

Лабораторная работа №7

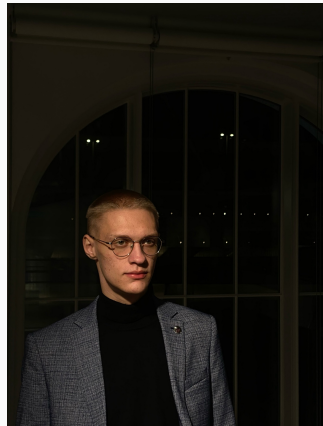
Компьютерный практикум по статистическому анализу данных

Махорин И. С.

2024

Российский университет дружбы народов имени Патриса Лумумбы, Москва, Россия

- Махорин Иван Сергеевич
- Студент группы НПИбд-02-21
- Студ. билет 1032211221
- Российский университет дружбы народов имени Патриса Лумумбы



- Изучить специализированные пакеты Julia для обработки данных.

Выполнение лабораторной работы

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

1. Julia для науки о данных

1.1. Считывание данных

```
[2]: # 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
```

Рис. 1: Установка пакетов

```
[5]: # 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")
```

```
[5]: 1991
```

Рис. 2: Считывание данных и запись в структуру

```
[62]: # Пример вызова функции и определение даты создания языка Julia:  
      language_created_year(P, "Julia")  
  
[62]: 2012
```

Рис. 3: Пример


```
[63]: language_created_year(P, "julia")
```

Рис. 4: Поиск "julia" со строчной буквы

```
[8]: # 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")
```

```
[8]: 2012
```

Рис. 5: Изменение исходной функции

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

```
[9]: 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"
```

Рис. 6: Построчное считывание данных

▼ 1.2. Запись данных в файл

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

Рис. 7: Запись данных в файл

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

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

Рис. 8: Пример с указанием типа данных и разделителем данных

Запись данных в файл

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

[13]: 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"
```

Рис. 9: Проверка корректности считывания созданного текстового файла

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

Рис. 10: Инициализация словаря

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

```
[15]: Dict{Any, Any}()
```

Рис. 11: Инициализация пустого словаря


```
[16]: # 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
```

Рис. 12: Заполнение словаря данными

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

```
[17]: 2-element Vector{String}:  
      "Groovy"  
      "Scala"
```

Рис. 13: Пример работы словаря

1.4. DataFrames

```
[18]: # Подгружаем пакет DataFrames:
      using DataFrames

[20]: # Задаём переменную со структурой DataFrame:
      df = DataFrame(year = P[:,1], language = P[:,2])
      # Вывод всех значения столбца year:
      df[:,year]
      # Получение статистических сведений о фрейме:
      describe(df)
```

[20]: 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. 14: Пример создания структуры DataFrame

1.5. RDatasets

```
[21]: # Подгружаем пакет RDatasets:  
      using RDatasets
```

```
[22]: # Задаём структуру данных в виде набора данных:  
      iris = dataset("datasets", "iris")  
      # Определения типа переменной:  
      typeof(iris)
```

```
[22]: DataFrame
```

Рис. 15: Работа с пакетом RDatasets

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

```
[23]: 5x7 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}

Рис. 16: Получение основных статических сведений о каждом столбце в наборе данных

1.6. Работа с переменными отсутствующего типа (Missing Values)

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

```
[24]: Missing
```

Рис. 17: Использование “отсутствующего” типа

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

```
[25]: missing
```

Рис. 18: Операция сложения числа и переменной с отсутствующим типом

Работа с переменными отсутствующего типа (Missing Values)

```
[26]: # Определение перечня продуктов:  
      foods = ["apple", "cucumber", "tomato", "banana"]  
      # Определение калорий:  
      calories = [missing, 47, 22, 105]  
      # Определение типа переменной:  
      typeof(calories)
```

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

Рис. 19: Пример работы с данными, среди которых есть данные с отсутствующим типом

Работа с переменными отсутствующего типа (Missing Values)

```
[27]: # Подключаем пакет Statistics:  
      using Statistics  
      # Определение среднего значения:  
      mean(calories)  
      # Определение среднего значения без значений с отсутствующим типом:  
      mean(skipmissing(calories))
```

```
[27]: 58.0
```

Рис. 20: Игнорирование отсутствующего типа

Работа с переменными отсутствующего типа (Missing Values)

```
[29]: # 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)
```

[29]: 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. 21: Формирование таблиц данных и их объединение в один фрейм

1.7. FileIO

```
[30]: # Подключаем пакет FileIO:  
      using FileIO
```

```
[31]: # Подключаем пакет ImageIO:  
      import Pkg  
      Pkg.add("ImageIO")
```

Рис. 22: Подключение пакетов

```
[35]: X1 = load("julialogo.png")  
      display(X1)
```

Рис. 23: Загрузка изображения

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

Рис. 24: Определение типа и размера данных

Обработка данных: стандартные алгоритмы машинного обучения в Julia. Кластеризация данных. Метод k-средних

2. Обработка данных: стандартные алгоритмы машинного обучения в Julia

2.1. Кластеризация данных. Метод k-средних

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

Рис. 25: Подключение нужных пакетов

Обработка данных: стандартные алгоритмы машинного обучения в Julia. Кластеризация данных. Метод k-средних

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

[38]: 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 2008	81900	38.5195	-121.436
6	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3	1	1122	Condo	Wed May 21 00:00:00 EDT 2008	89921	38.6626	-121.328
7	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3	2	1104	Residential	Wed May 21 00:00:00 EDT 2008	90895	38.6817	-121.352
8	2561 19TH AVE	SACRAMENTO	95820	CA	3	1	1177	Residential	Wed May 21 00:00:00 EDT 2008	91002	38.5351	-121.481
9	11150 TRINITY RIVER DR Unit 114	RANCHO CORDOVA	95670	CA	2	2	941	Condo	Wed May 21 00:00:00 EDT 2008	94905	38.6212	-121.271
10	7325 10TH ST	RIO LINDA	95673	CA	3	2	1146	Residential	Wed May 21 00:00:00 EDT 2008	98937	38.7009	-121.443
11	645 MORRISON AVE	SACRAMENTO	95838	CA	3	2	909	Residential	Wed May 21 00:00:00 EDT 2008	100309	38.6377	-121.452
12	4085 FAWN CIR	SACRAMENTO	95823	CA	3	2	1289	Residential	Wed May 21 00:00:00 EDT 2008	106250	38.4707	-121.459
13	2930 LA ROSA RD	SACRAMENTO	95815	CA	1	1	871	Residential	Wed May 21 00:00:00 EDT 2008	106852	38.6187	-121.436
:	:	:	:	:	:	:	:	:	:	:	:	:

Рис. 26: Загрузка данных

Обработка данных: стандартные алгоритмы машинного обучения в Julia. Кластеризация данных. Метод k-средних

```
[4]: using CSV
      using DataFrames
      using Plots
      # Загрузка данных из CSV
      houses = CSV.read("houses.csv", DataFrame)
      # Построение графика
      plot(size=(500, 500), leg=false)
      x = houses[:, :sq_ft] # Столбец площади
      y = houses[:, :price] # Столбец цены
      scatter(x, y, markersize=3)
```

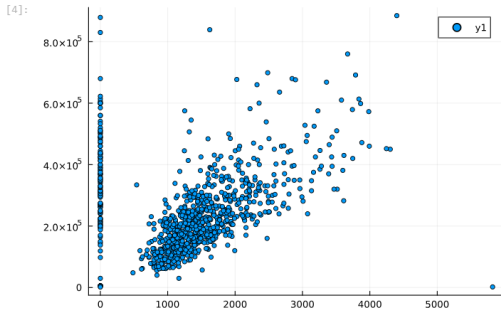


Рис. 27: Построение графика цен на недвижимость в зависимости от площади

Обработка данных: стандартные алгоритмы машинного обучения в Julia. Кластеризация данных. Метод k-средних

```
[5]: # 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)
```

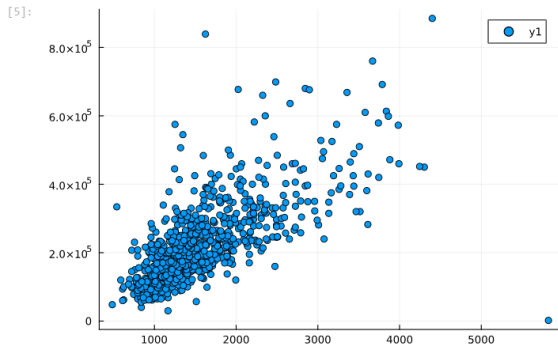


Рис. 28: Построение графика без “артефактов”

Обработка данных: стандартные алгоритмы машинного обучения в Julia. Кластеризация данных. Метод k-средних

Средние цены для каждого типа домов:

4x2 DataFrame

Row	type	mean_price
	String15	Float64
1	Residential	2.39186e5
2	Condo	1.50082e5
3	Multi-Family	2.24535e5
4	Unkown	275000.0

Houses color-coded by cluster

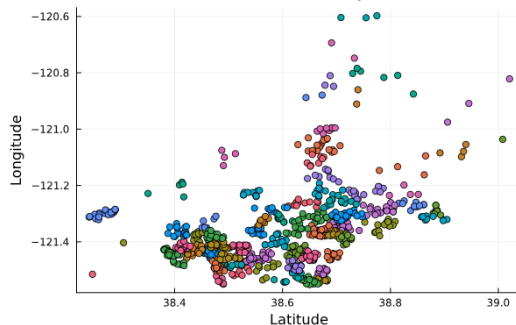


Рис. 29: Построение графика с кластерами разных цветов

Обработка данных: стандартные алгоритмы машинного обучения в Julia. Кластеризация данных. Метод k-средних

```
[15]: 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)
```

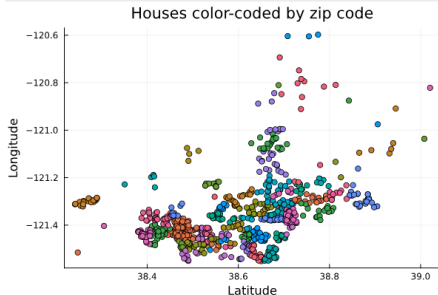


Рис. 30: Построение графика с кластерами разных цветов по почтовому индексу

Кластеризация данных. Метод k ближайших соседей

2.2. Кластеризация данных. Метод k ближайших соседей

```
[16]: # Подключение пакета NearestNeighbors:
import Pkg
Pkg.add("NearestNeighbors")
using NearestNeighbors
knearest = 10
id = 70
point = X[id,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)
```

Resolving package versions...
Updating "C:\Users\Ivan\.julia\environments\v1.10\Project.toml"
[08a86587] * NearestNeighbors v0.4.21
No Changes to "C:\Users\Ivan\.julia\environments\v1.10\Manifest.toml"

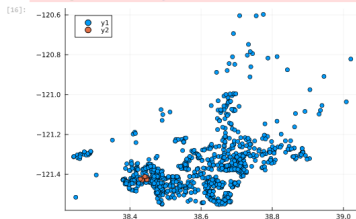


Рис. 31: Отображение на графике соседей выбранного объекта недвижимости

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

[17]: 10-element PooledArrays.PooledVector{String15, UInt32, Vector{UInt32}}:
      "SACRAMENTO"
      "ELK GROVE"
      "SACRAMENTO"
      "SACRAMENTO"
      "SACRAMENTO"
      "SACRAMENTO"
      "ELK GROVE"
      "ELK GROVE"
      "ELK GROVE"
      "ELK GROVE"
```

Рис. 32: Определение районов соседних домов

▼ 2.3. Обработка данных. Метод главных компонент

```
[ ]: # Фрейм с указанием площади и цены недвижимости:  
F = filter_houses[:, :sq__ft, :price]  
# Конвертация данных в массив:  
F = convert(Array{Float64,2}, F)  
# Подключение пакета MultivariateStats:  
import Pkg  
Pkg.add("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, :])
```

Рис. 33: Попытка уменьшения размера данных о цене и площади из набора данных домов

2.4. Обработка данных. Линейная регрессия

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

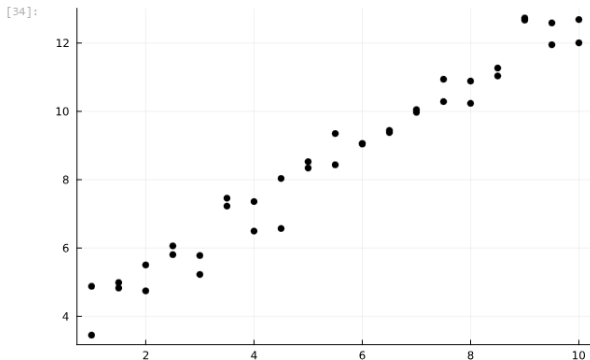


Рис. 34: Исходные данные

```
[36]: 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
a,b = find_best_fit(xvals,yvals)
ynew = a * xvals .* b
plot!(xvals,ynew)
```

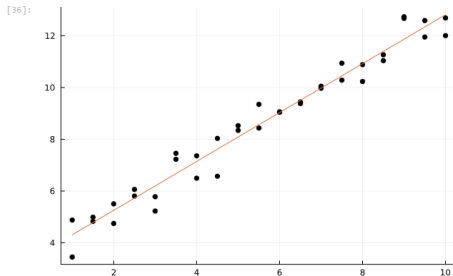


Рис. 35: Применение функции для построения графика


```
[ ]: xvals = 1:100000;
      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)
      import Pkg
      Pkg.add("PyCall")
      Pkg.add("Conda")
      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)
      import Pkg
      Pkg.add("BenchmarkTools")
      using BenchmarkTools
      @btime a,b = find_best_fit_python(xvals,yvals)
      @btime a,b = find_best_fit(xvals,yvals)
```

Рис. 36: Сравнение

▼ Самостоятельное выполнение

1.1) Кластеризация

```
[39]: # Загрузка данных
iris = dataset("datasets", "iris")
# Преобразуем DataFrame в матрицу
X = Matrix(Float64){iris[:, 1:4]} # Используем только числовые признаки
# Применяем кластеризацию методом k-средних
k = 3 # Количество кластеров
result = kmeans(X, k)
# Визуализация кластеров
scatter(X[:, 1], X[:, 2], group = result.assignments, xlabel = "Sepal Length", ylabel = "Sepal Width", title = "K-means Clustering")
```

[39]:

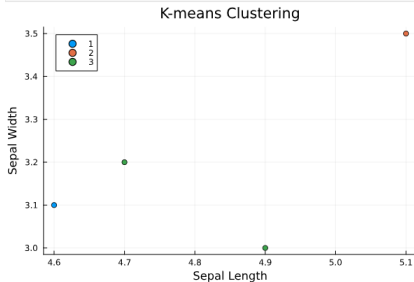


Рис. 37: Решение задания №1

1.2) Регрессия (метод наименьших квадратов в случае линейной регрессии)

▼ Часть 1.

```
[ ]: # Генерация данных
X = randn(1000, 3)
a0 = rand(3)
y = X * a0 + 0.1 * randn(1000)
# Добавляем столбец единиц в X для учета свободного члена
X2 = hcat(ones(1000), X)
# Применяем ридж-регрессию с небольшим значением регуляризации
ridge_result = ridge(X2, y, 1e-4)
println("Ридж-регрессия, коэффициенты:")
println(ridge_result)
# Сравнение с использованием GLM.jl
# Преобразуем X2 в DataFrame, чтобы использовать его с GLM
df = DataFrame(X2 = hcat(ones(1000), X)... , y = y) # Распаковываем X2 в отдельные столбцы
model = lm(@formula(y ~ X2), df)
println("Результаты с использованием GLM.jl:")
println(coef(model))
```

Рис. 38: Решение задания №2

Часть 2.

```
[49]: # Генерация данных
X = rand(100)
y = 2 * X + 0.1 * randn(100)
# Построение графика данных
scatter(X, y, label="Data", xlabel="X", ylabel="y", title="Regression Plot")
# Линейная регрессия
X2 = hcat(ones(100), X) # Добавляем столбец единиц
beta = (X2' * X2) \ (X2' * y)
# Добавление линии регрессии
plot!(X, x -> beta[1] + beta[2] * x, label="Regression Line", color=:red)
```

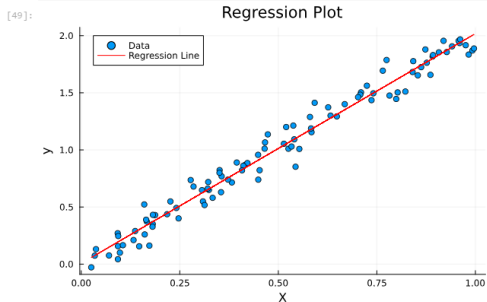


Рис. 39: Решение задания №2

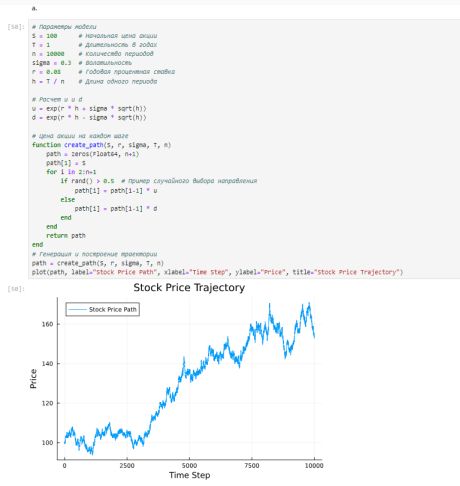


Рис. 40: Решение задания №3

b.

```
# Генерация 10 траекторий
paths = [create_path(S, r, sigma, T, n) for _ in 1:10]
# Построение всех траекторий на одном графике
for path in paths
    plot!(path, label="Trajectory", xlabel="Time Step", ylabel="Price", title="Multiple Stock Price Trajectories")
end
```

Рис. 41: Решение задания №3

с.

```
[ ]: using Threads
# Функция для параллельной генерации траекторий
function parallel_paths(n_paths)
    paths = Vector{Array{Float64, 1}}{undef, n_paths}
    Threads.@threads for i in 1:n_paths
        paths[i] = create_path(S, r, sigma, T, n)
    end
    return paths
end
# Генерация траекторий с использованием многозадачности
paths_parallel = parallel_paths(10)
# Построение всех параллельных траекторий
for path in paths_parallel
    plot!(path, label="Trajectory", xlabel="Time Step", ylabel="Price", title="Parallel Stock Price Trajectories")
end
```

Рис. 42: Решение задания №3

Вывод

- В ходе выполнения лабораторной работы были изучены специализированные пакеты Julia для обработки данных.

Список литературы. Библиография

[1] Julia Documentation: <https://docs.julialang.org/en/v1/>