_data2.txt", Tx, '-')
```

```
In [8]: # Построчное считывание данных с указанием разделителя:  
P_new_delim = readdlm("programming_languages_data2.txt", '-')
```

```
Out[8]: 74x2 Matrix{Any}:
    "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"
    :
    2003      "Scala"
    2005      "F#"
    2006      "PowerShell"
    2007      "Clojure"
    2009      "Go"
    2010      "Rust"
    2011      "Dart"
    2011      "Kotlin"
    2011      "Red"
    2011      "Elixir"
    2012      "Julia"
    2014      "Swift"
```

Словари

```
In [9]: # Инициализация словаря:
dict = Dict{Integer,Vector{String}}()
```

```
Out[9]: Dict{Integer, Vector{String}}()
```

```
In [10]: # Инициализация словаря:
dict2 = Dict()
```

```
Out[10]: Dict{Any, Any}()
```

```
In [11]: # Заполнение словаря данными:
for i = 1:size(P,1)
    year,lang = P[i,:]
    if year in keys(dict)
        dict[year] = push!(dict[year],lang)
    else
        dict[year] = [lang]
    end
end
dict
```

```
Out[11]: Dict{Integer, Vector{String}} with 45 entries:  
1985 => ["Eiffel"]  
2002 => ["Scratch"]  
1952 => ["Autocode"]  
1963 => ["CPL"]  
1964 => ["Speakeasy", "BASIC", "PL/I"]  
1967 => ["BCPL"]  
2001 => ["C#", "D"]  
1991 => ["Python", "Visual Basic"]  
1957 => ["FORTRAN", "COMTRAN"]  
1988 => ["Tcl", "Wolfram Language "]  
1955 => ["FLOW-MATIC"]  
1951 => ["Regional Assembly Language"]  
1994 => ["CLOS "]  
2011 => ["Dart", "Kotlin", "Red", "Elixir"]  
1959 => ["FACT", "COBOL", "RPG"]  
1962 => ["APL", "Simula", "SNOBOL"]  
2005 => ["F#"]  
1969 => ["B"]  
1972 => ["C", "Smalltalk", "Prolog"]  
1997 => ["Rebol"]  
1986 => ["Objective-C", "LabVIEW ", "Erlang"]  
1993 => ["Lua", "R"]  
1958 => ["LISP", "ALGOL 58"]  
1987 => ["Perl"]  
1954 => ["IPL"]  
:    => :
```

```
In [12]: # Пример определения в словаре языков программирования, созданных в 2003 году:  
dict[2003]
```

```
Out[12]: 2-element Vector{String}:  
"Groovy"  
"Scala"
```

DataFrames

```
In [13]: # Задаём переменную со структурой DataFrame:  
df = DataFrame(year = P[:,1], language = P[:,2])  
# Вывод всех значения столбца year:  
df[!,:year]
```

```
Out[13]: 73-element Vector{Int64}:
```

```
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 [14]: # Получение статистических сведений о фрейме:  
describe(df)
```

```
Out[14]: 2×7 DataFrame
```

Row	variable	mean	min	median	max	nmissing	eltype
Symbol	Union... Any	Union... Any	Union... Any	Int64	Data Type		
1	year	1982.99	1951	1986.0	2014	0	Int64
2	language		ALGOL 58		dBase III	0	String31

RDatasets

```
In [15]: # Подгружаем пакет RDatasets:  
using RDatasets  
# Задаём структуру данных в виде набора данных:  
iris = dataset("datasets", "iris")
```

Out[15]: 150×5 DataFrame

125 rows omitted

Row	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
13	4.8	3.0	1.4	0.1	setosa
:	:	:	:	:	:
139	6.0	3.0	4.8	1.8	virginica
140	6.9	3.1	5.4	2.1	virginica
141	6.7	3.1	5.6	2.4	virginica
142	6.9	3.1	5.1	2.3	virginica
143	5.8	2.7	5.1	1.9	virginica
144	6.8	3.2	5.9	2.3	virginica
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

In [16]: # Определения типа переменной:
typeof(iris)

```
Out[16]: DataFrame
```

```
In [17]: describe(iris)
```

```
Out[17]: 5×7 DataFrame
```

Row	variable	mean	min	median	max	nmissing	eltype
	Symbol	Union... Any	Union... Any	Union... Any	Int64		DataType
1	SepalLength	5.84333	4.3	5.8	7.9	0	Float64
2	SepalWidth	3.05733	2.0	3.0	4.4	0	Float64
3	PetalLength	3.758	1.0	4.35	6.9	0	Float64
4	PetalWidth	1.19933	0.1	1.3	2.5	0	Float64
5	Species		setosa		virginica	0	CategoricalValue{String, UInt8}

Работа с переменными отсутствующего типа (Missing Values)

```
In [18]: # Отсутствующий тип:  
a = missing  
typeof(a)
```

```
Out[18]: Missing
```

```
In [19]: # Пример операции с переменной отсутствующего типа:  
a + 1
```

```
Out[19]: missing
```

```
In [20]: # Определение перечня продуктов:  
foods = ["apple", "cucumber", "tomato", "banana"]  
# Определение калорий:  
calories = [missing, 47, 22, 105]
```

```
Out[20]: 4-element Vector{Union{Missing, Int64}}:  
    missing  
    47  
    22  
    105
```

```
In [21]: # Определение типа переменной:  
typeof(calories)
```

```
Out[21]: Vector{Union{Missing, Int64}} (alias for Array{Union{Missing, Int64}, 1})
```

```
In [22]: # Подключаем пакет Statistics:  
using Statistics
```

```
# Определение среднего значения:  
mean(calories)
```

Out[22]: missing

```
In [23]: # Определение среднего значения без значений с отсутствующим типом:  
mean(skipmissing(calories))
```

Out[23]: 58.0

```
In [24]: # Задание сведений о ценах:  
prices = [0.85,1.6,0.8,0.6]  
# Формирование данных о калориях:  
dataframe_calories = DataFrame(item=foods,calories=calories)  
# Формирование данных о ценах:  
dataframe_prices = DataFrame(item=foods,price=prices)  
# Объединение данных о калориях и ценах:  
DF = innerjoin(dataframe_calories,dataframe_prices,on=:item)
```

Out[24]: 4×3 DataFrame

Row	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

FileIO

```
In [25]: # Подключаем пакет FileIO:  
using FileIO  
using ImageIO  
# Загрузка изображения:  
X1 = load("julialogo.png")
```

In [26]: # Определение типа и размера данных:

```
@show typeof(X1);  
@show size(X1);
```

```
typeof(X1) = Matrix{ColorTypes.RGBA{FixedPointNumbers.NOf8}}
size(X1) = (250, 400)
```

Обработка данных: стандартные алгоритмы машинного обучения в Julia

Кластеризация данных. Метод k-средних

```
In [27]: using Plots  
# Загрузка данных:  
houses = CSV.File("houses.csv") |> DataFrame
```

Out[27]: 985×12 DataFrame

960 rows omitted

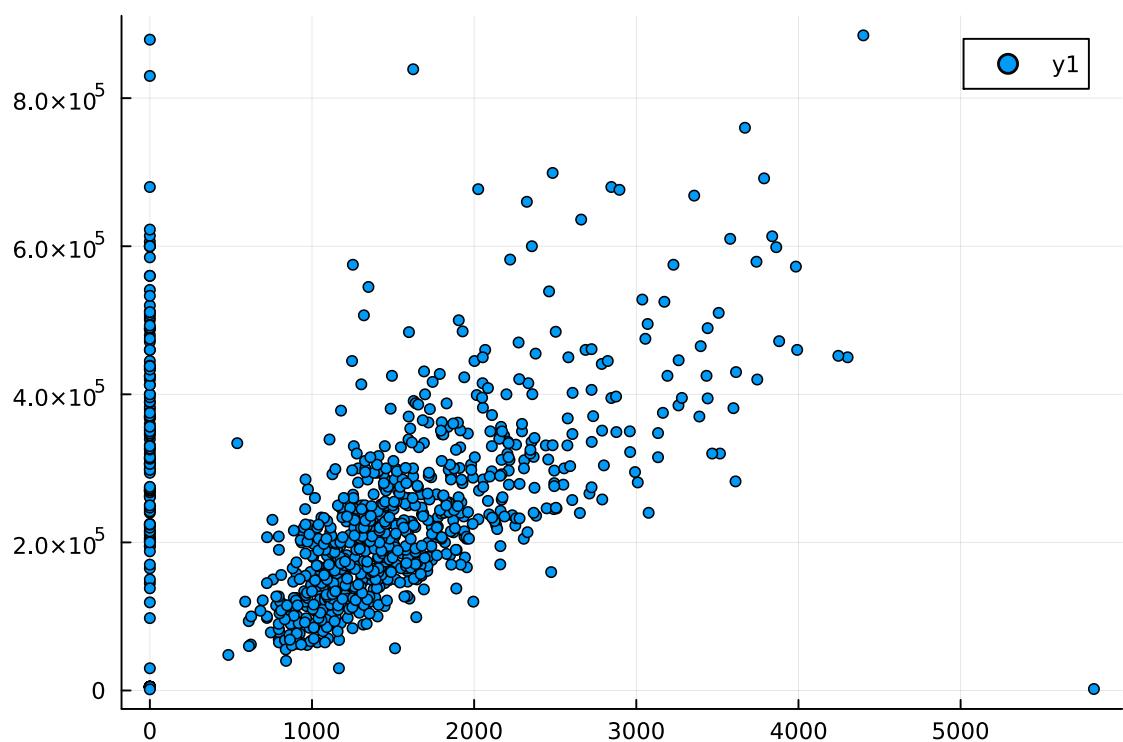
Row	street	city	zip	state	beds	baths	sq_ft	type	sal
	String	String15	Int64	String3	Int64	Int64	Int64	String15	Str
1	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836	Residential	We 21 00: ED
2	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167	Residential	We 21 00: ED
3	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796	Residential	We 21 00: ED
4	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	We 21 00: ED
5	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797	Residential	We 21 00: ED
6	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122	Condo	We 21 00: ED
7	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104	Residential	We 21 00: ED
8	2561 19TH AVE	SACRAMENTO	95820	CA	3	1	1177	Residential	We 21 00: ED
9	11150 TRINITY RIVER DR Unit 114	RANCHO CORDOVA	95670	CA	2	2	941	Condo	We 21 00: ED
10	7325 10TH ST	RIO LINDA	95673	CA	3	2	1146	Residential	We 21 00: ED
11	645 MORRISON AVE	SACRAMENTO	95838	CA	3	2	909	Residential	We 21

Row	street	city	zip	state	beds	baths	sq_ft	type	sal
	String	String15	Int64	String3	Int64	Int64	Int64	String15	Str
									00: ED
12	4085 FAWN CIR	SACRAMENTO	95823	CA	3	2	1289	Residential	We 21 00: ED
13	2930 LA ROSA RD	SACRAMENTO	95815	CA	1	1	871	Residential	We 21 00: ED
	:	:	:	:	:	:	:	:	:
974	2181 WINTERHAVEN CIR	CAMERON PARK	95682	CA	3	2	0	Residential	Thi 15 00: ED
975	7540 HICKORY AVE	ORANGEVALE	95662	CA	3	1	1456	Residential	Thi 15 00: ED
976	5024 CHAMBERLIN CIR	ELK GROVE	95757	CA	3	2	1450	Residential	Thi 15 00: ED
977	2400 INVERNESS DR	LINCOLN	95648	CA	3	2	1358	Residential	Thi 15 00: ED
978	5 BISHOPGATE CT	SACRAMENTO	95823	CA	4	2	1329	Residential	Thi 15 00: ED
979	5601 REXLEIGH DR	SACRAMENTO	95823	CA	4	2	1715	Residential	Thi 15 00: ED
980	1909 YARNELL WAY	ELK GROVE	95758	CA	3	2	1262	Residential	Thi 15 00: ED
981	9169 GARLINGTON CT	SACRAMENTO	95829	CA	4	3	2280	Residential	Thi 15 00: ED

Row	street	city	zip	state	beds	baths	sq_ft	type	sal
	String	String15	Int64	String3	Int64	Int64	Int64	String15	Str
982	6932 RUSKUT WAY	SACRAMENTO	95823	CA	3	2	1477	Residential	Thi 15 00: ED
983	7933 DAFFODIL WAY	CITRUS HEIGHTS	95610	CA	3	2	1216	Residential	Thi 15 00: ED
984	8304 RED FOX WAY	ELK GROVE	95758	CA	4	2	1685	Residential	Thi 15 00: ED
985	3882 YELLOWSTONE LN	EL DORADO HILLS	95762	CA	3	2	1362	Residential	Thi 15 00: ED

```
In [28]: plot(size=(500,500),leg=False)
x = houses[!,:sq_ft]
y = houses[!,:price]
scatter(x,y,markersize=3)
```

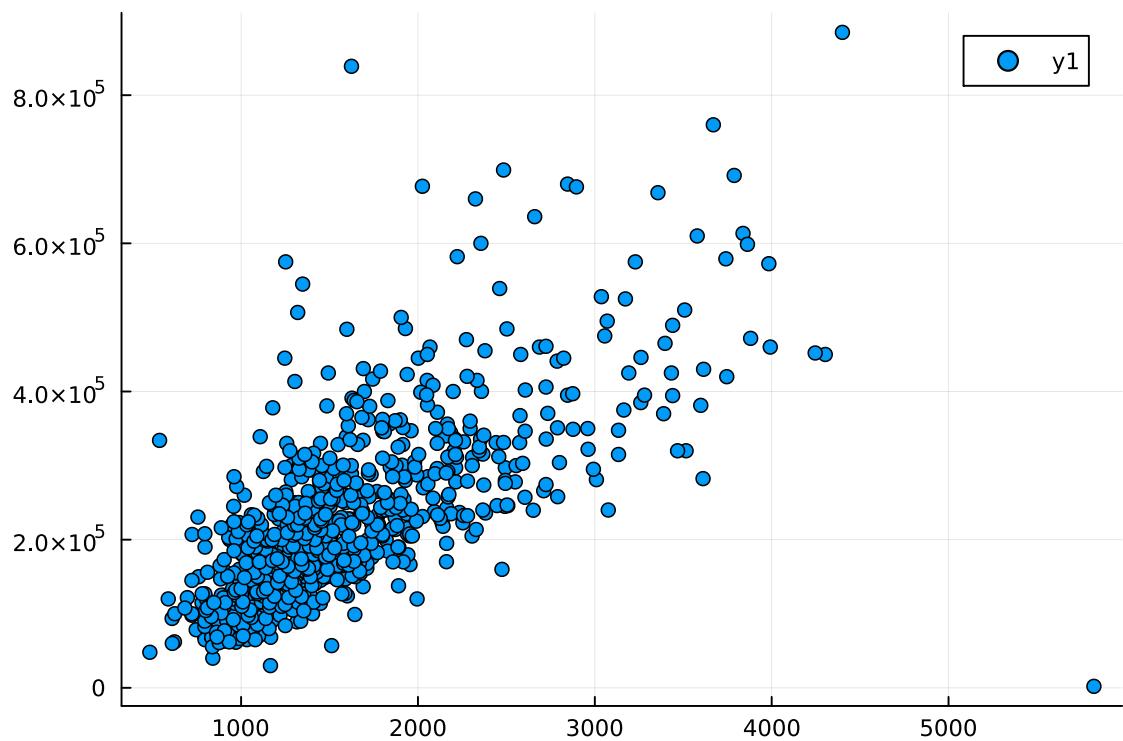
Out[28]:



```
In [29]: # Фильтрация данных по заданному условию:
filter_houses = houses[houses[!,:sq_ft]>0,:]
# Построение графика:
```

```
x = filter_houses[!, :sq_ft]
y = filter_houses[!, :price]
scatter(x, y)
```

Out[29]:



In [30]:

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

ArgumentError: by function was removed from DataFrames.jl. Use the `combine(groupby(...), ...)` or `combine(f, groupby(...))` instead.

Stacktrace:

```
[1] by(::DataFrame, ::Vararg{Any}; kwargs::Base.Pairs{Symbol, Union{}, Tuple{}, NamedTuple{(), Tuple{}}})
    @ DataFrames C:\Users\vemanaeva\.julia\packages\DataFrames\58MUJ\src\deprecated.jl:4
[2] by(::DataFrame, ::Vararg{Any})
    @ DataFrames C:\Users\vemanaeva\.julia\packages\DataFrames\58MUJ\src\deprecated.jl:4
[3] top-level scope
    @ In[30]:2
```

In [31]:

```
combine(filter_houses->mean(filter_houses[!, :price]), groupby(filter_houses, :type))
```

Out[31]: 3×2 DataFrame

Row	type	x1
	String15	Float64
1	Residential	2.34802e5
2	Condo	1.34213e5
3	Multi-Family	2.24535e5

In [32]:

```
using Clustering  
# Добавление данных :latitude и :longitude в новый фрейм:  
X = filter_houses[:, [:latitude,:longitude]]
```

Out[32]: 814×2 DataFrame

789 rows omitted

Row	latitude	longitude
	Float64	Float64
1	38.6319	-121.435
2	38.4789	-121.431
3	38.6183	-121.444
4	38.6168	-121.439
5	38.5195	-121.436
6	38.6626	-121.328
7	38.6817	-121.352
8	38.5351	-121.481
9	38.6212	-121.271
10	38.7009	-121.443
11	38.6377	-121.452
12	38.4707	-121.459
13	38.6187	-121.436
:	:	:
803	38.7035	-121.375
804	38.7031	-121.235
805	38.3898	-121.446
806	38.8978	-121.325
807	38.4679	-121.445
808	38.4453	-121.442
809	38.4174	-121.484
810	38.4577	-121.36
811	38.4999	-121.459
812	38.7088	-121.257
813	38.417	-121.397
814	38.6552	-121.076

In [33]: # Конвертация данных в матричный вид:
X = Matrix(X[:, 1:2])

```
Out[33]: 814x2 Matrix{Float64}:
```

```
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 [34]: # Транспонирование матрицы с данными:
```

```
X = X'  
# Задание количества кластеров:  
k = length(unique(filter_houses[!, :zip]))  
# Определение k-среднего:  
C = kmeans(X, k)
```

```
Out[34]: KmeansResult{Matrix{Float64}, Float64, Int64}([38.47975119354839 38.737452 ... 38.63  
562266666667 38.60819775; -121.41092403225811 -120.910963 ... -121.33890555555556 -1  
21.371695125], [41, 1, 41, 41, 26, 30, 19, 36, 58, 27 ... 39, 20, 24, 17, 13, 35,  
24, 4, 29, 62], [0.00012031390360789374, 0.0004048906521347817, 0.0002828492943081  
0735, 0.0003708190743054729, 0.00015539828018518165, 0.00014710579853272066, 0.000  
27759934528148733, 0.00026751534460345283, 1.144897032645531e-5, 0.000340708586008  
98646 ... 5.784670793218538e-5, 5.241325561655685e-5, 0.0002126721228705719, 0.000  
16335299733327702, 0.0007778412618790753, 7.293177623068914e-5, 0.0004220604350848  
589, 0.00032489683144376613, 0.000387906427931739, 0.00011059417010983452], [31,  
1, 36, 9, 4, 18, 15, 8, 5, 9 ... 4, 9, 12, 4, 19, 5, 8, 15, 9, 8], [31, 1, 36, 9,  
4, 18, 15, 8, 5, 9 ... 4, 9, 12, 4, 19, 5, 8, 15, 9, 8], 0.1989128205095767, 11, t  
rue)
```

```
In [35]: # Формирование фрейма данных:
```

```
df = DataFrame(cluster = C.assignments, city = filter_houses[!, :city], latitude = fi
```

Out[35]: 814×5 DataFrame

789 rows omitted

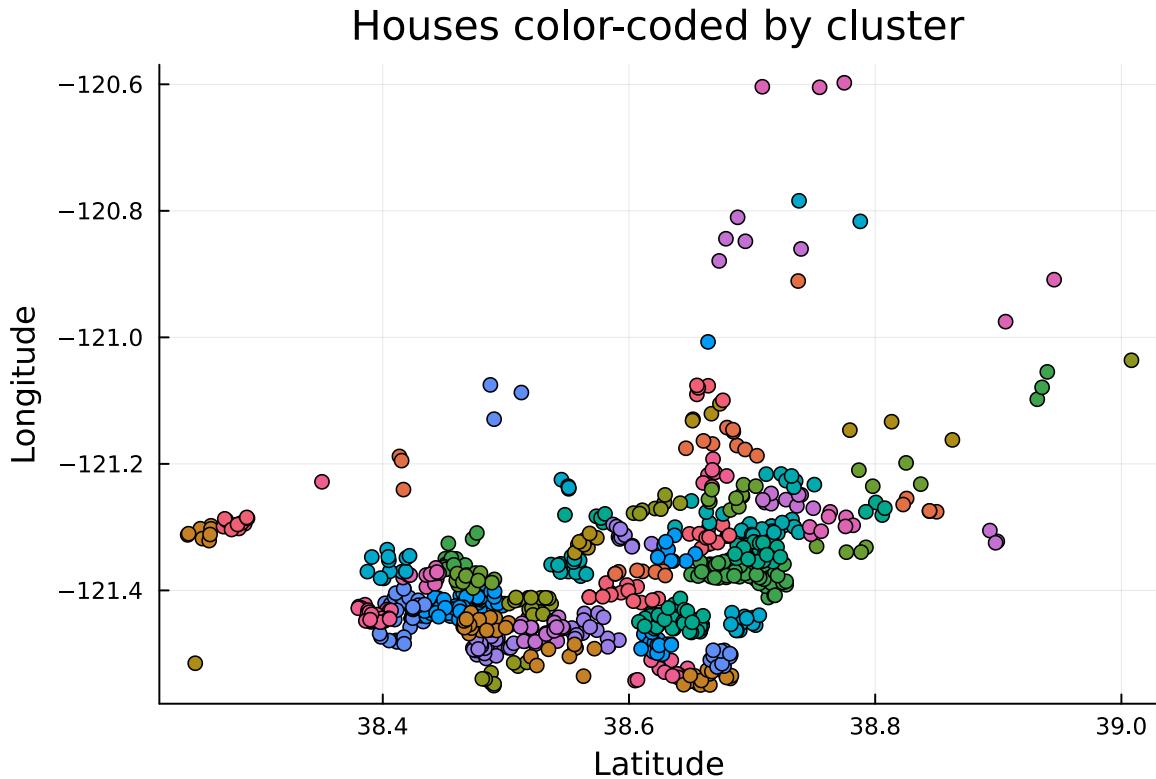
Row	cluster	city	latitude	longitude	zip
	Int64	String15	Float64	Float64	Int64
1	41	SACRAMENTO	38.6319	-121.435	95838
2	1	SACRAMENTO	38.4789	-121.431	95823
3	41	SACRAMENTO	38.6183	-121.444	95815
4	41	SACRAMENTO	38.6168	-121.439	95815
5	26	SACRAMENTO	38.5195	-121.436	95824
6	30	SACRAMENTO	38.6626	-121.328	95841
7	19	SACRAMENTO	38.6817	-121.352	95842
8	36	SACRAMENTO	38.5351	-121.481	95820
9	58	RANCHO CORDOVA	38.6212	-121.271	95670
10	27	RIO LINDA	38.7009	-121.443	95673
11	41	SACRAMENTO	38.6377	-121.452	95838
12	24	SACRAMENTO	38.4707	-121.459	95823
13	41	SACRAMENTO	38.6187	-121.436	95815
:	:	:	:	:	:
803	3	NORTH HIGHLANDS	38.7035	-121.375	95660
804	48	ORANGEVALE	38.7031	-121.235	95662
805	39	ELK GROVE	38.3898	-121.446	95757
806	20	LINCOLN	38.8978	-121.325	95648
807	24	SACRAMENTO	38.4679	-121.445	95823
808	17	SACRAMENTO	38.4453	-121.442	95823
809	13	ELK GROVE	38.4174	-121.484	95758
810	35	SACRAMENTO	38.4577	-121.36	95829
811	24	SACRAMENTO	38.4999	-121.459	95823
812	4	CITRUS HEIGHTS	38.7088	-121.257	95610
813	29	ELK GROVE	38.417	-121.397	95758
814	62	EL DORADO HILLS	38.6552	-121.076	95762

```
In [36]: clusters_figure = plot(legend = False)
for i = 1:k
    clustered_houses = df[df[:, cluster] == i, :]
```

```

xvals = clustered_houses[!, :latitude]
yvals = clustered_houses[!, :longitude]
scatter!(clusters_figure, xvals, yvals, markersize=4)
end
xlabel!("Latitude")
ylabel!("Longitude")
title!("Houses color-coded by cluster")
display(clusters_figure)

```

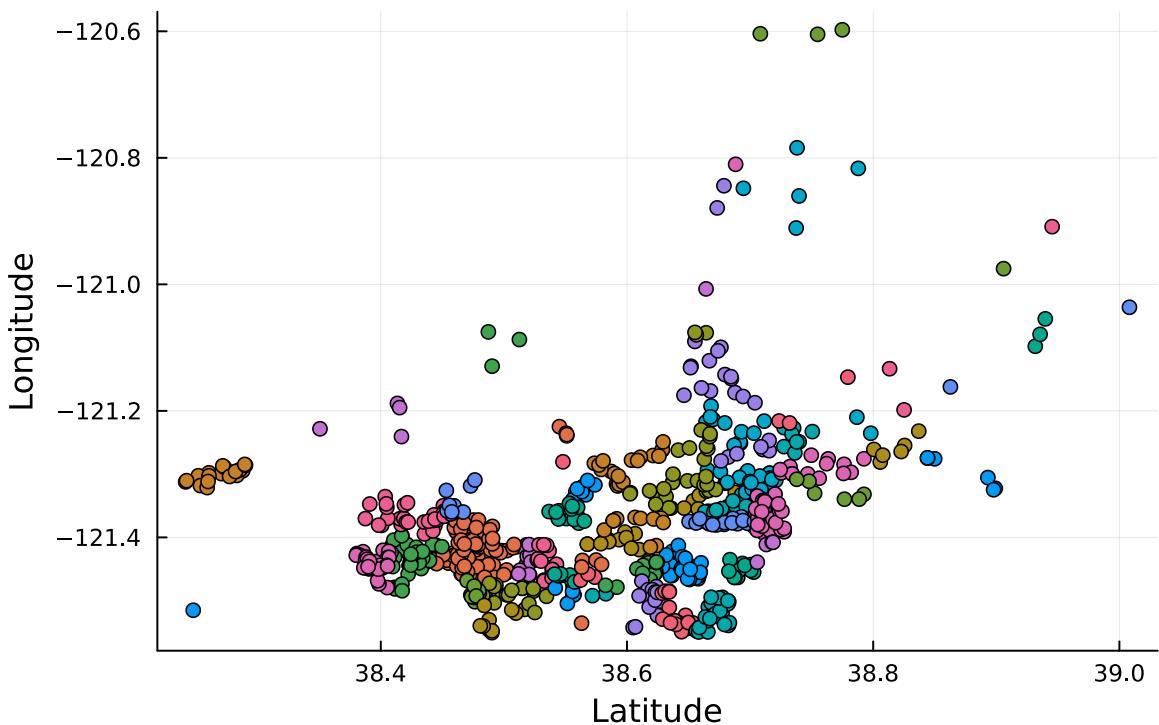


```

In [37]: unique_zips = unique(filter_houses[!, :zip])
zips_figure = plot(legend = false)
for uzip in unique_zips
    subs = filter_houses[filter_houses[!, :zip].==uzip,:]
    x = subs[!, :latitude]
    y = subs[!, :longitude]
    scatter!(zips_figure, x, y)
end
xlabel!("Latitude")
ylabel!("Longitude")
title!("Houses color-coded by zip code")
display(zips_figure)

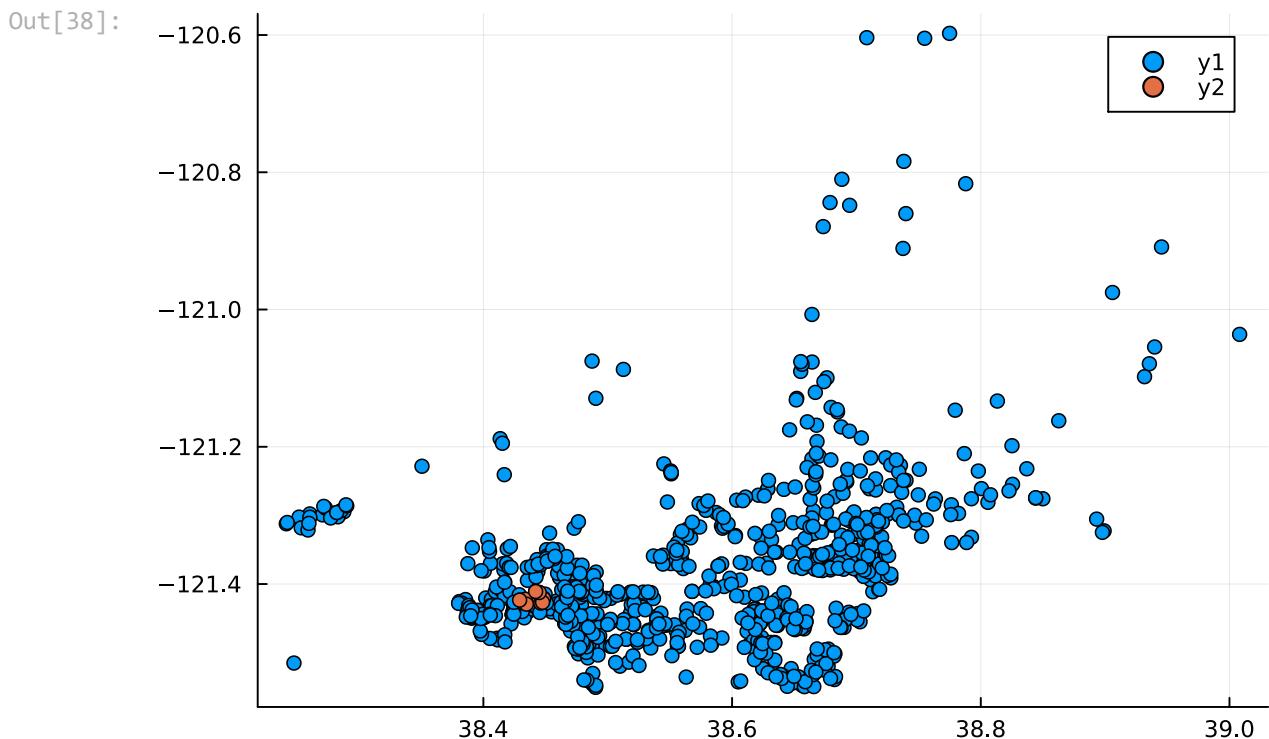
```

Houses color-coded by zip code



Кластеризация данных. Метод k ближайших соседей

```
In [38]: using NearestNeighbors  
knearest = 10  
id = 70  
point = X[:,id]  
# Поиск ближайших соседей:  
kdtree = KDTree(X)  
idxs, dists = knn(kdtree, point, knearest, true)  
# Все объекты недвижимости:  
x = filter_houses[!,:latitude];  
y = filter_houses[!,:longitude];  
scatter(x,y)  
# Соседи:  
x = filter_houses[idxs,:latitude];  
y = filter_houses[idxs,:longitude];  
scatter!(x,y)
```



In [39]:

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

Out[39]:

```
10-element PooledArrays.PooledVector{String15, UInt32, Vector{UInt32}}:
 "SACRAMENTO"
 "ELK GROVE"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "ELK GROVE"
 "ELK GROVE"
 "ELK GROVE"
 "ELK GROVE"
```

Обработка данных. Метод главных компонент

In [40]:

```
# Фрейм с указанием площади и цены недвижимости:
F = filter_houses[!,:sq_ft,:price]
# Конвертация данных в массив:
F = Array{Float64,2}(F)'
# Подключение пакета MultivariateStats:
using MultivariateStats
# Приведение типов данных к распределению для PCA:
M = fit(PCA, F)
# Выделение значений главных компонент в отдельную переменную:
Xr = reconstruct(M, y)
# Построение графика с выделением главных компонент:
scatter(F[1,:],F[2,:])
scatter!(Xr[1,:],Xr[2,:])
```

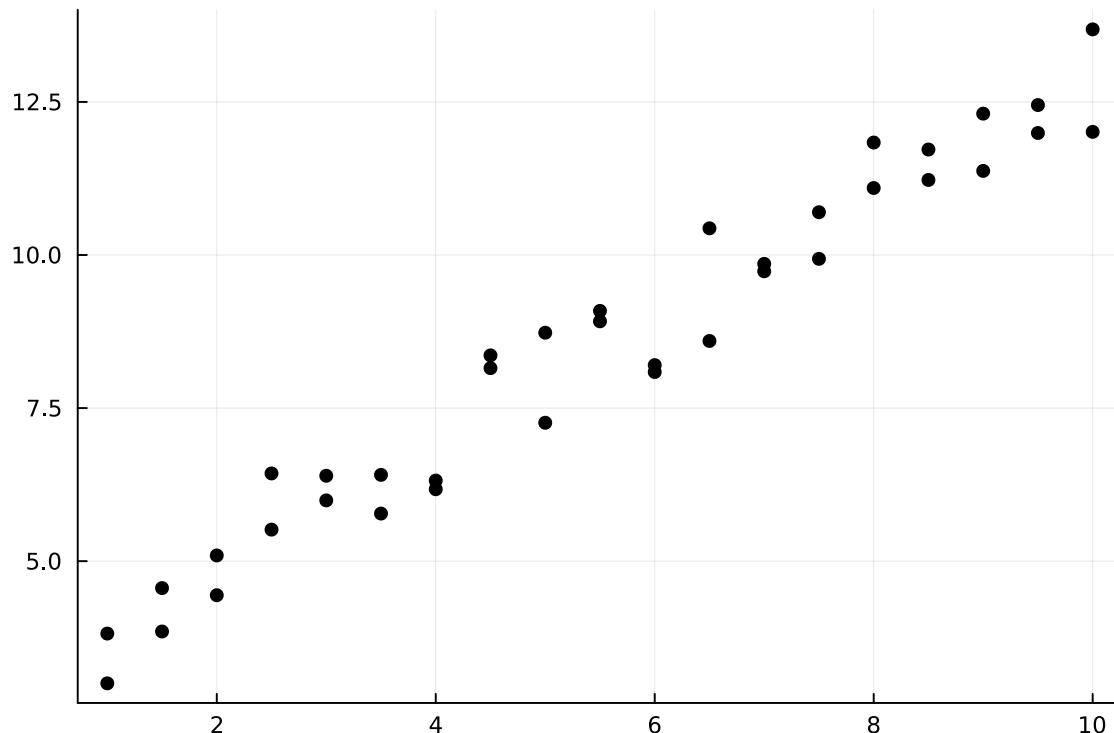
```
DimensionMismatch: second dimension of A, 1, does not match length of x, 10

Stacktrace:
 [1] gemv!(y::Vector{Float64}, tA::Char, A::Matrix{Float64}, x::Vector{Float64}, α::Bool, β::Bool)
   @ LinearAlgebra C:\Users\vemanaeva\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\matmul.jl:493
 [2] mul!
   @ C:\Users\vemanaeva\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\matmul.jl:65 [inlined]
 [3] mul!
   @ C:\Users\vemanaeva\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\matmul.jl:276 [inlined]
 [4] *(A::Matrix{Float64}, x::Vector{Float64})
   @ LinearAlgebra C:\Users\vemanaeva\AppData\Local\Programs\Julia-1.9.3\share\julia\stdlib\v1.9\LinearAlgebra\src\matmul.jl:52
 [5] reconstruct(M::PCA{Float64}, y::Vector{Float64})
   @ MultivariateStats C:\Users\vemanaeva\.julia\packages\MultivariateStats\F3eLE\src\pca.jl:135
 [6] top-level scope
 @ In[40]:10
```

Обработка данных. Линейная регрессия

```
In [41]: xvals = repeat(1:0.5:10, inner=2)
yvals = 3 .+ xvals + 2*rand(length(xvals)) .- 1
scatter(xvals,yvals,color=:black,leg=false)
```

Out[41]:



```
In [42]: function find_best_fit(xvals,yvals)
meanx = mean(xvals)
meany = mean(yvals)
```

```

stdx = std(xvals)
stdy = std(yvals)
r = cor(xvals,yvals)
a = r*stdy/stdx
b = meany - a*meanx
return a,b
end

```

Out[42]: find_best_fit (generic function with 1 method)

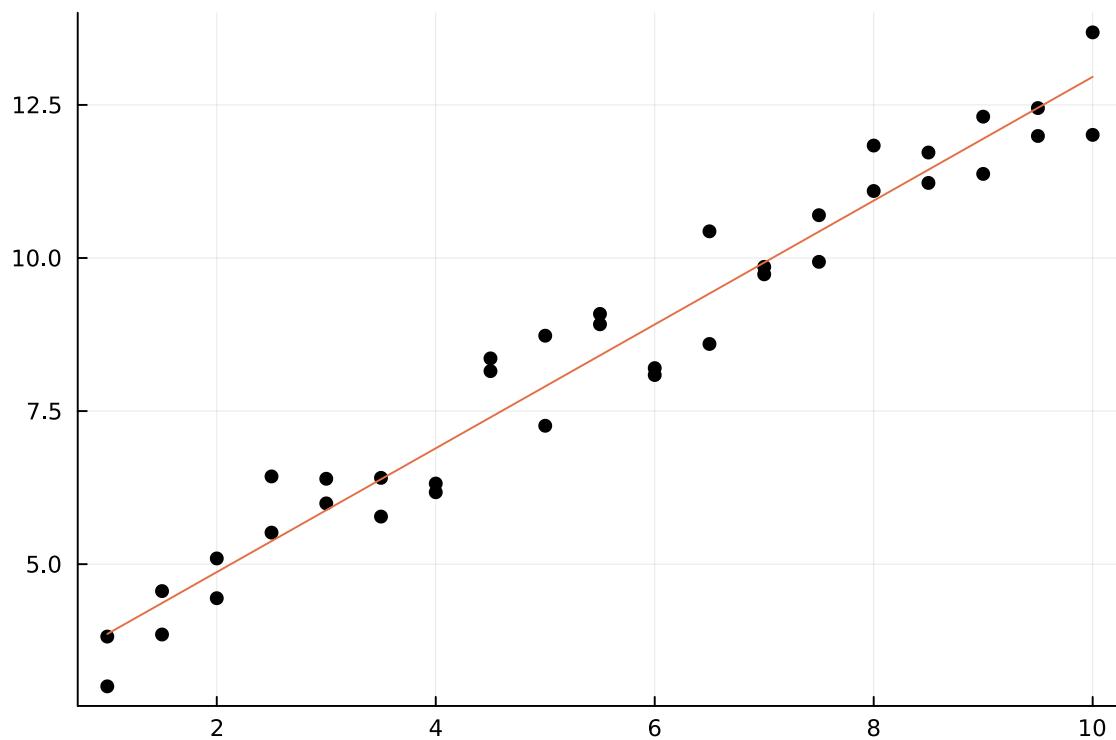
In [43]:

```

a,b = find_best_fit(xvals,yvals)
ynew = a * xvals .+ b
plot!(xvals,ynew)

```

Out[43]:



In [44]:

```

xvals = 1:1000000;
xvals = repeat(xvals,inner=3);
yvals = 3 .+ xvals + 2*rand(length(xvals)) .- 1;
@show size(xvals)
@show size(yvals)
@time a,b = find_best_fit(xvals,yvals)

```

```

size(xvals) = (300000,)
size(yvals) = (300000,)
0.042879 seconds (23.13 k allocations: 1.507 MiB, 96.45% compilation time)

```

Out[44]: (1.000000044268917, 2.998837311293755)

In [45]:

```

using PyCall
using Conda
py"""
import numpy
def find_best_fit_python(xvals,yvals):

```

```

meanx = numpy.mean(xvals)
meany = numpy.mean(yvals)
stdx = numpy.std(xvals)
stdy = numpy.std(yvals)
r = numpy.corrcoef(xvals,yvals)[0][1]
a = r*stdy/stdx
b = meany - a*meanx
return a,b
"""
xpy = PyObject(xvals)
ypy = PyObject(yvals)
@time a,b = find_best_fit_python(xpy,ypy)

```

```

PyError ($(Expr(:escape, :(ccall(#= C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX
\src\pyeval.jl:34 =# @pysym(:Py_CompilerString), PyPtr, (Cstring, Cstring, Cint), s,
fname, input_type)))) <class 'SyntaxError'>
SyntaxError("'return' outside function", ('C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX\src\pyeval.jl', 10, 1, '
                                return PyDict{String,PyObject,true}(pyinref(@pycheckn ccall((@pysym :PyModule_GetDict), PyPtr, (PyPtr,), pyimport("_main_")))), 10, 11))

```

Stacktrace:

```

[1] pyerr_check
    @ C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX\src\exception.jl:75 [inlined]
[2] pyerr_check
    @ C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX\src\exception.jl:79 [inlined]
[3] _handle_error(msg::String)
    @ PyCall C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX\src\exception.jl:96
[4] macro expansion
    @ C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX\src\exception.jl:110 [inlined]
[5] pyeval_(s::String, globals::PyDict{String, PyObject, true}, locals::PyDict{String, PyObject, true}, input_type::Int64, fname::String)
    @ PyCall C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX\src\pyeval.jl:34
[6] top-level scope
    @ C:\Users\vemanaeva\.julia\packages\PyCall\ilqDX\src\pyeval.jl:230

```

In [46]:

```

using BenchmarkTools
@btime a,b = find_best_fit_python(xvals,yvals)
@btime a,b = find_best_fit(xvals,yvals)

```

```
UndefVarError: `find_best_fit_python` not defined

Stacktrace:
 [1] var##core#304}()
   @ Main C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\execution.jl:489
 [2] var##sample#305"(::Tuple{}, __params::BenchmarkTools.Parameters)
   @ Main C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\execution.jl:495
 [3] _run(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters; verbose::Bool,
pad::String, kwargs::Base.Pairs{Symbol, Integer, NTuple{4, Symbol}, NamedTuple{(:sam
ples, :evals, :gctrail, :gcsample), Tuple{Int64, Int64, Bool, Bool}}})
   @ BenchmarkTools C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\exe
cution.jl:99
 [4] #invokelatest#2
   @ .\essentials.jl:821 [inlined]
 [5] invokelatest
   @ .\essentials.jl:816 [inlined]
 [6] #run_result#45
   @ C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\execution.jl:34 [i
nlined]
 [7] run_result
   @ C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\execution.jl:34 [i
nlined]
 [8] run(b::BenchmarkTools.Benchmark, p::BenchmarkTools.Parameters; progressid::Not
hing, nleaves::Float64, ndone::Float64, kwargs::Base.Pairs{Symbol, Integer, NTuple
{5, Symbol}, NamedTuple{(:verbose, :samples, :evals, :gctrail, :gcsample), Tuple{Boo
l, Int64, Int64, Bool, Bool}}})
   @ BenchmarkTools C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\exe
cution.jl:117
 [9] run (repeats 2 times)
   @ C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\execution.jl:117
[inlined]
 [10] #warmup#54
   @ C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\execution.jl:169
[inlined]
 [11] warmup(item::BenchmarkTools.Benchmark)
   @ BenchmarkTools C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\exe
cution.jl:168
 [12] top-level scope
   @ C:\Users\vemanaeva\.julia\packages\BenchmarkTools\0owsb\src\execution.jl:575
```

Самостоятельная работа

Кластеризация

Используйте Clustering.jl для кластеризации на основе k-средних. Сделайте точечную диаграмму полученных кластеров. Подсказка: вам нужно будет проиндексировать фрейм данных, преобразовать его в массив и транспонировать.

In [47]: **using** GLM

```
In [48]: iris = dataset("datasets", "iris")
select!(iris, Not(:Species))
X = Matrix(iris)
X = X'

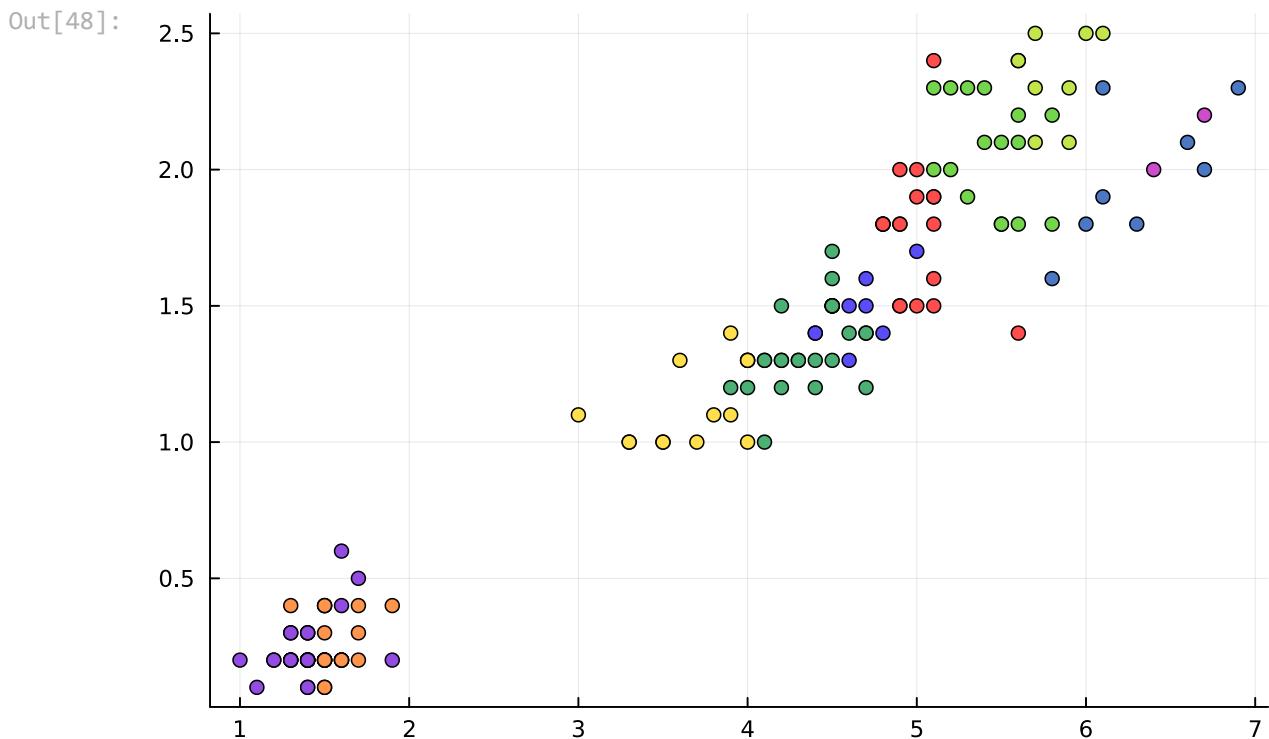
result = kmeans(X, 10, maxiter=10, display=:iter)
display(result)

scatter(iris.PetalLength, iris.PetalWidth, marker_z=result.assignments, color=:lightblue)
```

Iters	objv	objv-change affected
0	5.666000e+01	
1	3.367176e+01	-2.298824e+01 9
2	3.065446e+01	-3.017298e+00 6
3	2.882822e+01	-1.826248e+00 5
4	2.814717e+01	-6.810483e-01 5
5	2.795575e+01	-1.914226e-01 3
6	2.785331e+01	-1.024312e-01 2
7	2.783033e+01	-2.298006e-02 0
8	2.783033e+01	0.000000e+00 0

K-means converged with 8 iterations (objv = 27.830334723886182)

KmeansResult{Matrix{Float64}, Float64, Int64}([7.800000000000001 4.818181818181818 ...
5.370588235294117 6.0058823529411764; 3.8 3.236363636363636 ... 3.8 2.735294117647059
3; 6.550000000000001 1.4333333333333331 ... 1.5176470588235293 5.011764705882351; 2.1
0.23030303030303034 ... 0.2764705882352942 1.788235294117647], [2, 2, 2, 2, 2, 9, 2,
2, 2, 2 ... 7, 6, 10, 7, 6, 10, 6, 6, 10], [0.15095500459136701, 0.06459136822772
393, 0.03398530762167695, 0.07156106519741456, 0.1673186409549885, 0.059377162629758
12, 0.08034894398529957, 0.0651974288337982, 0.2900459136822633, 0.04671258034895231
5 ... 0.08703124999999545, 0.3177508650519485, 0.0639100346020598, 0.009531250000009
095, 0.05453124999996817, 0.1389273356401759, 0.15449826989620874, 0.065397923875423
2, 0.300692041522467, 0.08920415224912404], [2, 33, 12, 8, 23, 17, 8, 13, 17, 17],
[2, 33, 12, 8, 23, 17, 8, 13, 17, 17], 27.830334723886182, 8, true)



Регрессия (метод наименьших квадратов в случае линейной регрессии)

Часть 1.

Для самостоятельного решения необходимо добавить колонку единиц и решить СЛАУ, где матрица X с добавленной колонкой единиц является матрицей коэффициентов, а вектор y --- вектором ответов. Колонка единиц добавляется для того, чтобы решение ограничивалось исключительно линейными случаями (так как производная линейной функции равна 1).

In [49]:

```
X = randn(1000, 3)
a0 = rand(3)
y = X * a0 + 0.1 * randn(1000)
```

```
Out[49]: 1000-element Vector{Float64}:
-0.7945765229015872
-0.6720296638517435
1.3716244627873222
2.1785938205075484
-0.3260138255273183
0.26170158513295094
0.9557760816312258
-2.304971670565372
-0.6942984013717575
-0.9267042171300182
-1.2557384299492562
-1.6445050915198212
1.5532534557172666
⋮
-1.6250530641171432
0.6461356986934509
-0.6007257958443772
-0.010337923351421996
1.328135396010722
-1.983401200698946
-0.14795949976913234
-1.3949207950494904
1.3520963911727373
-1.8038372400597185
0.6129175787211228
-0.16580110961773115
```

```
In [50]: N = 1000
X2 = hcat(ones(N), X)
```

```
Out[50]: 1000x4 Matrix{Float64}:
 1.0 -1.11694  0.644413 -0.217648
 1.0  0.0169132 -3.37187 -0.46774
 1.0 -0.377588 -0.516486  2.02971
 1.0  1.76627  -0.0865646 1.21452
 1.0 -0.160411  1.0795  -0.616407
 1.0 -0.239455 -1.6024  0.592198
 1.0 -0.294295 -0.119782 1.64942
 1.0  0.174658 -1.81967 -2.58481
 1.0  0.586109 -0.217839 -1.3919
 1.0 -0.227832 -1.57785 -0.780279
 1.0 -0.779791  0.705667 -0.947964
 1.0 -1.06939  -2.02278 -0.96679
 1.0  1.9599   0.108364  0.154524
 :
 1.0 -1.87459 -0.0208567 -0.152682
 1.0 -0.343614 -0.24057  1.08071
 1.0  0.468206  1.78004 -1.55159
 1.0 -0.524671  2.51546  0.178462
 1.0  2.25013 -0.118962 -0.477887
 1.0 -1.27969 -2.2827  -1.07568
 1.0 -1.29073  0.22623  1.09623
 1.0 -1.96463 -0.676192 0.228304
 1.0  0.794014 -0.378956 0.901407
 1.0 -2.10044  0.719611 -0.520299
 1.0  1.91276 -2.24416 -0.702585
 1.0  0.591597 -0.296568 -0.848651
```

```
In [51]: betahat1 = X2 \ y
yp = X2 * betahat1
mse1 = sqrt(sum(abs2.(y - yp)) / N)
display(betahat1)
mse1
```

```
4-element Vector{Float64}:
 0.003675260146041499
 0.7485522672623919
 0.09827734019830424
 0.808080938633673
```

```
Out[51]: 0.09996843006061548
```

```
In [52]: betahat2 = llsq(X, y; bias=false)
yp = X * betahat2
mse2 = sqrt(sum(abs2.(y - yp)) / N)
display(betahat2)
mse2
```

```
3-element Vector{Float64}:
 0.7485913584036923
 0.0981758667102467
 0.8079549588733729
```

```
Out[52]: 0.10003581652171566
```

```
In [53]: X3 = DataFrame(a=y, b=X[:,1], c=X[:,2], d=X[:,3])
lmMSE = lm(@formula(a ~ b + c + d), X3)
```

```

betahat3 = GLM.coeftable(lmMSE).cols[1]
yp = X2 * betahat3
mse3 = sqrt(sum(abs2.(y - yp)) / N)
display(betahat3)
mse3

```

4-element Vector{Float64}:

0.0036752601460415322
 0.7485522672623924
 0.09827734019830442
 0.8080809386336726

Out[53]: 0.09996843006061547

Часть 2.

```

In [54]: X = rand(100)
y = 2X + 0.1 * randn(100)
Xh = hcat(ones(100), X)
lm2 = fit(LinearModel, Xh, y)
display(lm2)

Plots.plot(title="График регрессии", xlabel="x", ylabel="y", legend=:topleft)
Plots.plot!(X, predict(lm2), label="Линия регрессии")
Plots.scatter!(X, y, label="Данные")

```

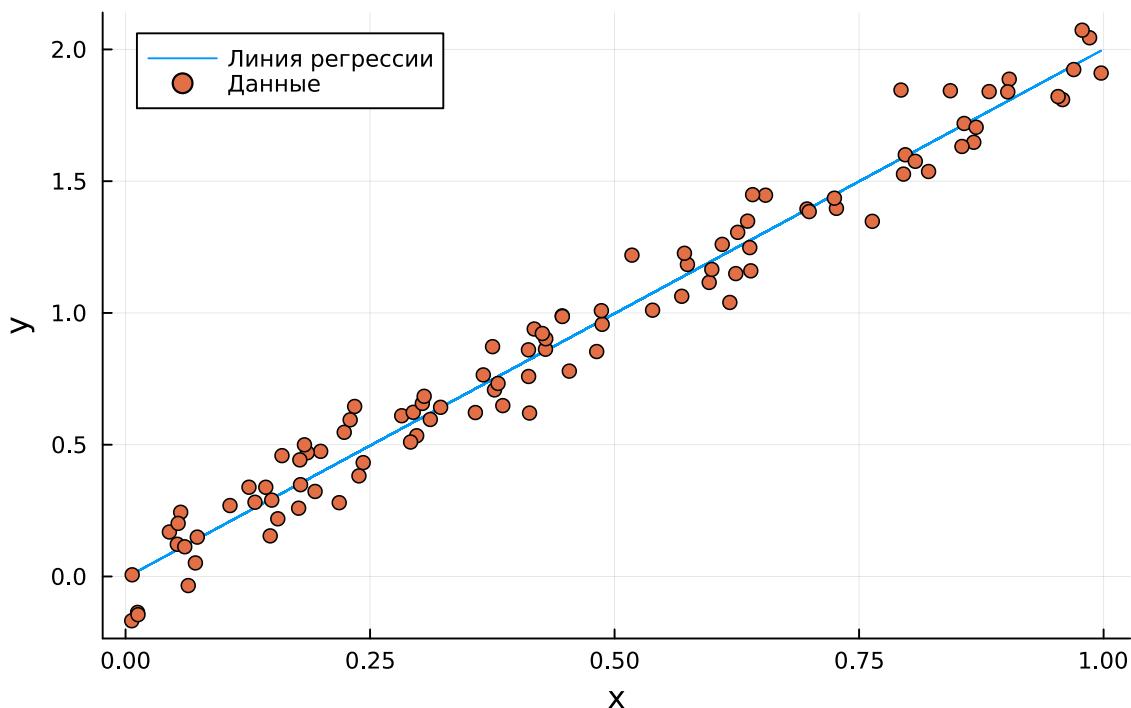
LinearModel{GLM.LmResp{Vector{Float64}}, GLM.DensePredChol{Float64, LinearAlgebra.CholeskyPivoted{Float64, Matrix{Float64}}, Vector{Int64}}}:

Coefficients:

	Coef.	Std. Error	t	Pr(> t)	Lower	95%	Upper	95%
x1	-0.00481807	0.0181552	-0.27	0.7913	-0.0408465	0.0312103		
x2	2.00541	0.0341101	58.79	<1e-77	1.93772	2.0731		

Out[54]:

График регрессии



Модель ценообразования биномиальных опционов

Пункты a и d .

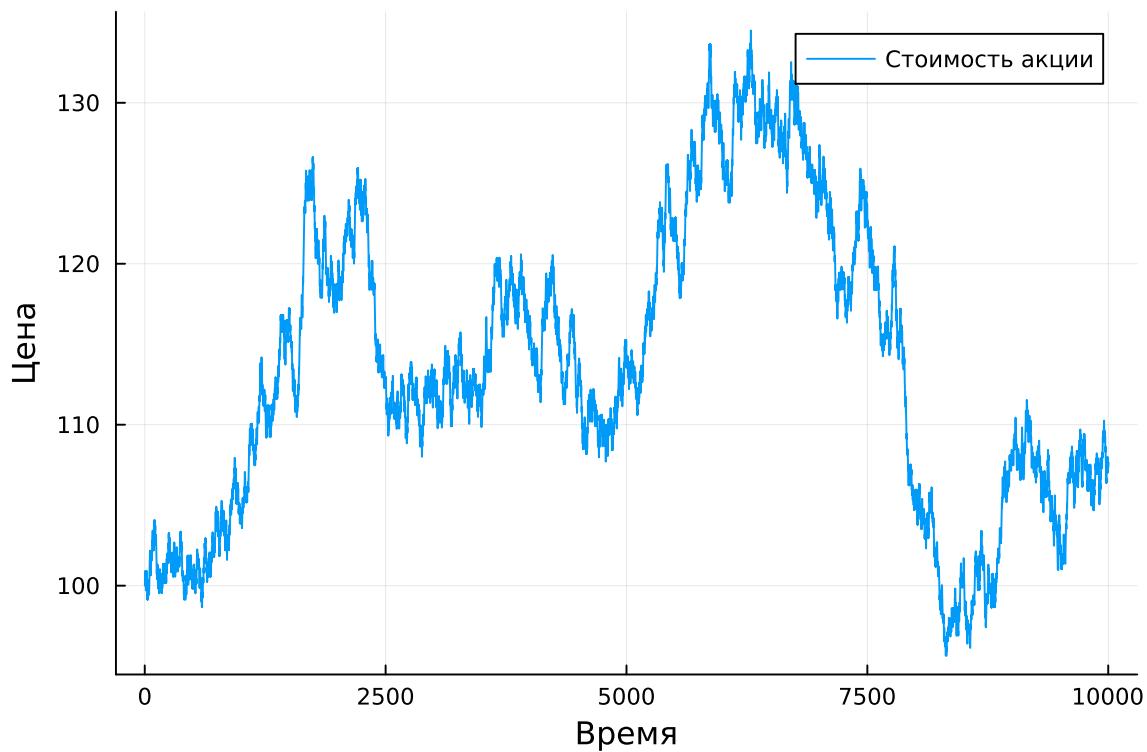
In [55]:

```
function binomial_stock_price(S, T, n, sigma, r)
    h = T/n
    u = exp(r * h + sigma * sqrt(h))
    d = exp(r * h - sigma * sqrt(h))
    p = (exp(r * h) - d) / (u - d)
    stock_prices = zeros(n+1)
    stock_prices[1] = S
    for i in 2:n+1
        epsilon = rand()
        if epsilon > 0.5
            stock_prices[i] = stock_prices[i-1]*u
        else
            stock_prices[i] = stock_prices[i-1]*d
        end
    end
    return stock_prices
end

S, T, n, sigma, r = 100.0, 1.0, 10000, 0.3, 0.08
stock_prices = binomial_stock_price(S, T, n, sigma, r)

plot(stock_prices, xlabel="Время", ylabel="Цена", label="Стоимость акции")
```

Out[55]:



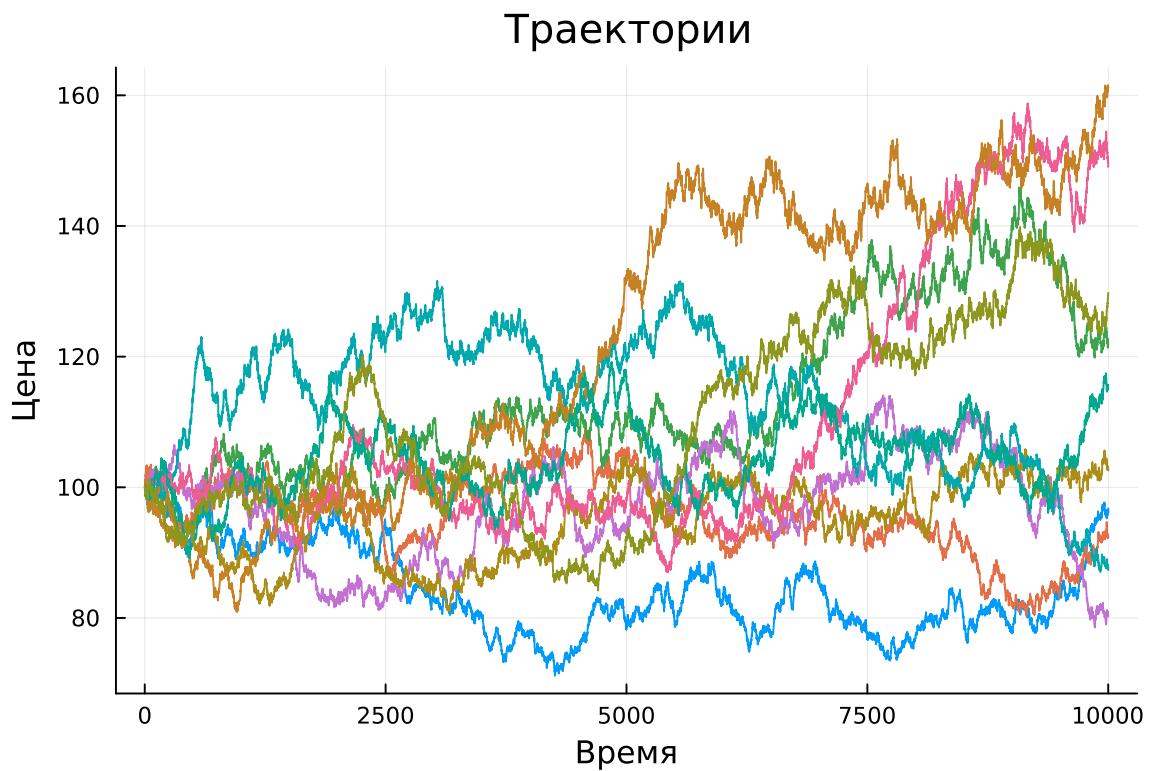
In [56]:

```
function createPath(S::Float64, r::Float64, sigma::Float64, T::Float64, n::Int64)
    dt = T / n
    path = [S]

    for i in 1:n
        epsilon = randn()
        S = S * exp((r - 0.5 * sigma^2) * dt + sigma * sqrt(dt) * epsilon)
        push!(path, S)
    end
    return path
end

paths = [createPath(S, r, sigma, T, n) for i in 1:10]
plot(paths, title="Траектории", xlabel="Время", ylabel="Цена", legend=false)
```

Out[56]:



In [57]:

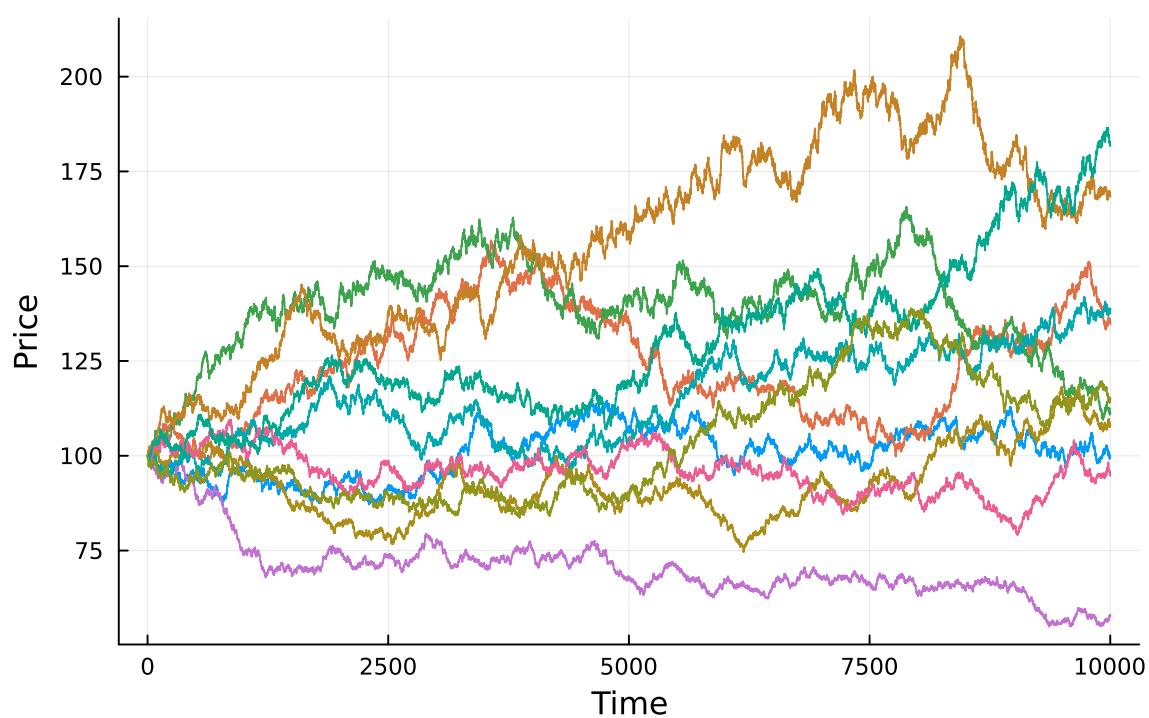
```
using Random
using Base.Threads

paths2 = []
@threads for i = 1:10
    push!(paths2, createPath(S, r, sigma, T, n))
end

plot(paths2, title="Stock Price Paths (threads)", xlabel="Time", ylabel="Price", le
```

Out[57]:

Stock Price Paths (threads)



In []: