Отчёт по лабораторной работе №7
Введение в работу с данными

Статический анализ данных

Выполнила: Коняева Марина Александровна,
НФИбд-01-21, 1032217044

Содержание

Цель работы **4**

Теоретическое введение **5**

Задачи лабораторной работы **6**

Выполнение лабораторной работы **7**
Julia для науки о данных 7
Считывание данных 7
Словари 9
DataFrames 10
RDatasets 12
Работа с переменными отсутствующего типа (Missing Values) 13
Обработка данных: стандартные алгоритмы машинного обучения в Julia . . . 14
Кластеризация данных. Метод k-средних 14
Кластеризация данных. Метод k ближайших соседей 19
Обработка данных. Метод главных компонент 20
Обработка данных. Линейная регрессия 23

Задания для самостоятельного выполнения **26**

Выводы по проделанной работе **38**
Вывод 38

Список литературы **39**

Список иллюстраций

1 Считывание данных из файла 8
2 Функция и считывание данных 9
3 Работа со словарем 10
4 Работа с DataFrames 11
5 Работа с пакетом RDatasets и вывод набора данных 12
6 Действия с переменными отсутствующего типа 13
7 Действия с переменными отсутствующего типа 14
8 Метод k-средних 15
9 Метод k-средних 16
10 Кластеризация данных 17
11 Кластеризация объектов 18
12 Кластеризация объектов 19
13 Метод k ближайших соседей 20
14 Метод главных компонент 22
15 Функция линейной регрессии 24
16 Функция линейной регрессии 25

1 Загрузка датасета 27
2 Загрузка датасета 28
3 Транспонирование матрицы с данными, задание количества кластеров и определение k-среднего 29
4 Формирование фрейма данных 30
5 Построение графика кластеризации данных 31
6 Задание значений матрицы наблюдений и матрицы данных 32
7 Создание фрейма данных и решений системы уравнений 33
8 Построение графика линии регрессии 34
9 График траектории курса акций 35
10 Код 36
11 График 10 разных траекторий цен на акции 36
12 Код 37
13 График 10 разных траекторий цен на акции 37

Цель работы

Освоить специализированные пакеты Julia для обработки данных.

Теоретическое введение

Обработка и анализ данных, полученных в результате проведения исследований, — важная и неотъемлемая часть исследовательской деятельности. Большое значение имеет выявление определённых связей и закономерностей в имеющихся неструктурированных данных, особенно в данных больших размерностей. Выявленные в данных связей и закономерностей позволяет строить прогнозные модели с предполагаемым результатом. Для решения таких задач применяют методы из таких областей знаний как математическая статистика, программирование, искусственный интеллект, машинное обучение.

Задачи лабораторной работы

1. Используя Jupyter Lab, повторите примеры из раздела 7.2.
2. Выполните задания для самостоятельной работы (раздел 7.4).

Выполнение лабораторной работы

Julia для науки о данных

В Julia для обработки данных используются наработки из других языков программирования, в частности, из R и Python.

Считывание данных

Установим нужные пакеты, загрузим в систему необходимые файлы с данными, а затем считаем данные из файла и запишем их в структуру:

```julia
# Обновление окружения:
using Pkg
Pkg.update
# Установка пакетов:
using Pkg
for p in ["CSV", "DataFrames", "RDatasets", "FileIO"]
Pkg.add(p)
end
using CSV, DataFrames, DelimitedFiles
```

```
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
```

```julia
# 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
```

```
language_created_year (generic function with 1 method)
```

```julia
# Считывание данных и их запись в структуру:
P = CSV.File("programminglanguages.csv") |> DataFrame
```

73×2 DataFrame

Row	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
⋮	⋮	⋮
62	2003	Scala
63	2005	F#
64	2006	PowerShell
65	2007	Clojure
66	2009	Go
67	2010	Rust
68	2011	Dart

Рис. 1: Считывание данных из файла

Напишем функцию для определения по названию языка программирования года его создания и протестируем её. Затем построчно считаем данные с указанием разделителя и запишем данные в CSV-файл:

```
# Функция определения по названию языка программирования
# года его создания (без учёта регистра):
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")
```

```
2012
```

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

```
74×2 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 CSV-файл:
CSV.write("programming_languages_data2.csv", P)
```

```
"programming_languages_data2.csv"
```

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

Рис. 2: Функция и считывание данных

Словари

Я инициализировала словарь и заполнила его данными. При инициализации словаря можно задать конкретные типы данных для ключей и значений:

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

```
Dict{Integer, Vector{String}}()
```

```
# 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
```

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

```
2-element Vector{String}:
 "Groovy"
 "Scala"
```

Рис. 3: Работа со словарем

DataFrames

Работа с данными, записанными в структуре DataFrame, позволяет использовать индексацию и получить доступ к столбцам по заданному имени заголовка или по индексу столбца.

Подгрузила необходимый пакет DataFrames, задала переменную со структурой DataFrame и вывела ее:

```
# Подгружаем пакет DataFrames:
using DataFrames
# Задаём переменную со структурой DataFrame:
df = DataFrame(year = P[:,1], language = P[:,2])
```

73×2 DataFrame

Row	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
⋮	⋮	⋮
62	2003	Scala
63	2005	F#
64	2006	PowerShell
65	2007	Clojure
66	2009	Go
67	2010	Rust
68	2011	Dart
69	2011	Kotlin
70	2011	Red
71	2011	Elixir
72	2012	Julia
73	2014	Swift

```
# Вывод всех значения столбца year:
df[!,:year]
```

```
73-element Vector{Int64}:
 1951
 1952
 1954
 1955
 1957
 1957
 1958
 1958
 1959
 1959
 1959
 1962
 1962
    ⋮
 2003
 2005
```

Рис. 4: Работа с DataFrames

RDatasets

С данными можно работать также как с наборами данных через пакет RDatasets языка R. Подгрузила пакет RDatasets и задала структуру данных в виде набора данных:

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

2×7 DataFrame

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

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

150×5 DataFrame

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

Рис. 5: Работа с пакетом RDatasets и вывод набора данных

Работа с переменными отсутствующего типа (Missing Values)

Пакет DataFrames позволяет использовать так называемый «отсутствующий» тип.

Выполняем действия с переменными отсутствующего типа:

```
# Определения типа переменной:
typeof(iris)
```

```
DataFrame
```

```
describe(iris)
```

5×7 DataFrame

Row	variable	mean	min	median	max	nmissing	eltype
	Symbol	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}

```
# Отсутствующий тип:
a = missing
typeof(a)
```

```
Missing
```

```
# Пример операции с переменной отсутствующего типа:
a + 1
```

```
missing
```

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

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

```
# Определение типа переменной:
typeof(calories)
```

```
Vector{Union{Missing, Int64}} (alias for Array{Union{Missing, Int64}, 1})
```

Рис. 6: Действия с переменными отсутствующего типа

```
# Подключаем пакет Statistics:
using Statistics
```

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

```
missing
```

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

```
58.0
```

```
# 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 = outerjoin(dataframe_calories,dataframe_prices,on=:item)
```

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

Рис. 7: Действия с переменными отсутствующего типа

Обработка данных: стандартные алгоритмы машинного обучения в Julia

Кластеризация данных. Метод k-средних

Задача кластеризации данных заключается в формировании однородной группы упорядоченных по какому-то признаку данных. Метод k-средних позволяет минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров.

Рассмотрим задачу кластеризации данных на примере данных о недвижимости. Файл с данными houses.csv содержит список транзакций с недвижимостью в районе Сакраменто, о которых было сообщено в течение определённого числа дней.

Загружаю пакеты, а также данные из файла и строю графики, а затем произвожу кластеризацию данных:

```
# Загрузка пакетов:
import Pkg
Pkg.add("DataFrames")
Pkg.add("Statistics")
using DataFrames
using CSV
import Pkg
Pkg.add("Plots")
```

```
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
```

```
# Загрузка данных:
houses = CSV.File("houses.csv") |> DataFrame
```

985×12 DataFrame — *960 rows omitted*

Row	street	city	zip	state	beds	baths	sq_ft	type	sale_date	price	latitude	longitude
	String	String15	Int64	String3	Int64	Int64	Int64	String15	String31	Int64	Float64	Float64
1	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836	Residential	Wed May 21 00:00:00 EDT 2008	59222	38.6319	-121.435
2	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167	Residential	Wed May 21 00:00:00 EDT 2008	68212	38.4789	-121.431
3	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796	Residential	Wed May 21 00:00:00 EDT 2008	68880	38.6183	-121.444
4	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed May 21 00:00:00 EDT 2008	69307	38.6168	-121.439
5	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797	Residential	Wed May 21 00:00:00 EDT	81900	38.5195	-121.436

```
# Построение графика:
using Plots
plot(size=(500,500),leg=false)
x = houses[!,:sq__ft]
y = houses[!,:price]
scatter(x,y,markersize=3)
```

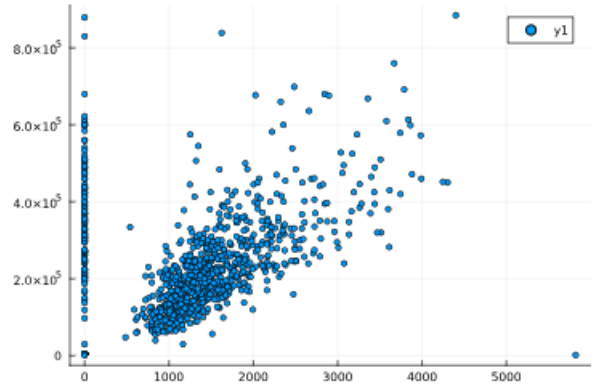
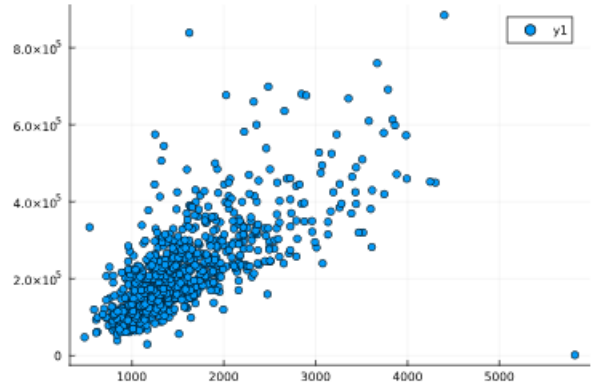


Рис. 8: Метод k-средних

Используя для фильтрации значений функцию by пакета DataFrames и для вычисления среднего значения функцию mean пакета Statistics, можно посмотреть среднюю цену домов определённого типа:

```
# 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)
```




```
using DataFrames
using CSV
import Pkg
Pkg.add("Plots")
```

```
  Resolving package versions...
  No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
  No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
```

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

3×2 DataFrame

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

Рис. 9: Метод k-средних

Отфильтровав таким образом данные, можно приступить к формированию кластеров. Сначала подключаем необходимые пакеты и формируем данные в нужном виде:

```
import Pkg
Pkg.add("Clustering")
using Clustering
```

```
  Resolving package versions...
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
 No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
```

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

814×2 DataFrame 789 ro

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

```
# Конвертация данных в матричный вид:
X = Matrix(X)
```

```
814×2 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
```

Рис. 10: Кластеризация данных

Каждая функция хранится в виде строки X, но можно транспонировать получившуюся матрицу, чтобы иметь возможность работать с столбцами данных X:

```
# Транспонирование матрицы с данными:
X = X'
```

```
2×814 adjoint(::Matrix{Float64}) with eltype Float64:
   38.6319    38.4789    38.6183  …    38.7088    38.417    38.6552
 -121.435   -121.431   -121.444      -121.257   -121.397  -121.076
```

```
# Задание количества кластеров:
k = length(unique(filter_houses[!,:zip]))
```

```
66
```

```
# Определение k-среднего:
C = kmeans(X,k)
```

```
KmeansResult{Matrix{Float64}, Float64, Int64}([38.28195836363636 38.7155373076923 … 38.69318723076923 38.786281875; -121.29388763636362 -121.39096746153845 … -121.45054476923073 -121.306017], [51, 26, 11, 11, 38, 59, 6, 18, 64, 65  …  19, 14, 56, 33, 45, 46, 4, 42, 53, 3], [8.7414540757890796e-5, 0.00040097561213769995, 2.1663516236003488e-5, 9.1543406597338626e-5, 0.00011844087202916853, 0.00017304826178587973, 0.00011862715109600686, 0.00030225235241232244, 9.2562331701628866e-6, 0.00011686658399412408  …  4.1920327930711216e-5, 5.2413251978578046e-5, 0.00011806063048425 13, 0.00023417692508515 84, 0.00013415837747743353, 0.00014433951582759 62, 0.00016121321459650862, 9.2051098177847076e-5, 0.00037928381389911 21, 7.0613299612887 2e-5], [11, 13, 7, 14, 11, 19, 3, 2, 11, 1  …  21, 20, 17, 5, 4, 22, 15, 8, 13, 8], [11, 13, 7, 14, 11, 19, 3, 2, 11, 1  …  21, 20, 17, 5, 4, 22, 15, 8, 13, 8], 0.2177436743366 81618, 11, true)
```

```
# Формирование фрейма данных:
df = DataFrame(cluster = C.assignments,city = filter_houses[!,:city],
latitude = filter_houses[!,:latitude],longitude = filter_houses[!,:longitude],zip = filter_houses[!,:zip])
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")
```

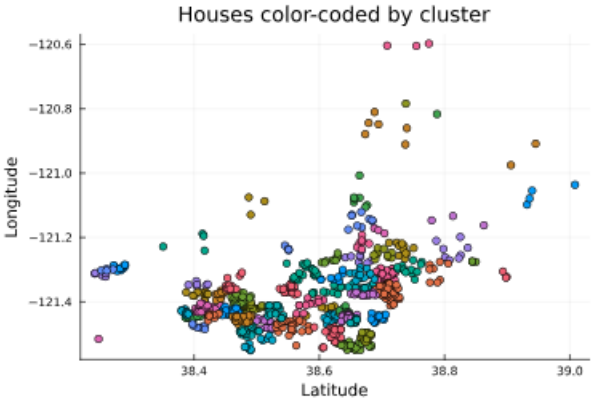


Рис. 11: Кластеризация объектов

```
display(clusters_figure)
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)
```

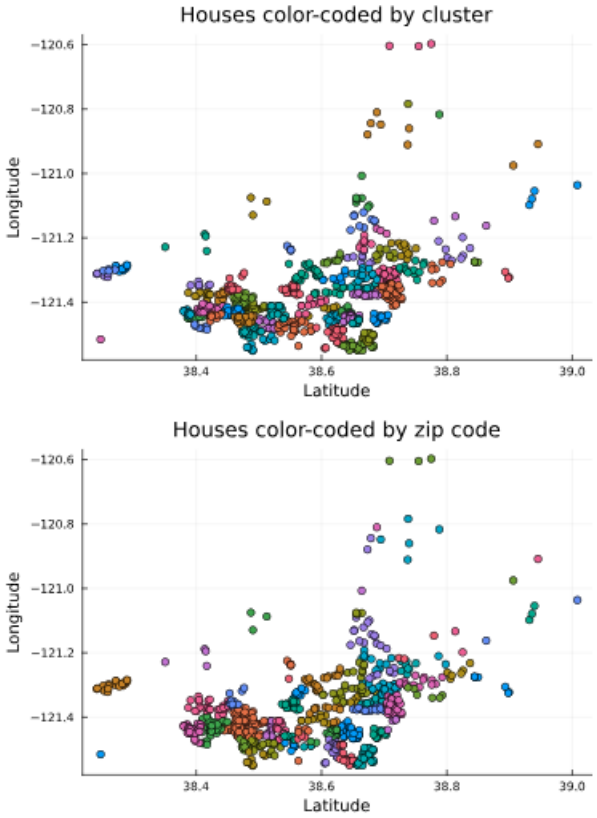


Рис. 12: Кластеризация объектов

Кластеризация данных. Метод k ближайших соседей

Данный метод заключается в отнесении объекта к тому из известных классов, который является наиболее распространённым среди □ соседей данного элемента. В случае использования метода для регрессии, объекту присваивается среднее значение по □ ближайшим к нему объектам.

Решаю задачу методом k ближайших соседей и строю график:

```
# Подключение пакета NearestNeighbors:
import Pkg
Pkg.add("NearestNeighbors")
```

```
   Resolving package versions...
    Updating `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
  [b8a86587] + NearestNeighbors v0.4.21
  No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
```

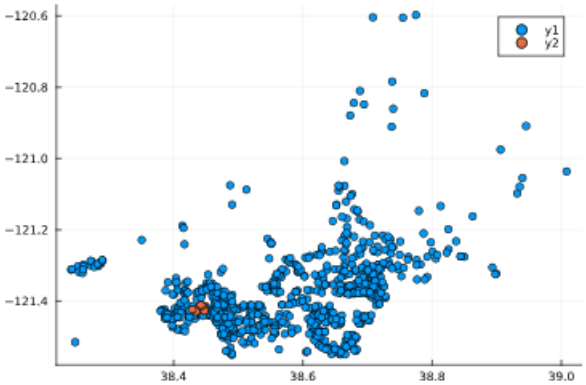
```
using NearestNeighbors
knearest = 10
id = 70
point = X[:,id]
```

```
2-element Vector{Float64}:
   38.44004
 -121.421012
```

```
# Поиск ближайших соседей:
kdtree = KDTree(X)
idxs, dists = knn(kdtree, point, knearest, true)
```

```
([70, 764, 196, 125, 557, 368, 415, 92, 112, 683], [0.0, 0.006264891539364138, 0.008253202590504 62, 0.008473585132630057, 0.009164073548
5065124697706, 0.009921759722950759, 0.009941028618812013, 0.010332637707777167, 0.011168993911721985])
```

```
# 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)
```




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

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

Рис. 13: Метод k ближайших соседей

Обработка данных. Метод главных компонент

Метод главных компонент (Principal Components Analysis, PCA) позволяет уменьшить размерность данных, потеряв наименьшее количество полезной информации. Метод

имеет широкое применение в различных областях знаний, например, при визуализации данных, компрессии изображений, в эконометрике, некоторых гуманитарных предметных областях, например, в социологии или в политологии.

На примере с данными о недвижимости попробуем уменьшить размеры данных о цене и площади из набора данных домов.

Подключаю необходимые пакеты и строю график с выделением главных компонент:

```
# Фрейм с указанием площади и цены недвижимости:
F = filter_houses[!,[:sq__ft,:price]]
# Конвертация данных в массив:
F = Array(F)'
```

```
2×814 adjoint(::Matrix{Int64}) with eltype Int64:
   836   1167    796    852    797   1122  …    1477    1216    1685    1362
 59222  68212  68880  69307  81900  89921     234000  235000  235301  235738
```

```
# Подключение пакета MultivariateStats:
import Pkg
Pkg.add("MultivariateStats")
```

```
   Resolving package versions...
  No Changes to `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
  No Changes to `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
```

```
using MultivariateStats
# Приведение типов данных к распределению для PCA:
M = fit(PCA, F)
```

```
PCA(indim = 2, outdim = 1, principalratio = 0.9999840784692097)

Pattern matrix (unstandardized loadings):
──────────
           PC1
──────────
1  460.52
2    1.19826e5
──────────

Importance of components:
──────────────────────────────────────
                                  PC1
──────────────────────────────────────
SS Loadings (Eigenvalues)  1.43584e10
Variance explained         0.999984
Cumulative variance        0.999984
Proportion explained       1.0
Cumulative proportion      1.0
──────────────────────────────────────
```

```
# Выделение значений главных компонент в отдельную переменную:
Xr = reconstruct(M, y')
```

```
2×10 Matrix{Float64}:
 1590.68       1590.68       …   1590.68    1590.68  1590.68
    2.29327e5     2.29327e5      229327.0   229327.0     2.29327e5
```

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

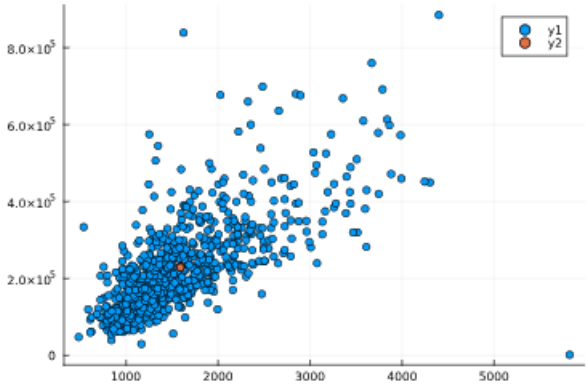


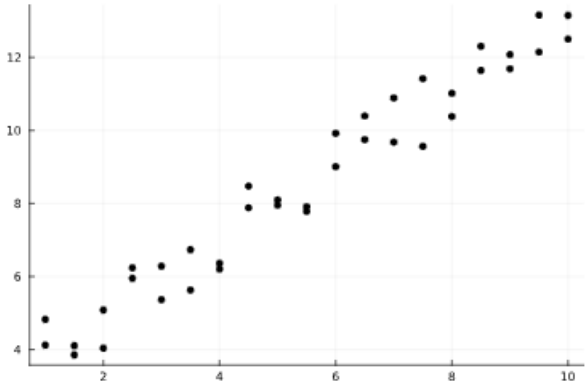
Рис. 14: Метод главных компонент

Обработка данных. Линейная регрессия

Регрессионный анализ представляет собой набор статистических методов исследования влияния одной или нескольких независимых переменных (регрессоров) на зависимую (критериальная) переменную. Терминология зависимых и независимых переменных отражает лишь математическую зависимость переменных, а не причинноследственные отношения. Наиболее распространённый вид регрессионного анализа — линейная регрессия, когда находят линейную функцию, которая согласно определённым математическим критериям наиболее соответствует данным.

Применяю функцию линейной регрессии для построения соответствующего графика::

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




```
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
```

```
find_best_fit (generic function with 1 method)
```

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

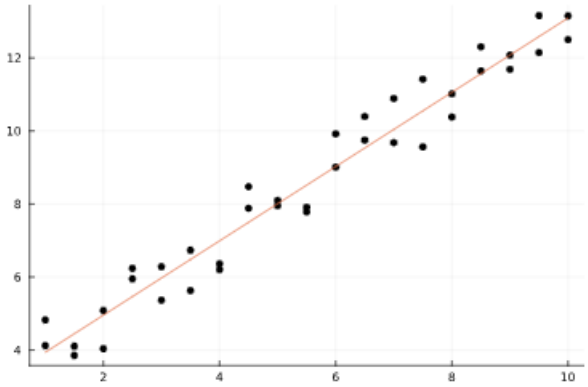


Рис. 15: Функция линейной регрессии

```
xvals = 1:100000;
xvals = repeat(xvals,inner=3);
yvals = 3 .+ xvals + 2*rand(length(xvals)) .- 1;
@show size(xvals)
@show size(yvals)
```

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

```
@time a,b = find_best_fit(xvals,yvals)
```

```
  0.066963 seconds (20.15 k allocations: 1.315 MiB, 95.55% compilation time)
(0.999999986637046, 3.0007307429841603)
```

Рис. 16: Функция линейной регрессии

Задания для самостоятельного выполнения

Затем приступим к заданиям для самостоятельного выполнения:

1. Кластеризация:

Загрузите using RDatasets iris = dataset(“datasets”, “iris”) 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

```
using RDatasets
iris = dataset("datasets", "iris")
```

150×5 DataFrame

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

Рис. 1: Загрузка датасета

```
using Clustering
using DataFrames
X = iris[!,[:SepalLength, :PetalWidth]]
```

150×2 DataFrame

Row	SepalLength	PetalWidth
	Float64	Float64
1	5.1	0.2
2	4.9	0.2
3	4.7	0.2
4	4.6	0.2
5	5.0	0.2
6	5.4	0.4
7	4.6	0.3
8	5.0	0.2
9	4.4	0.2
10	4.9	0.1
11	5.4	0.2
12	4.8	0.2
13	4.8	0.1
⋮	⋮	⋮
139	6.0	1.8
140	6.9	2.1
141	6.7	2.4
142	6.9	2.3
143	5.8	1.9
144	6.8	2.3
145	6.7	2.5
146	6.7	2.3
147	6.3	1.9
148	6.5	2.0
149	6.2	2.3
150	5.9	1.8

Рис. 2: Загрузка датасета

```
X = Matrix(X)
```

```
150×2 Matrix{Float64}:
 5.1  0.2
 4.9  0.2
 4.7  0.2
 4.6  0.2
 5.0  0.2
 5.4  0.4
 4.6  0.3
 5.0  0.2
 4.4  0.2
 4.9  0.1
 5.4  0.2
 4.8  0.2
 4.8  0.1
  ⋮
 6.0  1.8
 6.9  2.1
 6.7  2.4
 6.9  2.3
 5.8  1.9
 6.8  2.3
 6.7  2.5
 6.7  2.3
 6.3  1.9
 6.5  2.0
 6.2  2.3
 5.9  1.8
```

```
X = X'
```

```
2×150 adjoint(::Matrix{Float64}) with eltype Float64:
 5.1  4.9  4.7  4.6  5.0  5.4  4.6  5.0  …  6.8  6.7  6.7  6.3  6.5  6.2  5.9
 0.2  0.2  0.2  0.2  0.2  0.4  0.3  0.2     2.3  2.5  2.3  1.9  2.0  2.3  1.8
```

```
k = length(unique(iris[!,:Species]))
```

```
3
```

```
C = kmeans(X, k)
```

```
KmeansResult{Matrix{Float64}, Float64, Int64}([5.005555555555555 6.857142857142857 5.892592592592593; 0.3037037037037036 2.0119047619047614 1.4629629629629628], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1  …  2, 2, 3, 2, 2, 2, 2, 2, 2, 3], [0.01967421124828661, 0.02189643347050918, 0.10411865569273004, 0.17522976680384517, 0.010785322359396332, 0.1648593964334708, 0.16448902606310156, 0.010785322359396332, 0.3774519890260635, 0.052637174211248805  …  0.17531179138320852, 0.08483560090702724, 0.19957475994512208, 0.08626417233558925, 0.26293083900225156, 0.10769274376416149, 0.32293083900225383, 0.12769274376417172, 0.5148356089070199, 0.11364883401920167], [54, 42, 54], [54, 42, 54], 32.73746031746019, 4, true)
```

Рис. 3: Транспонирование матрицы с данными, задание количества кластеров и определение k-среднего

```
df = DataFrame(cluster = C.assignments, SepalLength = iris[!,:SepalLength],
SepalWidth = iris[!,:SepalWidth], Petallength = iris[!,:PetalLength],
Petalwidth = iris[!, :PetalWidth])
```

150×5 DataFrame

Row	cluster	SepalLength	SepalWidth	Petallength	Petalwidth
	Int64	Float64	Float64	Float64	Float64
1	1	5.1	3.5	1.4	0.2
2	1	4.9	3.0	1.4	0.2
3	1	4.7	3.2	1.3	0.2
4	1	4.6	3.1	1.5	0.2
5	1	5.0	3.6	1.4	0.2
6	1	5.4	3.9	1.7	0.4
7	1	4.6	3.4	1.4	0.3
8	1	5.0	3.4	1.5	0.2
9	1	4.4	2.9	1.4	0.2
10	1	4.9	3.1	1.5	0.1
11	1	5.4	3.7	1.5	0.2
12	1	4.8	3.4	1.6	0.2
13	1	4.8	3.0	1.4	0.1
⋮	⋮	⋮	⋮	⋮	⋮
139	3	6.0	3.0	4.8	1.8
140	2	6.9	3.1	5.4	2.1
141	2	6.7	3.1	5.6	2.4
142	2	6.9	3.1	5.1	2.3
143	3	5.8	2.7	5.1	1.9
144	2	6.8	3.2	5.9	2.3
145	2	6.7	3.3	5.7	2.5
146	2	6.7	3.0	5.2	2.3
147	2	6.3	2.5	5.0	1.9
148	2	6.5	3.0	5.2	2.0
149	2	6.2	3.4	5.4	2.3
150	3	5.9	3.0	5.1	1.8

Рис. 4: Формирование фрейма данных

```
using Plots
clusters_figure = plot(legend = false)
for i = 1:k
clustered_iris = df[df[!,:cluster].== i,:]
xvals = clustered_iris[!,:SepalLength]
yvals = clustered_iris[!,:Petalwidth]
scatter!(clusters_figure, xvals, yvals, markersize=4)
end
xlabel!("SepalLength")
ylabel!("Petalwidth")
title!("Iris color-coded by cluster")
display(clusters_figure)
```

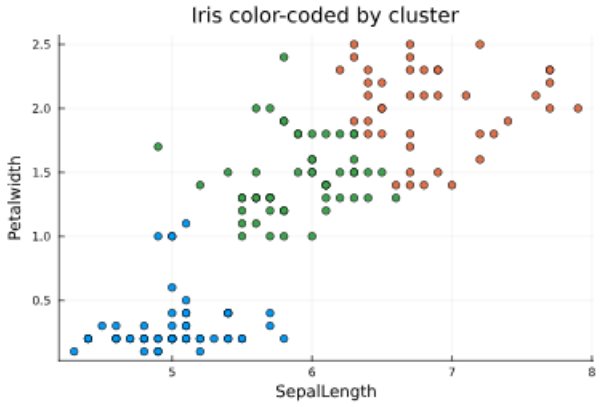


Рис. 5: Построение графика кластеризации данных

2. Регрессия (метод наименьших квадратов в случае линейной регрессии):

Часть 1 X = randn(1000, 3) a0 = rand(3) y = X * a0 + 0.1 * randn(1000);

Часть 2 X = rand(100); y = 2X + 0.1 * randn(100);

Часть 1: Пусть регрессионная зависимость является линейной. Матрица наблюдений факторов □ имеет размерность □ × 3 randn (N, 3), массив результатов □ × 1, регрессионная зависимость является линейной. Найдите МНК-оценку для линейной модели. – Сравните свои результаты с результатами использования llsq из MultivariateStats.jl (просмотрите документацию). – Сравните свои результаты с результатами использования регулярной регрессии наименьших квадратов из GLM.jl. Подсказка. Создайте матрицу данных X2, которая добавляет столбец единиц в начало матрицы данных, и решите систему линейных уравнений. Объясните с помощью теоретических выкладок.

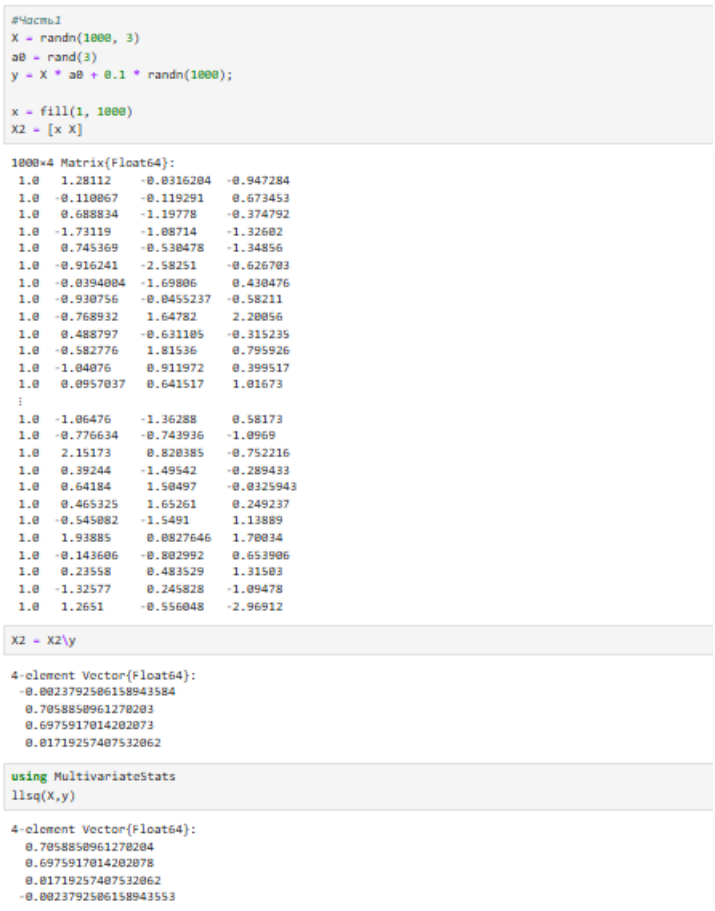


Рис. 6: Задание значений матрицы наблюдений и матрицы данных

```
import Pkg
Pkg.add("GLM")
using GLM
DF = DataFrame(y=y, x1=X[:,1], x2=X[:,2], x3=X[:,3])
lm(@formula(y ~ x1 + x2 + x3), DF)
```

```
   Resolving package versions...
   Installed GLM ──────────── v1.9.0
   Installed ShiftedArrays ─ v2.0.0
   Installed StatsModels ──── v0.7.4
    Updating `C:\Users\Admin\.julia\environments\v1.10\Project.toml`
  [38e38edf] + GLM v1.9.0
    Updating `C:\Users\Admin\.julia\environments\v1.10\Manifest.toml`
  [38e38edf] + GLM v1.9.0
  [1277b4bf] + ShiftedArrays v2.0.0
  [3eaba693] + StatsModels v0.7.4
Precompiling project...
  ✓ ShiftedArrays
  ✓ StatsModels
  ✓ GLM
  3 dependencies successfully precompiled in 17 seconds. 569 already precompiled.
```

```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}}, GLM.DensePredChol{Float64, LinearAlgebra.CholeskyPivoted{Float64, Matrix{Float
64}, Vector{Int64}}}}, Matrix{Float64}}

y ~ 1 + x1 + x2 + x3

Coefficients:
──────────────────────────────────────────────────────────────────────────────────
                  Coef.  Std. Error       t  Pr(>|t|)    Lower 95%   Upper 95%
──────────────────────────────────────────────────────────────────────────────────
(Intercept)  -0.00237925  0.00315398   -0.75    0.4508  -0.00856845  0.00380995
x1            0.705885    0.00319307  221.07    <1e-99   0.699619    0.712151
x2            0.697592    0.00316063  220.71    <1e-99   0.691389    0.703794
x3            0.0171926   0.00310787    5.53    <1e-07   0.0110938   0.0232913
──────────────────────────────────────────────────────────────────────────────────
```

```
using Statistics
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
```

```
find_best_fit (generic function with 1 method)
```

Рис. 7: Создание фрейма данных и решений системы уравнений

Часть 2: Найдите линию регрессии, используя данные (□, □). Постройте график (□, □), используя точечный график. Добавьте линию регрессии, используя abline!. Добавьте заголовок «График регрессии» и подпишите оси □ и □.

```
#Часть2
using Plots
X = rand(100);
y = 2X + 0.1 * randn(100);
a,b = find_best_fit(X,y)
scatter(X,y, title = "График регрессии", xlabel="x", ylabel="y")
Plots.abline!(a,b)
```

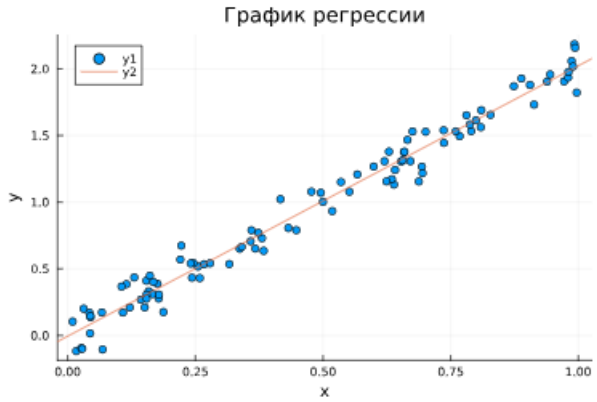


Рис. 8: Построение графика линии регрессии

3. Модель ценообразования биномиальных опционов:

Описание модели ценообразования биномиальных опционов можно найти на стр. https://en.wikipedia.org/wiki/Binomial_options_pricing_model.

a-d. Построение графика траектории курса акций (Пусть □ = 100, □ = 1, □ = 10000, □ = 0.3 и □ = 0.08. Попробуйте построить траекторию курса акций. Функция rand () генерирует случайное число от 0 до 1. Вы можете использовать функцию построения графика из библиотеки графиков.):

```
S=100
T=1
n=10000
h=T/n
sigma=0.3
r=0.08
u=exp(r*h+sigma*sqrt(h))
d=exp(r*h-sigma*sqrt(h))
p=(exp(r*h)-d)/(u-d)
```

```
0.4992500085625153
```

```
Pr = []
c = 0
append!(Pr, S)
for i in 1:n
    prob=rand()
    if (prob<p)
        append!(Pr,S*u^(i-c)*d^c)
    else
        append!(Pr,S*u^(i-c-1)*d^(c+1))
        c+=1
    end
end
```

```
plot(Pr, xlabel = "Период времени", ylabel = "Цена акции", title = "Траектория цен на акции")
```

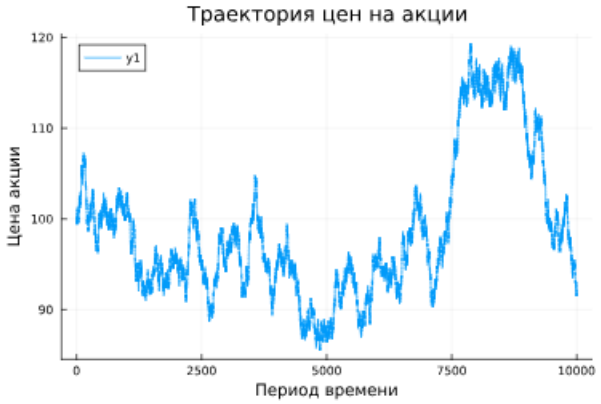


Рис. 9: График траектории курса акций

b. Построение графика 10 разных траекторий цен на акции (Создайте функцию createPath (S :: Float64, r :: Float64, sigma :: Float64, T :: Float64, n :: Int64), которая создает траекторию цены акции с учетом начальных параметров. Используйте createPath, чтобы создать 10 разных траекторий и построить их все на одном графике.):

```
function createPath(S, r, sigma, T, n)
    h=T/n
    u=exp(r*h+sigma*sqrt(h))
    d=exp(r*h-sigma*sqrt(h))
    p=(exp(r*h)-d)/(u-d)
    Pr=[]
    c=0
    append!(Pr,S)
    for i in 1:n
        prob=rand()
        if(prob<p)
            append!(Pr,S*u^(i-c)*d^c)
        else
            append!(Pr,S*u^(i-c-1)*d^(c+1))
            c+=1
        end
    end
    return Pr
end
```

```
createPath (generic function with 1 method)
```

```
p=plot()
@time for i in 1:10
    plot!(createPath(S,r,sigma,T,n), xlabel = "Период времени",
    ylabel = "Цена акции", title = "Траектория цен на акции",
    label = "Траектория №")
end
p
```

```
  0.076293 seconds (507.77 k allocations: 13.393 MiB, 48.00% compilation time)
```

Рис. 10: Код

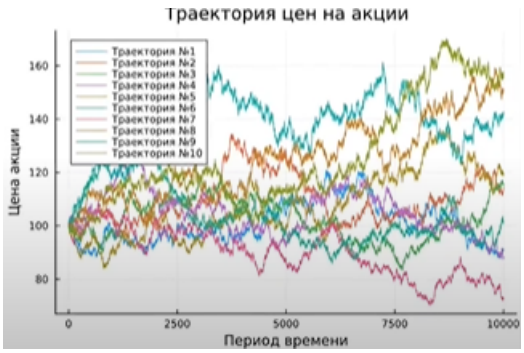


Рис. 11: График 10 разных траекторий цен на акции

c. То же задание, что и в предыдущем пункте, только с распараллеливанием генерации траекторий (Распараллельте генерацию траектории. Можете использовать Threads.@threads,pmap и @parallel.):

```
using Distributed
p=plot()
@time @distributed for i in 1:10
    plot!(createPath(S,r,sigma,T,n), xlabel = "Период времени",
    ylabel = "Цена акции", title = "Траектория цен на акции",
    label = "Траектория №$i")
end
p
```

```
  0.010352 seconds (3.80 k allocations: 273.828 KiB, 99.15% compilation time)
```

Рис. 12: Код

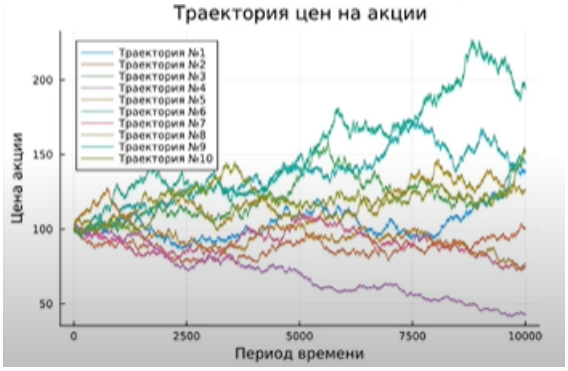


Рис. 13: График 10 разных траекторий цен на акции

Выводы по проделанной работе

Вывод

Я выполнила лабораторную работу №7 и успешно освоила специализированные пакеты Julia для обработки данных.

Список литературы

- Julia: https://ru.wikipedia.org/wiki/Julia
- https://julialang.org/packages/
- https://juliahub.com/ui/Home
- https://juliaobserver.com/
- https://github.com/svaksha/Julia.jl