Лабораторная работа №7

Введение в работу с данными

Лебедева Ольга Андреевна

Содержание

Список иллюстраций

Список таблиц

Цель работы

Основной целью работы является специализированных пакетов Julia для обработки данных.

Задачи

- Освоить специализированные пакеты Julia для обработки данных.
- Научиться применять методы кластеризации, регрессии и анализа данных с использованием Julia.
- Ознакомиться с работой с файлами данных в формате CSV и структурой DataFrame.

Термины и условные обозначения

- DataFrame — структура данных, аналогичная таблицам в базах данных.
- Кластеризация — метод машинного обучения для группировки объектов на основе их характеристик.
- Линейная регрессия — метод для нахождения линейной зависимости между переменными.
- PCA (Principal Component Analysis) — метод снижения размерности данных.
- k-средние — алгоритм кластеризации данных на основе центроидов.

Объект и предмет исследования

Объект исследования: данные о недвижимости, включая цену, площадь, географическое расположение, а также данные о языках программирования.

Предмет исследования: применение алгоритмов обработки данных для анализа и визуализации.

Техническое оснащение

Программное обеспечение: Jupyter Notebook, язык программирования Julia с установленными библиотеками (CSV, DataFrames, Clustering, Plots и др.).

Методы: 1. Считывание данных из файлов CSV. Кластеризация данных методами k-средних и k ближайших соседей. 2. Применение PCA для снижения размерности. 3. Линейная регрессия для выявления зависимостей между переменными.

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.

Задание лабораторной работы №7

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

Задание См. рис. 1, См. рис. 2

7.4. Задания для самостоятельного выполнения

7.4.1. 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.

7.4.2. 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 X имеет размерность $N \times 3$ `randn (N, 3)`, массив результатов $N \times 1$, 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.

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

Рис. 1: Задание_1

7.4.3. Модель ценообразования биномиальных опционов

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

Постройте траекторию возможных цен на акции:

- S — начальная цена акции;
- T — длина биномиального дерева в годах;
- n — количество периодов;
- $h = Tn$ — длина одного периода;
- σ — волатильность акции;
- r — годовая процентная ставка;
- $u = \exp(rh + \sigma\sqrt{h})$;
- $d = \exp(rh - \sigma\sqrt{h})$;
- $p^* = \dfrac{\exp(rh) - d}{u - d}$.

a) Пусть $S = 100, T = 1, n = 10000, \sigma = 0.3$ и $r = 0.08$. Попробуйте построить траекторию курса акций. Функция `rand()` генерирует случайное число от 0 до 1. Вы можете использовать функцию построения графика из библиотеки графиков.

b) 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.

c) Распараллельте генерацию траектории. Можете использовать `Threads.@threads`, `pmap` и `@parallel`.

d) Пусть $S = 100, T = 1, n = 10000, \sigma = 0.3$ и $r = 0.08$. Попробуйте построить траекторию курса акций. Функция `rand()` генерирует случайное число от 0 до 1. Вы можете использовать функцию построения графика из библиотеки графиков.

Рис. 2: Задание_2

Выполнение лабораторной работы.

Повторение примеров

Ниже приведены повторы примеров, данные в лабораторной работе для ознакомления: См. рис. 3, См. рис. 4, См. рис. 5, См. рис. 6, См. рис. 7, См. рис. 8, См. рис. 9, См. рис. 10, См. рис. 11, См. рис. 12, См. рис. 13, См. рис. 14, См. рис. 15, См. рис. 16, См. рис. 17, См. рис. 18, См. рис. 19, См. рис. 20, См. рис. 21.

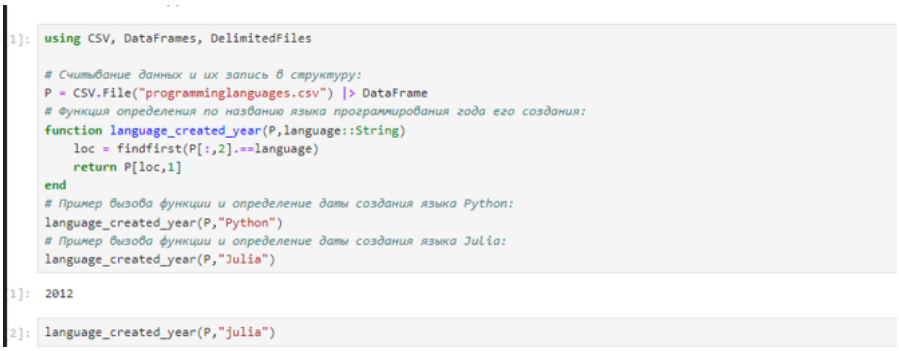


Рис. 1: Повторение примеров_1)

[3]:
```
# 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")
```

[3]:
```
2012
```

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

[4]:
```
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 файл ¶

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

Рис. 2: Повторение примеров_2

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

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

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

[8]:
```
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"
```

Словари

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

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

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

Рис. 3: Повторение примеров_3

[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
```

[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"]
   ⋮   => ⋮
```

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

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

Рис. 4: Повторение примеров_4

DataFrames

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

[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
```

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

[14]: 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

Рис. 5: Повторение примеров_5

RDatasets

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

[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

Рис. 6: Повторение примеров_6

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

[16]: DataFrame

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

[17]: 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}

Работа с переменными отсутствующего типа (Missing Values)

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

[18]: Missing

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

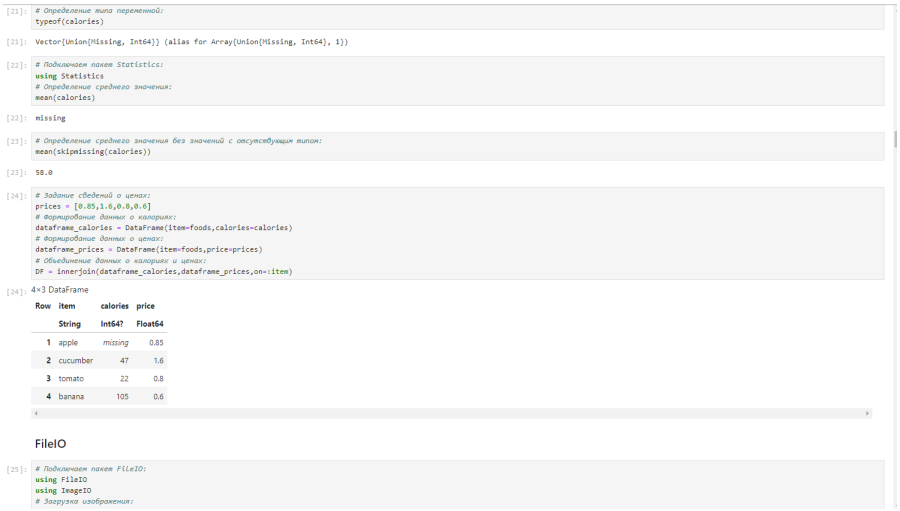
[19]: missing

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

[20]: 4-element Vector{Union{Missing, Int64}}:

```
 missing
  47
  22
 105
```

Рис. 7: Повторение примеров_7

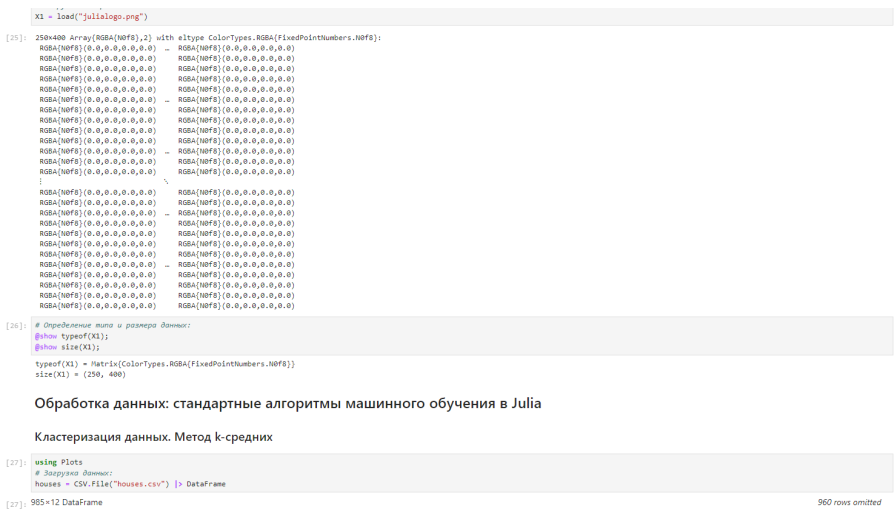


```
[21]: # Определение типа переменной:
      typeof(calories)

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

[22]: # Подключаем пакет Statistics:
      using Statistics
      # Определение среднего значения:
      mean(calories)

[22]: missing

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

[23]: 58.0

[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)

[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

[25]: # Подключаем пакет FileIO:
      using FileIO
      using ImageIO
      # Загрузка изображения:
```

Рис. 8: Повторение примеров_8



```
X1 = load("julialogo.png")

[25]: 250×400 Array{RGBA{N0f8},2} with eltype ColorTypes.RGBA{FixedPointNumbers.N0f8}:
       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)
       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)
       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)

[26]: # Определение типа и размера данных:
      @show typeof(X1);
      @show size(X1);

      typeof(X1) = Matrix{ColorTypes.RGBA{FixedPointNumbers.N0f8}}
      size(X1) = (250, 400)
```

Обработка данных: стандартные алгоритмы машинного обучения в Julia

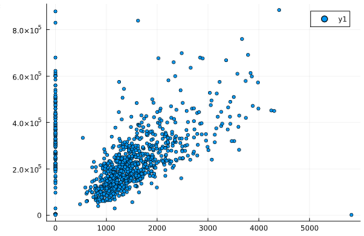
Кластеризация данных. Метод k-средних

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

[27]: 985×12 DataFrame                                                    960 rows omitted
```

Рис. 9: Повторение примеров_9

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

[28]:



[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)
```

[29]:

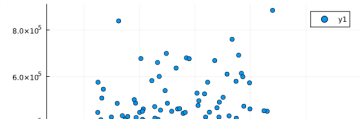


Рис. 10: Повторение примеров_10

[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
```

[31]:
```
combine(filter_houses->mean(filter_houses[!,:price]), groupby(filter_houses,:type))
```

[31]: 3×2 DataFrame

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

[32]:
```
using Clustering
# Добавление данных :Latitude и :Longitude в новый фрейм:
X = filter_houses[:, [:latitude,:longitude]]
```

[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

Рис. 11: Повторение примеров_11

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

```
[33]: 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
 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
 38.7008  -121.257
 38.417   -121.397
 38.6552  -121.076
```

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

```
[34]: KmeansResult{Matrix{Float64}, Float64, Int64}([38.47975119354839 38.737452 … 38.63562266666667 38.60819775; -121.41092403225811 -120.910963 … -121.33890555555556 -121.371695125], [41, 1, 41, 41, 26, 30, 19, 36, 58, 27  …  39, 20, 24, 17, 13, 35, 24, 4, 29, 62], [0.0001203113903607893 74, 0.00004089065211347817, 0.0002828492943081073 5, 0.0003708190743054729, 0.0001553982801851816 5, 0.00014710579853272066, 0.0002775993452814 8733, 0.0002675153446034528 3, 1.1448970326455 31e-5, 0.0003407085860089 8646  …  5.7846707932185 38e-5, 5.2413255616556 85e-5, 0.0002126721228705719, 0.0001633529073332 7701, 0.0007778412618790753, 7.2931776230689 14e-5, 0.0004220604350848589, 0.0003248968314437 6613, 0.0003879064279317 39, 0.0001105941701098345 2], [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, true)
```

Рис. 12: Повторение примеров_12

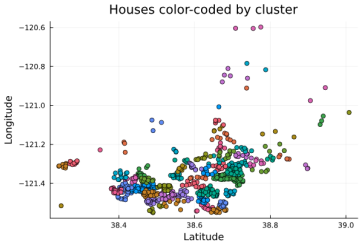
```
[35]: # Формирование фрейма данных:
df = DataFrame(cluster = C.assignments,city = filter_houses[!,:city], latitude = filter_houses[!,:latitude],longitude = filter_houses[!,:longitude],zip = filter_houses[!,:zip])
```

[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

Рис. 13: Повторение примеров_13

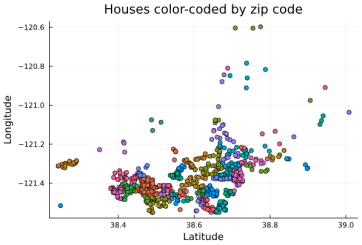
[36]:
```
clusters_figure = plot(legend = false)
for i = 1:k
clustered_houses = df[df[!,:cluster].== i,:]
xvals = clustered_houses[!,:latitude]
yvals = clustered_houses[!,:longitude]
scatter!(clusters_figure,xvals,yvals,markersize=4)
end
xlabel!("Latitude")
ylabel!("Longitude")
title!("Houses color-coded by cluster")
display(clusters_figure)
```



[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")
```

Рис. 14: Повторение примеров_14

```
title!("Houses color-coded by zip code")
display(zips_figure)
```

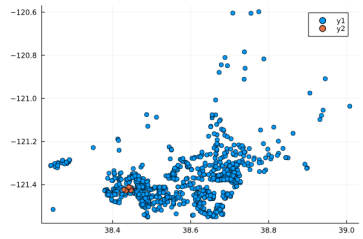


Кластеризация данных. Метод k ближайших соседей

[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)
```

Рис. 15: Повторение примеров_15

[38]:



[39]:
```
# Фильтрация по районам соседних домов:
cities = filter_houses[idxs,:city]
```

[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

[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
```

Рис. 16: Повторение примеров_16

```
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

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

[41]:

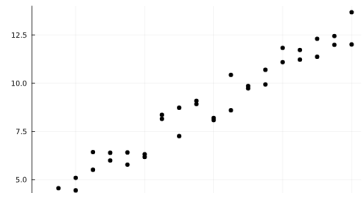
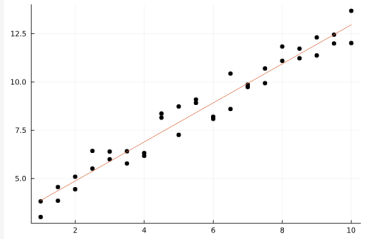


Рис. 17: Повторение примеров_17

```
[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

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

[43]: a,b = find_best_fit(xvals,yvals)
      ynew = a * xvals .+ b
      plot!(xvals,ynew)

[43]:
```




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

Рис. 18: Повторение примеров_18

```
      @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)
[44]: (1.000000044268917, 2.998837311293755)

[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_CompileString), 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}(pyincref(@pycheckn ccall((@
      pysym :PyModule_GetDict), PyPtr, (PyPtr,), pyimport("__main__"))))\n', 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

[46]: using BenchmarkTools
      @btime a,b = find_best_fit_python(xvals,yvals)
      @btime a,b = find_best_fit(xvals,yvals)
```

Рис. 19: Повторение примеров_19

Самостоятельная работа

Выполним задание 7.4.1. Кластеризация: См. рис. 22.

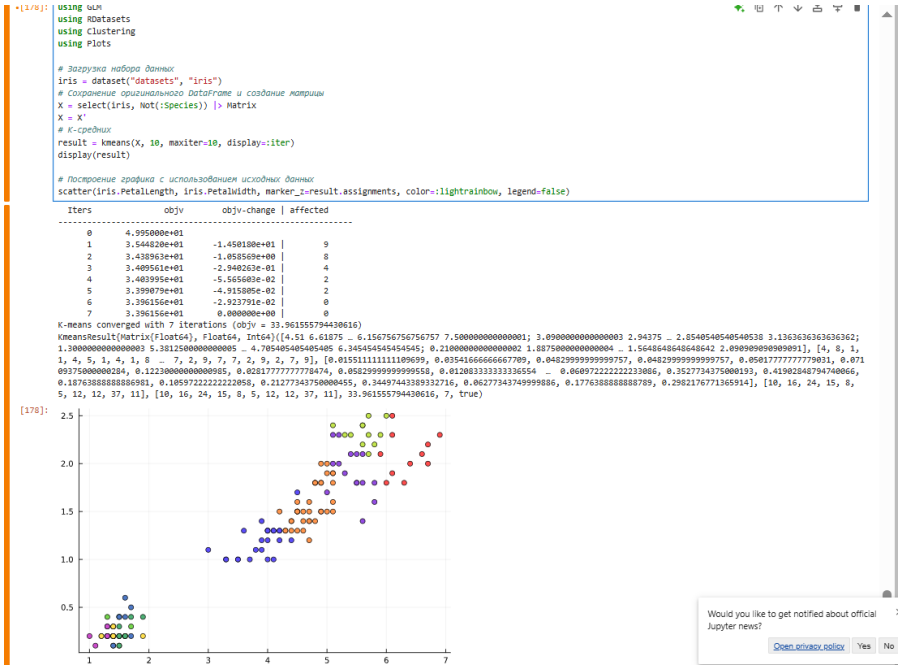


Рис. 1: Самостоятельная работа_1

Была выполнена кластеризация данных из набора iris с использованием языка Julia. Для этого данные были загружены с помощью библиотеки RDatasets, удалён столбец Species, а оставшиеся признаки преобразованы в матрицу и транспонированы для работы с алгоритмом k-средних.

С помощью библиотеки Clustering.jl данные были разделены на 10 кластеров. Результаты кластеризации визуализированы с использованием диаграммы рассеяния, где оси PetalLength и PetalWidth отражают размеры лепестков, а точки окрашены в зависимости

от принадлежности к кластеру. В результате удалось продемонстрировать корректную работу алгоритма и его применение для анализа данных.

7.4.2. Регрессия (метод наименьших квадратов в случае линейной регрессии). См. рис. 23, См. рис. 24, См. рис. 25.

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

```
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
```

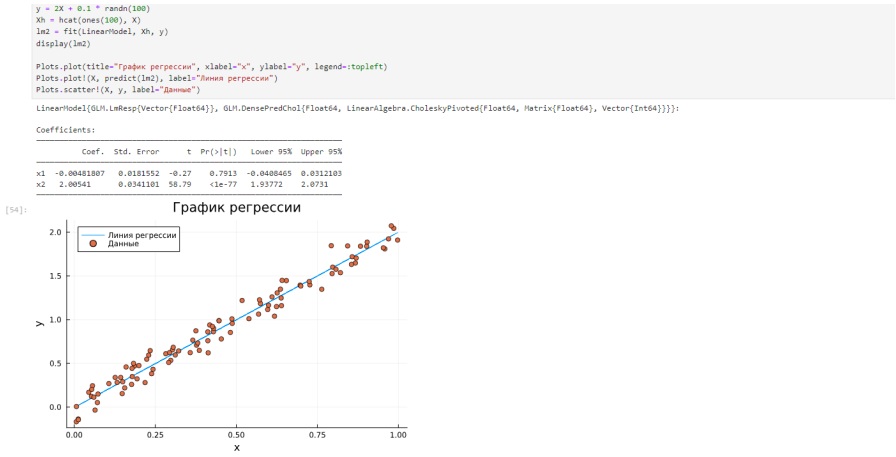
```
N = 1000
X2 = hcat(ones(N), X)
```

```
1000×4 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
```

Рис. 2: Самостоятельная работа_2_1

```
[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
       0.8080809386336673
[51]: 0.09996843006061548

[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
       0.09817586670102467
       0.8079549588733729
[52]: 0.10003581652171566

[53]: X3 = DataFrame(a=y, b=X[:,1], c=X[:,2], d=X[:,3])
      lm15E = lm(@formula(a ~ b + c + d), X3)
      betahat3 = GLM.coeftable(lm15E).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
[53]: 0.09996843006061547

      Часть 2.

[54]: X = rand(100)
      y = 2X + 0.1 * randn(100)
      Xh = hcat(ones(100), X)
```

Рис. 3: Самостоятельная работа_2_2



```
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
──────────────────────────────────────────────────────────────────────────
```

[54]:

Рис. 4: Самостоятельная работа_2_3

Была выполнена линейная регрессия с использованием метода наименьших квадратов. Для этого были сгенерированы случайные данные и построена линейная зависимость между независимой переменной X и зависимой переменной y.

На графике представлена точечная диаграмма исходных данных (оранжевые точки) и линия регрессии (синяя линия), которая показывает оптимальное соответствие между переменными. Линия регрессии демонстрирует сильную положительную зависимость, что подтверждается коэффициентами регрессии, рассчитанными методом наименьших

квадратов. Это визуализирует, как данный метод подходит для анализа линейных связей между переменными.

7.4.3. Модель ценообразования биномиальных опционов. См. рис. 26, См. рис. 27, См. рис. 28.

В этом задании была реализована модель ценообразования биномиальных опционов. Первоначально была написана функция binomial_stock_price, которая рассчитывает траекторию изменения цены акций, используя параметры начальной цены, волатильности, процентной ставки и длины временного периода. Функция возвращает массив цен, где каждая последующая цена зависит от случайного события (рост или падение), что моделируется с помощью случайных чисел.

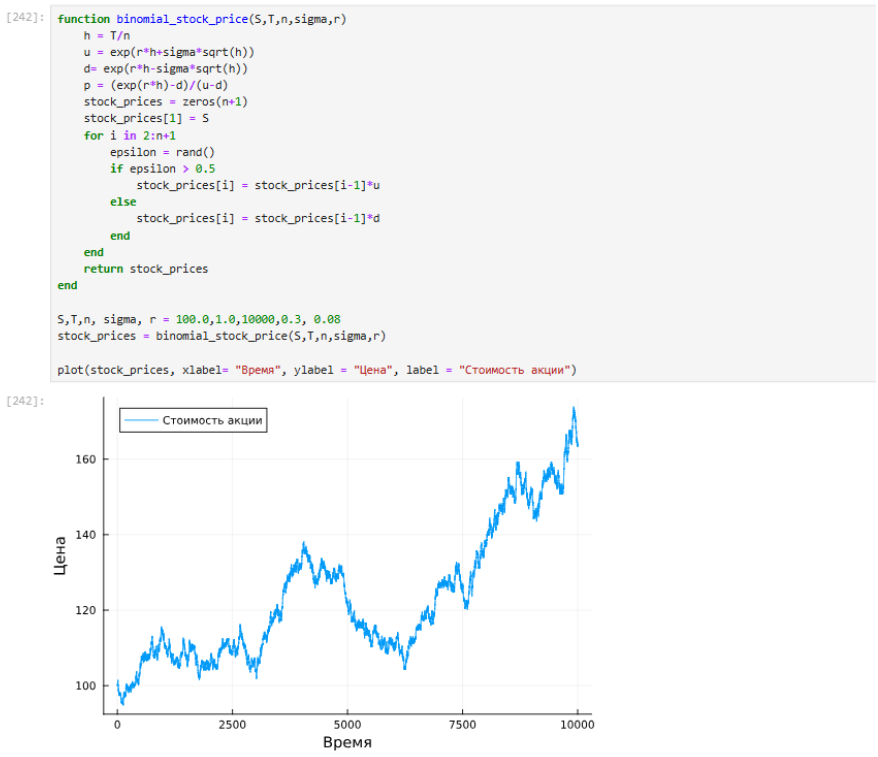


Рис. 5: Самостоятельная работа_3_1

На первом графике показана симуляция одной траектории изменения цены акции. Ось X представляет временные интервалы, а ось Y — стоимость акций. График демонстрирует динамику изменения цены, включая периоды роста и падения.

```
[244]: 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)
```

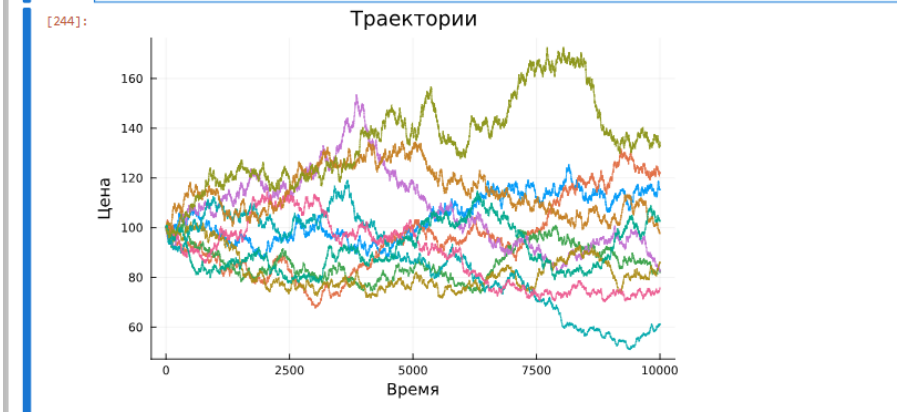


Рис. 6: Самостоятельная работа_3_2

Далее была реализована функция createPath, позволяющая сгенерировать несколько траекторий изменения цен. Результаты были визуализированы на втором графике, где представлены 10 различных траекторий. Каждая линия на графике — это отдельная симуляция изменения стоимости акции, что демонстрирует вариативность и влияние случайности на прогноз.

```
[252]: using Random
       using Base.Threads

       paths2 = []
       @threads for i = 1:10
           push!(paths2, createPath(S,r,sigma,T,n))
       end
       plot(paths2, title = "stocks price paths (threads)", xlabel = "Time", ylabel = "Price", legend = false)
```

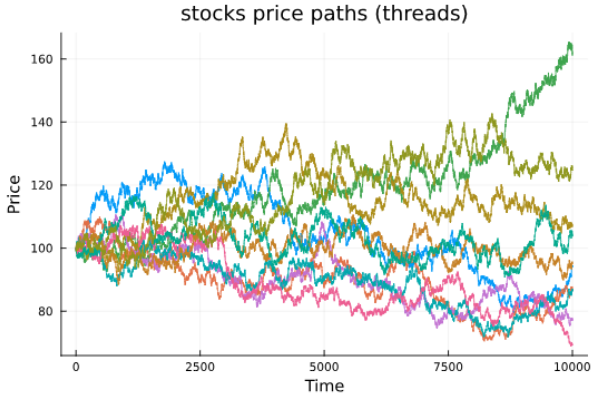


Рис. 7: Самостоятельная работа_3_3

Для оптимизации вычислений была использована параллельная обработка с помощью @threads, что позволило ускорить генерацию траекторий. На третьем графике представлены те же 10 траекторий, но они были сгенерированы с использованием многопоточности, что эффективно сокращает время расчётов при большом количестве симуляций.

Все графики отражают реалистичное поведение цен акций на рынке в условиях случайных изменений, что делает метод полезным для анализа и прогнозирования финансовых данных.

Полученные результаты

1. Считаны и обработаны данные в формате CSV.
2. Реализованы алгоритмы кластеризации методом k-средних и анализа методом главных компонент (PCA).
3. Построены графики для визуализации результатов анализа данных.
4. Выполнен анализ линейной регрессии с использованием больших наборов данных.
5. Полученные результаты подтвердили эффективность использования Julia для обработки и анализа данных.

Вывод

Научились работатать со специализированными пакетами Julia для обработки данных.

Список литературы

[1] Julia: https://ru.wikipedia.org/wiki/Julia