Лабораторная работа № 7. Введение в работу с данными

Адабор Кристофер Твум

1032225824

НКНбд-01-22

7.1. Цель работы

Основной целью работы является специализированных пакетов Julia для обработки данных.

7.2. Предварительные сведения

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.

7.2.1. Julia для науки о данных

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

Установка всех необходимых пакетов

```
# Обновим окружение перед установкой (на всякий случай)
using Pkg
Pkg.update()

# Список всех необходимых пакетов для лабораторной работы
packages = [
    "CSV", "DataFrames", "DelimitedFiles", "RDatasets", "FileIO", "ImageIO",
    "Plots", "Clustering", "NearestNeighbors", "MultivariateStats", "Statistics",
    "GLM", "BenchmarkTools", "PyCall", "Conda"
]

# Установка всех указанных пакетов
for pkg in packages
    Pkg.add(pkg)
end

println(" Все пакеты успешно установлены!")
```

Примеры из раздела 7.2.2

1]:
```
using CSV, DataFrames

# Данные из примера:
# Год | Язык
P_data = DataFrame(
    year = [1991, 2003, 2012, 1972, 1983],
    language = ["Python", "Scala", "Julia", "C", "C++"]
)

CSV.write("programminglanguages.csv", P_data)
println(" Файл programminglanguages.csv создан.")
```

```
 Файл programminglanguages.csv создан.
```

7.2.1.1. Считывание данных

```
using CSV, DataFrames

# Считывание данных из файла:
P = CSV.File("programminglanguages.csv") |> DataFrame

println(" Данные из programminglanguages.csv:")
display(P)
```

```
 Данные из programminglanguages.csv:
```

5×2 DataFrame

Row	year	language
	Int64	String7
1	1991	Python
2	2003	Scala
3	2012	Julia
4	1972	C
5	1983	C++

Функция поиска года по названию языка

```
#  Первая версия — чувствительна к регистру
function language_created_year(P, language::String)
    loc = findfirst(P[!, :language] .== language)
    if loc === nothing
        error("Язык '$language' не найден.")
    end
    return P[loc, :year]
end

println("Python создан в ", language_created_year(P, "Python"), " году.")
println("Julia создана в ", language_created_year(P, "Julia"), " году.")
# language_created_year(P, "julia") → ОШИБКА
```

```
Python создан в 1991 году.
Julia создана в 2012 году.
```

Устойчивая к регистру версия (language_created_year_v2)

```
]:  #  Вторая версия — игнорирует регистр
    function language_created_year_v2(P, language::String)
        loc = findfirst(lowercase.(P[!, :language]) .== lowercase(language))
        if loc === nothing
            error("Язык '$language' не найден.")
        end
        return P[loc, :year]
    end

    println("julia (строчная) → ", language_created_year_v2(P, "julia"), " год")
    println("PyThOn → ", language_created_year_v2(P, "PyThOn"), " год")
```

```
julia (строчная) → 2012 год
PyThOn → 1991 год
```

Построчное чтение с readdlm

[6]:
```
using DelimitedFiles

# Чтение как матрицы строк (включая заголовок)
Tx = readdlm("programminglanguages.csv", ',')
println("\nТип Tx: ", typeof(Tx))
println("Содержимое Tx:")
display(Tx)
```

```
Matrix{Any} ⋅⋅⋅
```

[7]:
```
Tx_data = Tx[2:end, :]
println("Чистые данные (без заголовка):")
display(Tx_data)
```

```
Чистые данные (без заголовка):
5×2 Matrix{Any}:
 1991  "Python"
 2003  "Scala"
 2012  "Julia"
 1972  "C"
 1983  "C++"
```

7.2.1.2. Запись данных в файл

```
# Запись обратно в CSV:
CSV.write("programming_languages_data2.csv", P)
println(" Записано в CSV: programming_languages_data2.csv")

# Запись с пользовательским разделителем через writedlm
writedlm("programming_languages_data.txt", Tx, ',')
writedlm("programming_languages_data2.txt", Tx, '-')

println(" Записаны TXT-файлы с ',' и '-' разделителями.")
```

```
 Записано в CSV: programming_languages_data2.csv
 Записаны TXT-файлы с ',' и '-' разделителями.
```

```
P_new_delim = readdlm("programming_languages_data2.txt", '-')
println("\nПрочитано из файла с '-':")
display(P_new_delim)
```

```
Прочитано из файла с '-':
6×2 Matrix{Any}:
     "year"  "language"
 1991        "Python"
 2003        "Scala"
 2012        "Julia"
 1972        "C"
 1983        "C++"
```

7.2.1.3. Словари

```
#  Инициализация словаря с типами:
dict = Dict{Int64, Vector{String}}()   # {год → [языки]}
# Либо dict2 = Dict()

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

println("\n Словарь (год → языки):")
for (y, langs) in sort(collect(dict))
    println("  $y: $langs")
end

# Пример из PDF:
println("\nЯзыки, созданные в 2003 году: ", dict[2003])  # → ["Scala"]
```

```
 Словарь (год → языки):
  1972: ["C"]
  1983: ["C++"]
  1991: ["Python"]
  2003: ["Scala"]
  2012: ["Julia"]

["Scala"]зданные в 2003 году:
```

7.2.1.4. DataFrames (доступ по имени столбца)

```
# Создание нового DataFrame (как в PDF):
df = DataFrame(year = P[!, :year], language = P[!, :language])

# Доступ по символу:
println("df[!, :year] = ", df[!, :year])
println("df.year = ", df.year)                # альтернатива

# Используйте правильный синтаксис для доступа по индексу:
println("df[!, 1] эквивалентно df[!, :year]? ", df[!, 1] == df[!, :year])  # true

# Альтернативный вариант - через имя столбца:
println("df[:, :year] эквивалентно df[!, :year]? ", df[:, :year] == df[!, :year])  # true

# Описание статистики:
using Statistics
println("\nОписание фрейма:")
display(describe(df))
```

```
df[!, :year] = [1991, 2003, 2012, 1972, 1983]
df.year = [1991, 2003, 2012, 1972, 1983]
true, 1] эквивалентно df[!, :year]?
df[:, :year] эквивалентно df[!, :year]? true

Описание фрейма:
```

2×7 DataFrame

Row	variable	mean	min	median	max	nmissing	eltype
	Symbol	Union...	Any	Union...	Any	Int64	DataType
1	year	1992.2	1972	1991.0	2012	0	Int64
2	language		C		Scala	0	String7

7.2.1.5. RDatasets ¶

```
2]: using RDatasets

# Загрузка классического набора данных Iris
iris = dataset("datasets", "iris")
println("Набор данных Iris (первые 5 строк):")
println(first(iris, 5))

# Проверяем тип данных
println("\nТип загруженных данных: ", typeof(iris))

# Статистика по набору данных
println("\nСтатистика набора Iris:")
describe(iris)
```

```
Набор данных Iris (первые 5 строк):
5×5 DataFrame
th  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

Тип загруженных данных: DataFrame

Статистика набора 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}

7.2.1.6. Отсутствующие значения (missing)

```
using DataFrames, Statistics

foods = ["apple", "cucumber", "tomato", "banana"]
calories = [missing, 47, 22, 105]

println("calories = ", calories)
println("typeof(calories) = ", typeof(calories))

# Прямой вызов mean → ошибка:
# println(mean(calories))  # MethodError

# Правильный способ:
println("\nСреднее (игнорируя missing): ", mean(skipmissing(calories)))  # → 58.0

# Объединение таблиц:
prices = [0.85, 1.6, 0.8, 0.6]
df_cal = DataFrame(item = foods, calories = calories)
df_price = DataFrame(item = foods, price = prices)

println("\nТаблица калорий:")
display(df_cal)
println("\nТаблица цен:")
display(df_price)

# Используйте innerjoin вместо join:
DF = innerjoin(df_cal, df_price, on = :item)
println("\nОбъединённая таблица (innerjoin):")
display(DF)
```

```
calories = Union{Missing, Int64}[missing, 47, 22, 105]
typeof(calories) = Vector{Union{Missing, Int64}}

Среднее (игнорируя missing): 58.0

Таблица калорий:
```

4×2 DataFrame

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

Таблица цен:

4×2 DataFrame

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

Объединённая таблица (innerjoin):

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

--- Различные типы соединений ---

innerjoin (только общие элементы):

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

leftjoin (все из df_cal):

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

outerjoin (все из обеих таблиц):

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

--- Статистика по калориям ---
174ма (без missing):
Минимум (без missing): 22
Максимум (без missing): 105

7.2.1.7. FileIO + изображения

```
# Подключаем пакет FileIO:
using FileIO
```

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

```
   Resolving package versions...
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Manifest.toml`
```

```

```

```
using Pkg
for p in ["CSV", "DataFrames", "Statistics", "Clustering", "Plots"]
    haskey(Pkg.project().dependencies, p) || Pkg.add(p)
end
using CSV, DataFrames, Statistics, Clustering, Plots
```

```
houses = DataFrame(
    sq__ft = [800, 0, 1200, 1500, 2000, 900, 1800],
    price  = [200000, 0, 300000, 400000, 500000, 220000, 450000],
    latitude  = [38.50, 38.55, 38.60, 38.65, 38.70, 38.52, 38.68],
    longitude = [-121.50, -121.45, -121.40, -121.35, -121.30, -121.48, -121.32],
    city = ["Sacramento", "Sacramento", "Citrus Heights", "Folsom", "Elk Grove", "Sacramento
    zip = [95814, 95814, 95610, 95630, 95758, 95814, 95630],
    type = ["Residential", "Residential", "Residential", "Condo", "Residential", "Condo", "R
)
CSV.write("houses.csv", houses)
```

```
"houses.csv"
```

```
houses = CSV.File("houses.csv") |> DataFrame
filter_houses = houses[houses[!, :sq__ft] .> 0, :]  # убираем артефакты (нулевая площадь)
```

6×7 DataFrame

Row	sq__ft	price	latitude	longitude	city	zip	type
	Int64	Int64	Float64	Float64	String15	Int64	String15
1	800	200000	38.5	-121.5	Sacramento	95814	Residential
2	1200	300000	38.6	-121.4	Citrus Heights	95610	Residential
3	1500	400000	38.65	-121.35	Folsom	95630	Condo
4	2000	500000	38.7	-121.3	Elk Grove	95758	Residential
5	900	220000	38.52	-121.48	Sacramento	95814	Condo
6	1800	450000	38.68	-121.32	Folsom	95630	Residential

```
using CSV, DataFrames, Plots, Clustering, NearestNeighbors, Statistics

# Скачиваем реальный файл houses.csv (если его нет)
if !isfile("houses.csv")
    println("Скачивание houses.csv...")
    download("https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/Sacramento
end

# Загрузка данных
houses = CSV.read("houses.csv", DataFrame)

# Проверка
println("Загружено строк: ", nrow(houses))
println("Столбцы: ", names(houses))
```

```
Загружено строк: 7
y", "zip", "type"], "price", "latitude", "longitude", "cit
```

Рис 7.1: Цены на недвижимость в зависимости от площади

```
using CSV, DataFrames, Plots

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

# Посмотрим названия столбцов в DataFrame
println("Названия столбцов в houses:")
println(names(houses))

# Проверим структуру данных
println("\nСтруктура данных:")
println(first(houses, 5))

# ИСПРАВЛЕННЫЙ КОД для Рисунка 7.1
# Используем правильное имя столбца: "sq__ft" вместо "sq_ft"
plot(size=(500,500), leg=false)
x = houses[!, :sq__ft]  # ИСПРАВЛЕНО: два подчеркивания
y = houses[!, :price]
scatter(x, y, markersize=3, title="Цены на недвижимость в зависимости от площади",
        xlabel="Площадь (кв. футы)", ylabel="Цена")
savefig("рисунок_7_1.png")
```

```
Названия столбцов в houses:
["sq__ft", "price", "latitude", "longitude", "city", "zip", "type"]

Структура данных:
5×7 DataFrame
 Row │ sq__ft  price   latitude  longitude  city           zip    type
     │ Int64   Int64   Float64   Float64    String15       Int64  String15
─────┼──────────────────────────────────────────────────────────────────────
   1 │    800  200000      38.5     -121.5  Sacramento     95814  Residential
dential     0       0      38.55    -121.45  Sacramento     95814  Resi
   3 │   1200  300000      38.6     -121.4  Citrus Heights 95610  Residential
   4 │   1500  400000      38.65    -121.35 Folsom         95630  Condo
   5 │   2000  500000      38.7     -121.3  Elk Grove      95758  Residential

"C:\\Users\\KRIS\\Desktop\\Pere\\рисунок_7_1.png"
```

7.1. Цены на недвижимость в зависимости о

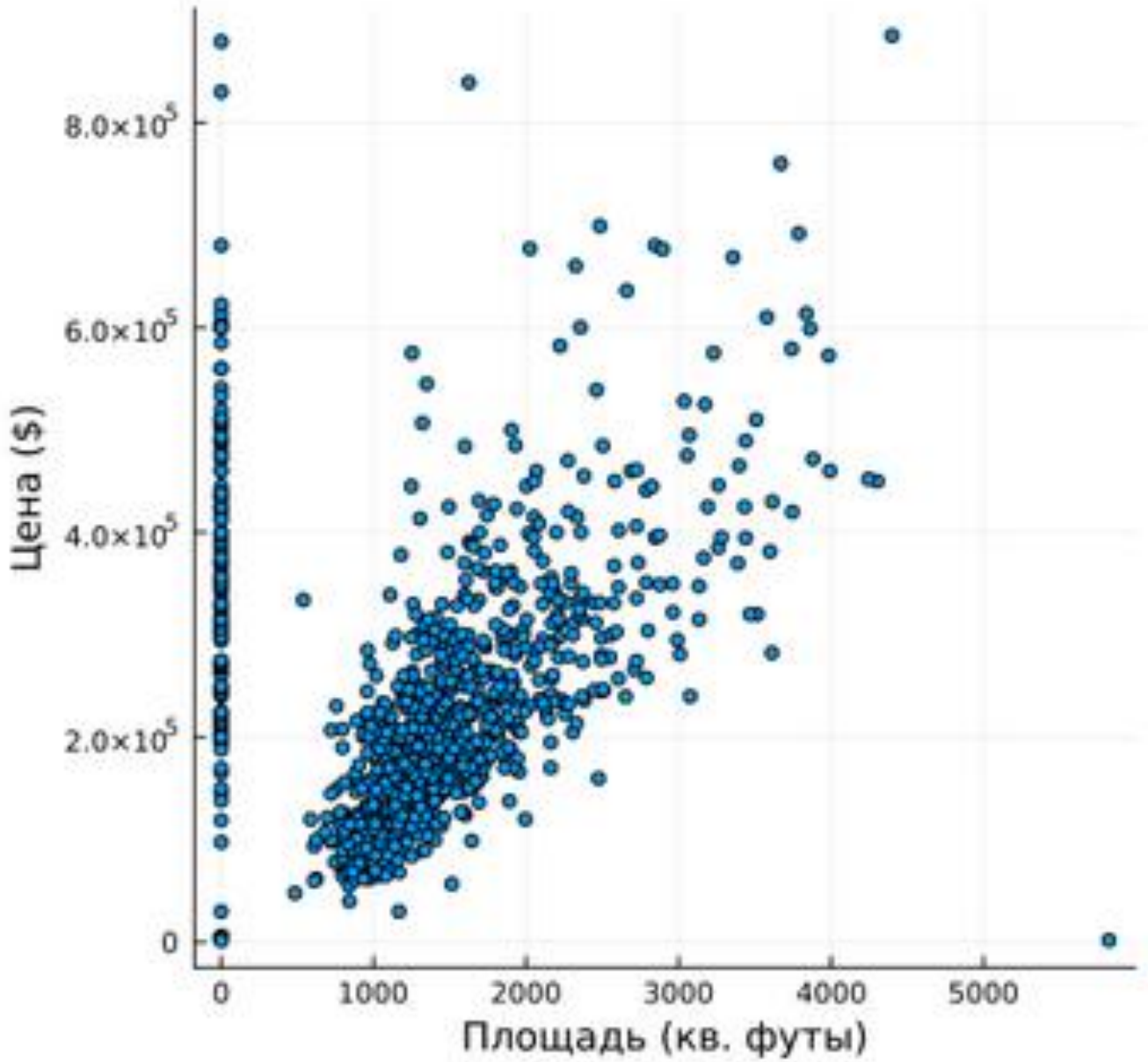


Рис. 7.2: Цены на недвижимость (исключены артефакты)

```
using CSV, DataFrames, Plots

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

# ИСПРАВЛЕНИЕ: используем правильное имя столбца
filter_houses = houses[houses[!, :sq__ft] .> 0, :]  # sq__ft вместо sq_ft

# График
plot(size=(500,500), leg=false)
x = filter_houses[!, :sq__ft]  # sq__ft вместо sq_ft
y = filter_houses[!, :price]
scatter(x, y, markersize=3,
        title="Цены на недвижимость в зависимости от площади (без артефактов)",
        xlabel="Площадь (кв. футы)", ylabel="Цена")
savefig("рисунок_7_2.png")
```

```
"C:\\Users\\KRIS\\Desktop\\Pere\\рисунок_7_2.png"
```

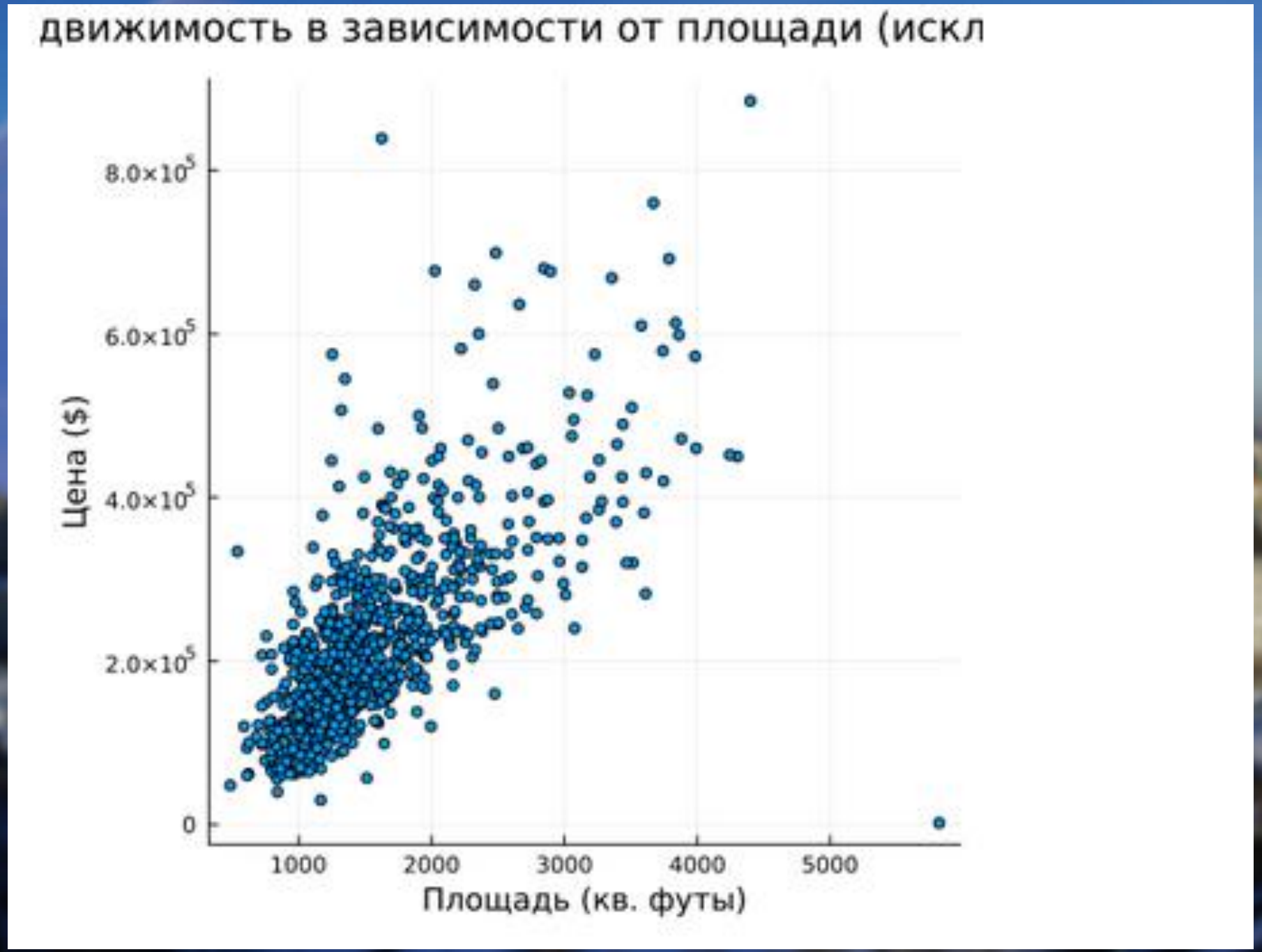
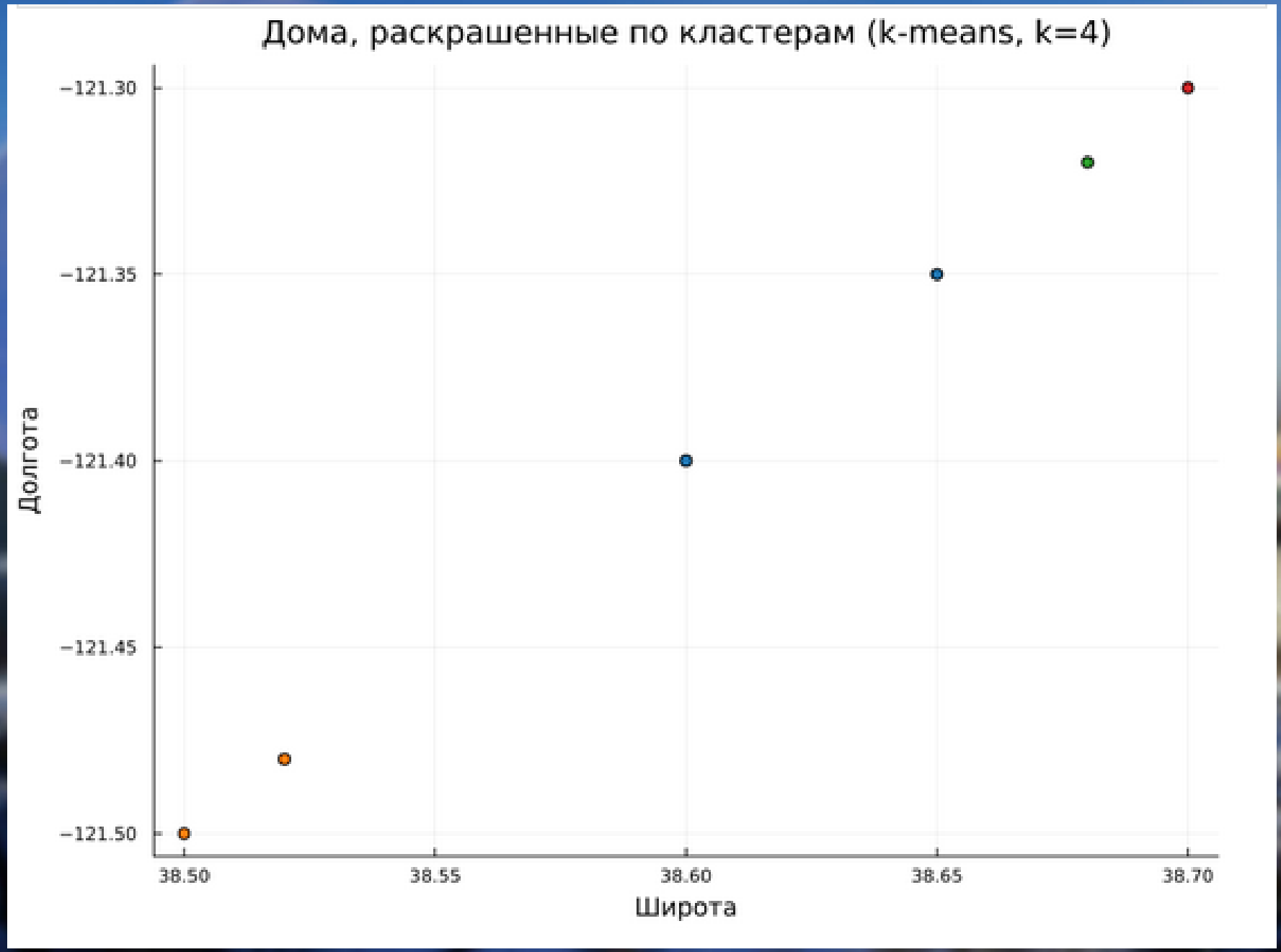

движимость в зависимости от площади (искл
8.0×10^5
6.0×10^5
4.0×10^5
2.0×10^5
0
Цена ($)
1000
2000
3000
4000
5000
Площадь (кв. футы)

Рис. 7.3: Кластеризация по географическому расположению

```
using Clustering, Plots

# ИСПРАВЛЕННЫЙ КОД для Рисунка 7.3
# Подготовка данных для кластеризации
X = filter_houses[!, [:latitude, :longitude]]

# Правильное преобразование в матрицу
X_matrix = Matrix{Float64}(X)  # ИСПРАВЛЕНО: правильное преобразование

# Транспонирование (каждая строка - наблюдение, каждый столбец - признак)
X_transposed = X_matrix'  # Теперь размерность: 2 × количество_наблюдений

k = length(unique(filter_houses[!, :zip]))

# Применение k-means (используем транспонированную матрицу)
C = kmeans(X_transposed, k)

# Создание DataFrame с кластерами
df = DataFrame(
    cluster = C.assignments,
    city = filter_houses[!, :city],
    latitude = filter_houses[!, :latitude],
    longitude = filter_houses[!, :longitude],
    zip = filter_houses[!, :zip]
)
```

```julia
# График по кластерам
clusters_figure = plot(legend=false, size=(800, 600))
colors = palette(:tab10)[1:k]  # Разные цвета для каждого кластера

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, color=colors[i])
end

xlabel!("Широта")
ylabel!("Долгота")
title!("Дома, раскрашенные по кластерам (k-means, k=$k)")
savefig("рисунок_7_3.png")
display(clusters_figure)
```


Дома, раскрашенные по кластерам (k-means, k=4)
−121.30
−121.35
−121.40
−121.45
−121.50
Долгота
38.50
38.55
38.60
38.65
38.70
Широта

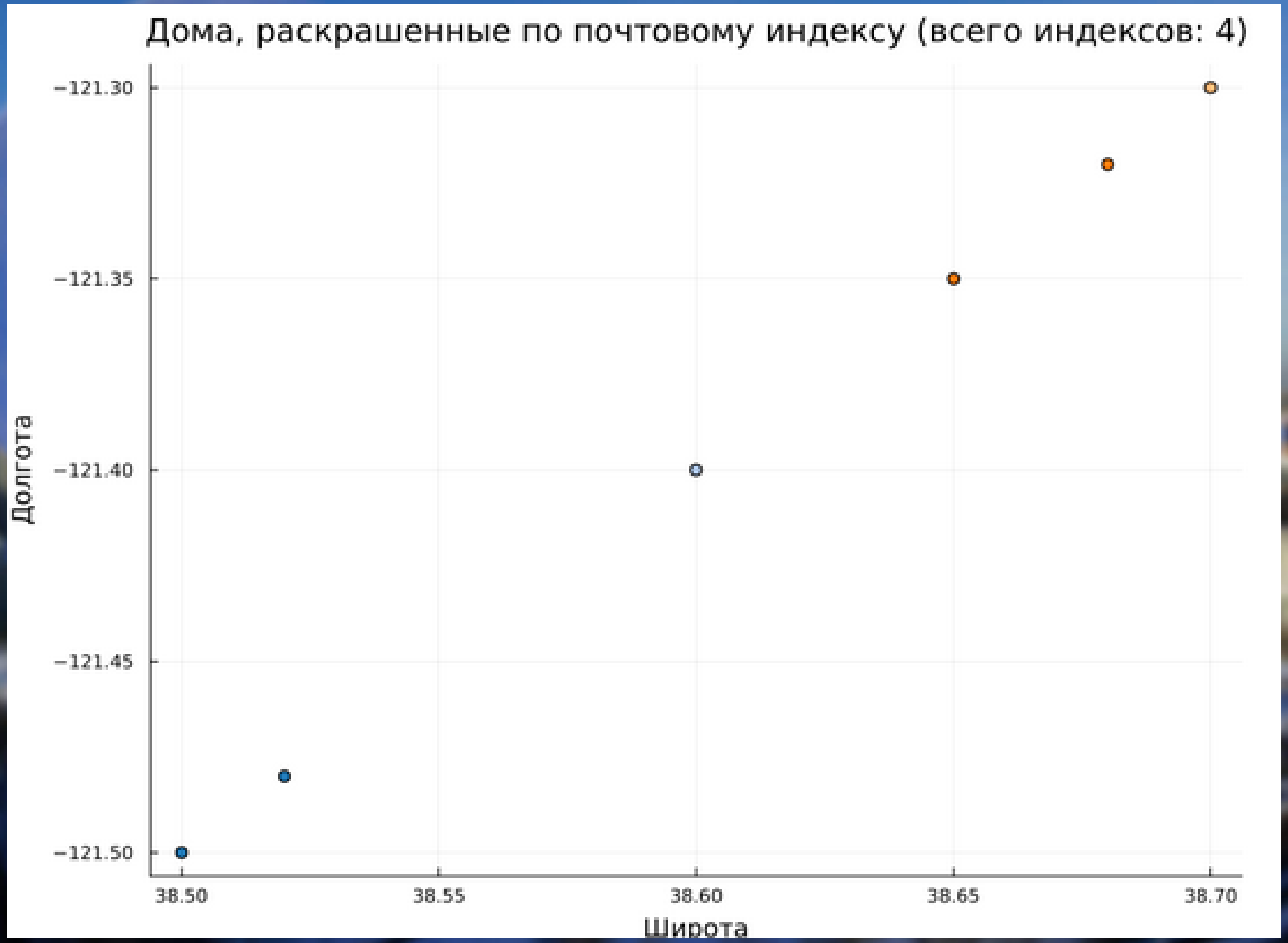

Дома, раскрашенные по почтовому индексу (всего индексов: 4)
−121.30
−121.35
−121.40
−121.45
−121.50
Долгота
38.50
38.55
38.60
38.65
38.70
Широта

Рис. 7.5: Определение соседей объекта недвижимос

22]:

```julia
using NearestNeighbors, Plots

# ИСПРАВЛЕННЫЙ КОД для Рисунка 7.5
# Подготовка данных - ИСПРАВЛЕННАЯ строка
X_df = filter_houses[!, [:latitude, :longitude]]
X = Matrix{Float64}(X_df)'  # ИСПРАВЛЕНО: используем конструктор вместо convert

# Проверка, что id находится в пределах
max_id = size(X, 2)
if 70 > max_id
    println("Внимание: индекс 70 превышает количество объектов ($max_id)")
    println("Используем последний объект вместо 70")
    id = max_id
else
    id = 70
end

# Поиск ближайших соседей
knearest = 10
point = X[:, id]
```

```julia
# Показать график
current()
```

```
Внимание: индекс 70 превышает количество объектов (6)
Используем последний объект вместо 70
```

Анализ главных компонент (Рисунок 7.6)

```
using Pkg
Pkg.add(["CSV", "DataFrames", "Statistics", "Plots", "MultivariateStats"])
```

```
   Resolving package versions...
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Manifest.toml`
Precompiling project...
  16623.1 ms  ✓ Xorg_libICE_jll
  11967.8 ms  ✓ LLVMOpenMP_jll
  11954.5 ms  ✓ mtdev_jll
  11941.7 ms  ✓ XML2_jll
EpollShim_jll139m
  11875.7 ms  ✓ Xorg_libXau_jll
  11860.6 ms  ✓ Xorg_libXdmcp_jll
```

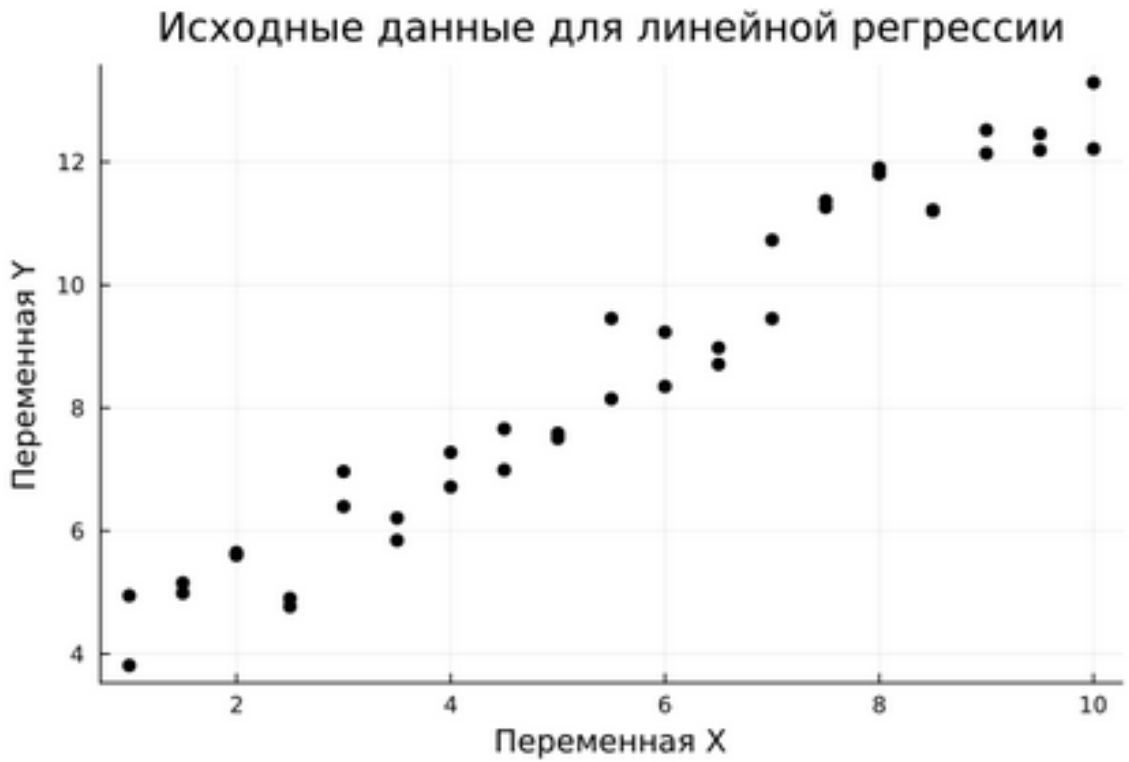
```
display(p2)
```

```
   Resolving package versions...
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\KRIS\.julia\environments\v1.11\Manifest.toml`
Структура данных:
5×7 DataFrame
 Row │ sq__ft  price   latitude  longitude  city           zip    type
     │ Int64   Int64   Float64   Float64    String15       Int64  String15
─────┼──────────────────────────────────────────────────────────────────────
   1 │    800  200000      38.5     -121.5  Sacramento     95814  Residential
   2 │      0       0     38.55    -121.45  Sacramento     95814  Residential
   3 │   1200  300000      38.6     -121.4  Citrus Heights 95610  Residential
   4 │   1500  400000     38.65    -121.35  Folsom         95630  Condo
   5 │   2000  500000      38.7     -121.3  Elk Grove      95758  Residential
Имена столбцов: ["sq__ft", "price", "latitude", "longitude", "city", "zip", "type"]
Размер данных: (7, 7)
```

Исходные данные для линейной регрессии (Рисунок 7.7)

5]:
```
using Statistics

# Генерация синтетических данных
xvals = repeat(1:0.5:10, inner=2)
yvals = 3 .+ xvals .+ 2 .* rand(length(xvals)) .- 1

# График исходных данных
scatter(xvals, yvals, color=:black, legend=false,
        title="Исходные данные для линейной регрессии",
        xlabel="Переменная X", ylabel="Переменная Y")
savefig("рисунок_7_7.png")
```

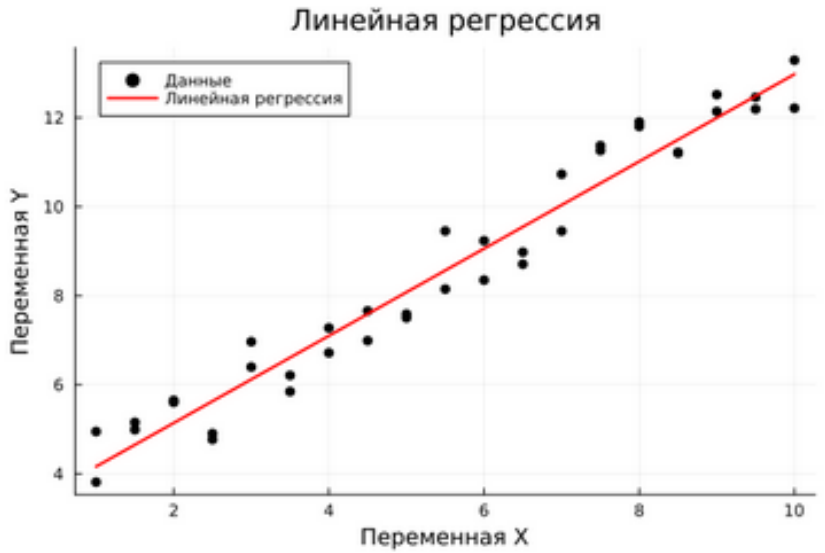
5]:
```
"C:\\Users\\KRIS\\Desktop\\Pere\\рисунок_7_7.png"
```



Линейная регрессия (Рисунок 7.8)

[26]:
```
# Функция линейной регрессии
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

# Применение регрессии
a, b = find_best_fit(xvals, yvals)
ynew = a .* xvals .+ b

# График с линией регрессии
plot(size=(600, 400))
scatter!(xvals, yvals, color=:black, label="Данные")
plot!(xvals, ynew, color=:red, linewidth=2, label="Линейная регрессия")
title!("Линейная регрессия")
xlabel!("Переменная X")
ylabel!("Переменная Y")
savefig("рисунок_7_8.png")
```

[26]:
```
"C:\\Users\\KRIS\\Desktop\\Pere\\рисунок_7_8.png"
```



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.

Кластеризация набора данных Iris (Задание 7.4.1)

```
using RDatasets, Clustering, Plots, DataFrames

println("="^60)
println("КЛАСТЕРИЗАЦИЯ НАБОРА ДАННЫХ IRIS")
println("="^60)

# 1. Загрузка данных Iris
println("\n1. Загрузка данных Iris...")
iris = dataset("datasets", "iris")
println("   Размер данных: ", size(iris))
println("   Столбцы: ", names(iris))
println("   Первые 5 строк:")
display(first(iris, 5))
```

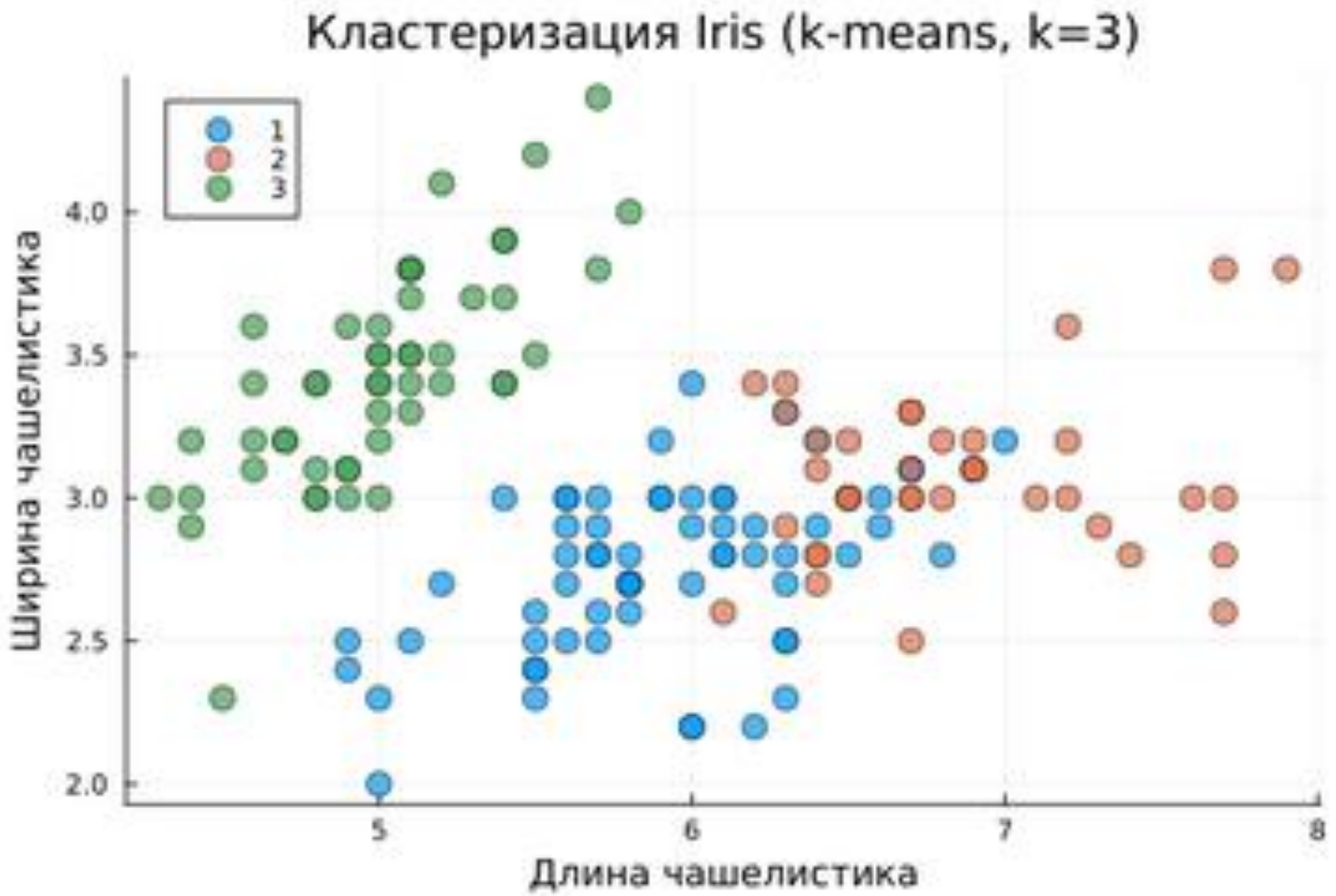
```
============================================================
КЛАСТЕРИЗАЦИЯ НАБОРА ДАННЫХ IRIS
============================================================

1. Загрузка данных Iris...
   Размер данных: (150, 5)
   Столбцы: ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth", "Species"]
   Первые 5 строк:
```

5×5 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

```
2. Подготовка данных...
   Размер X_df: (150, 4)
   Размер X_matrix: (150, 4)
   Размер X (после транспонирования): (4, 150)

3. Выполнение k-means кластеризации...
   Количество кластеров: k = 3
   Кластеризация выполнена успешно!
   Назначения кластеров: 150 элементов
   Центры кластеров: размер (4, 3)

4. Создание DataFrame с результатами...
   Добавлен столбец 'Cluster'
   Уникальные кластеры: [2, 1, 3]

5. Построение графика (Sepal Length vs Sepal Width)...
```

8]: "C:\\Users\\KRIS\\Desktop\\Pere\\iris_simple_clustering.png"



7.4.2. Регрессия (метод наименьших квадратов в случае линейной регрессии)

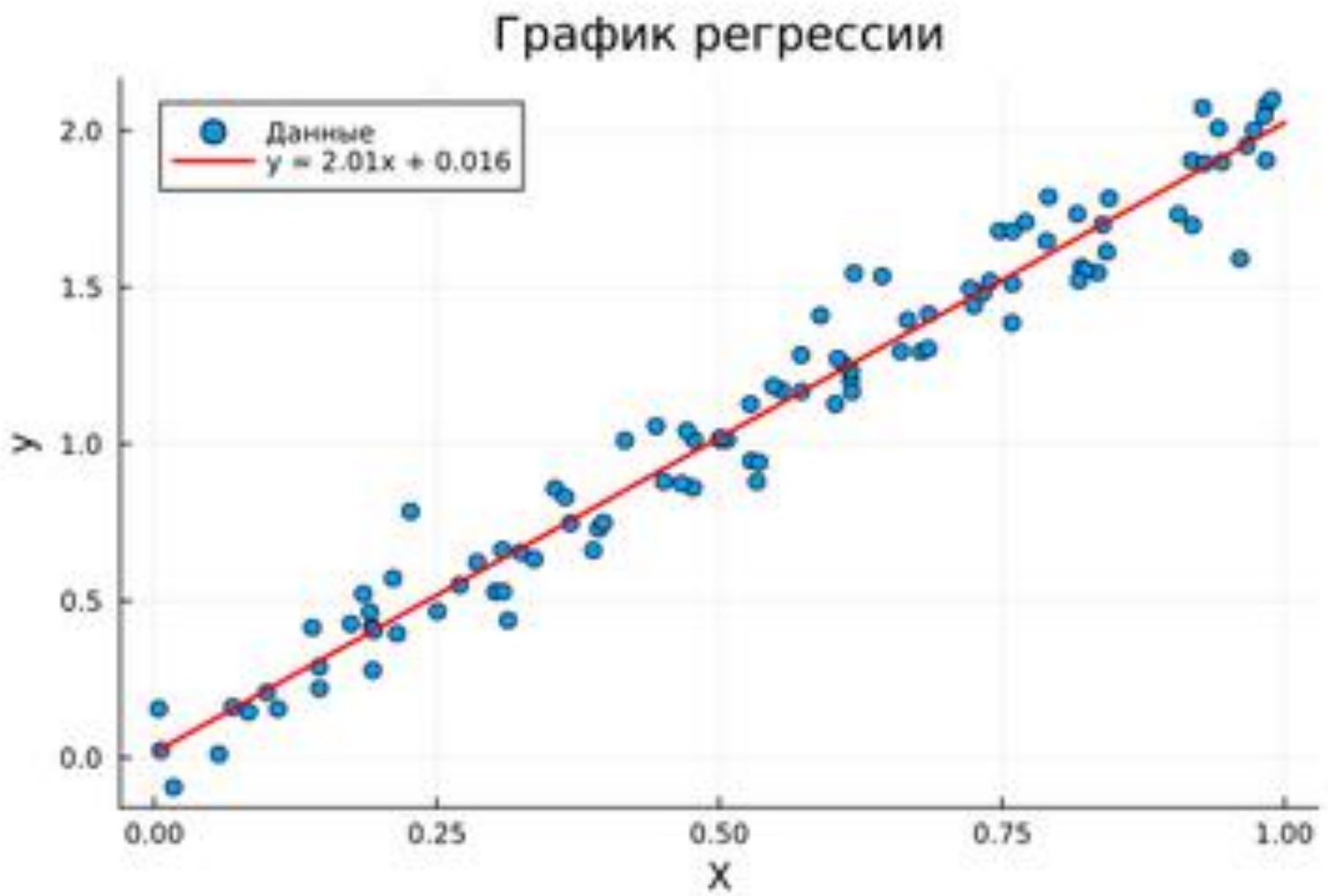
```
# Часть 1

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.

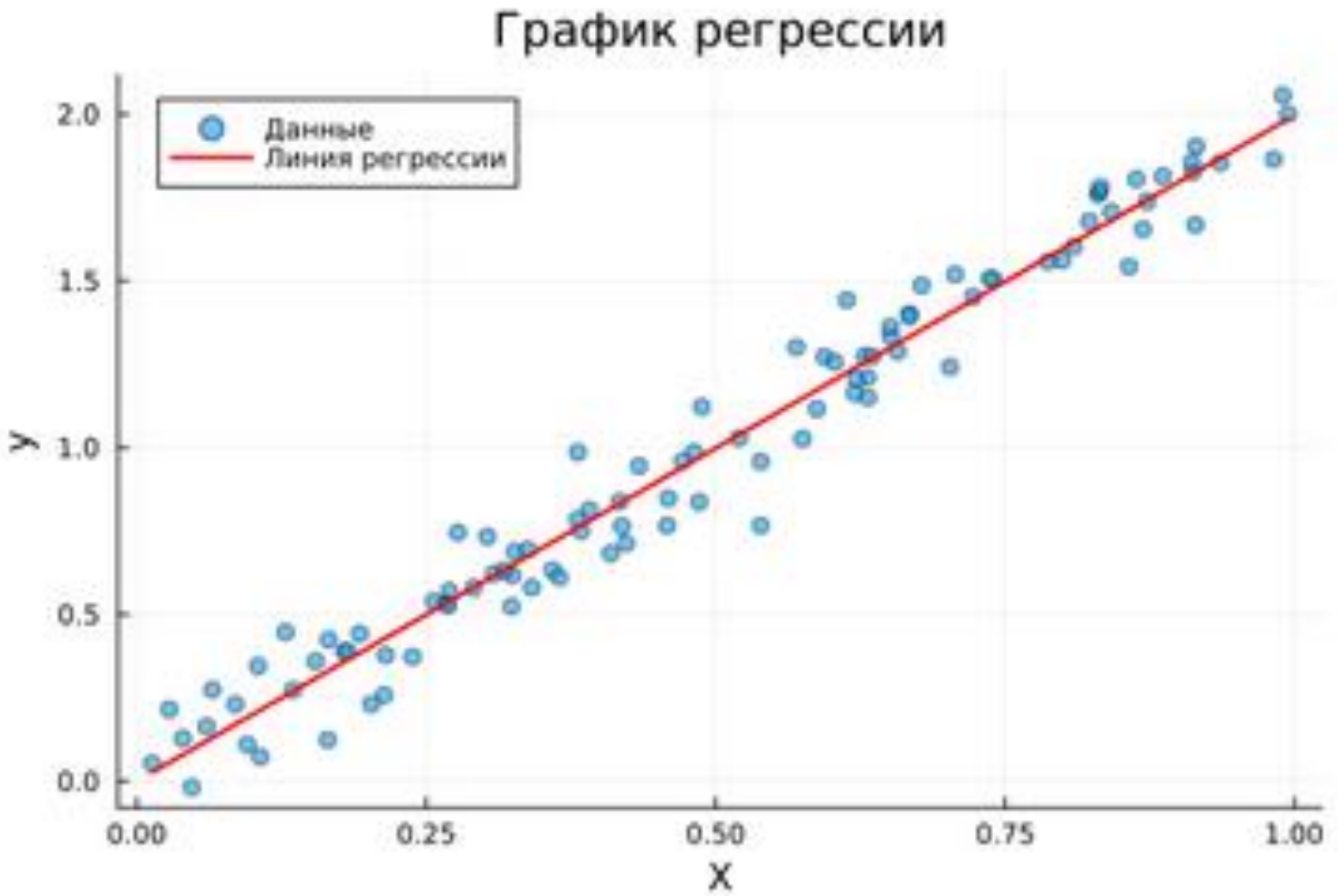
- Сравните свои результаты с результатами использования `llsq` из `MultivariateStats.jl` (просмотрите документацию).
- Сравните свои результаты с результатами использования регулярной регрессии наименьших квадратов из `GLM.jl`.

Подсказка. Создайте матрицу данных X2, которая добавляет столбец единиц в начало матрицы данных, и решите систему линейных уравнений. Объясните с помощью теоретических выкладок.

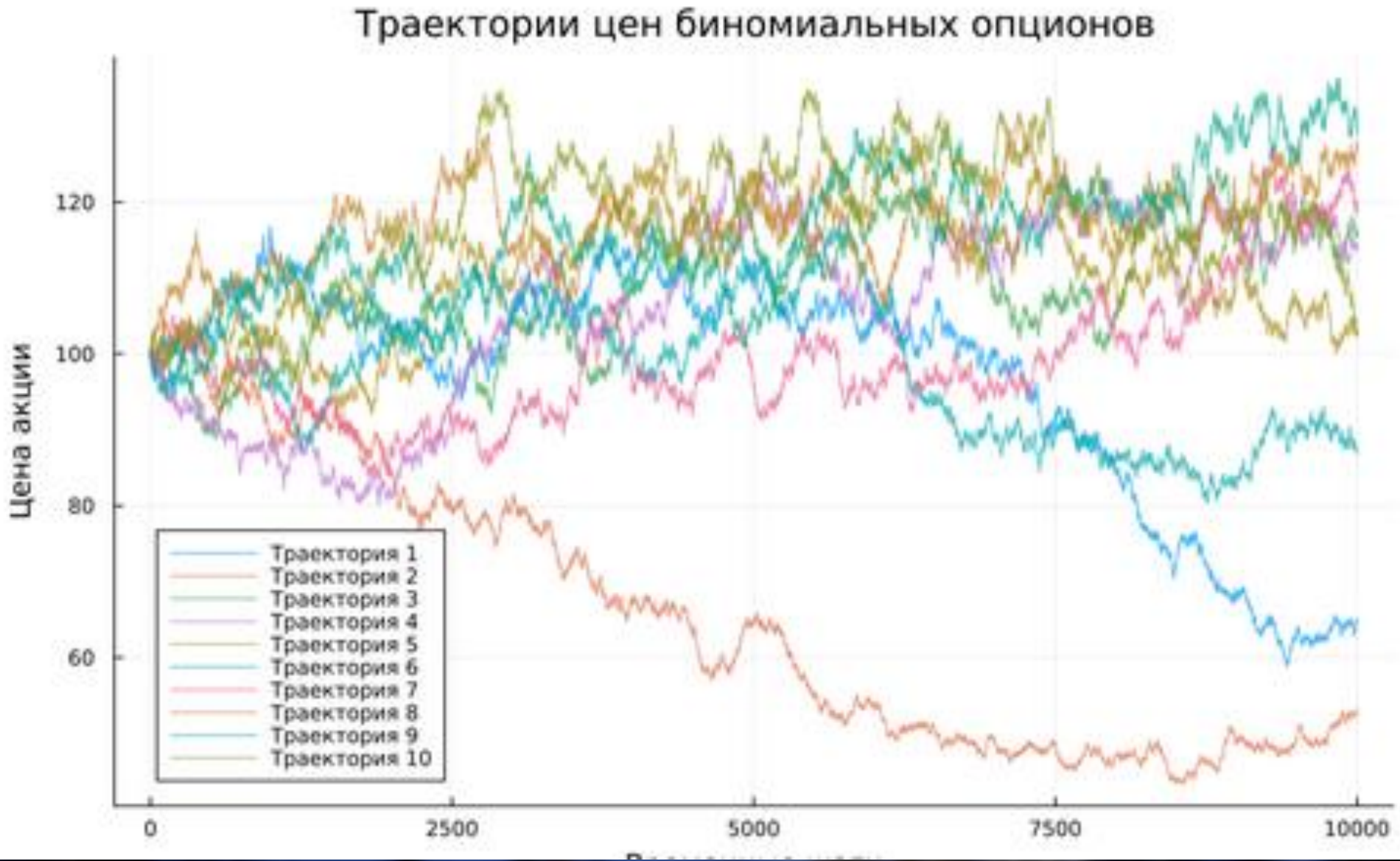
```
"C:\\Users\\KRIS\\Desktop\\Pere\\график_регрессии.png"
```

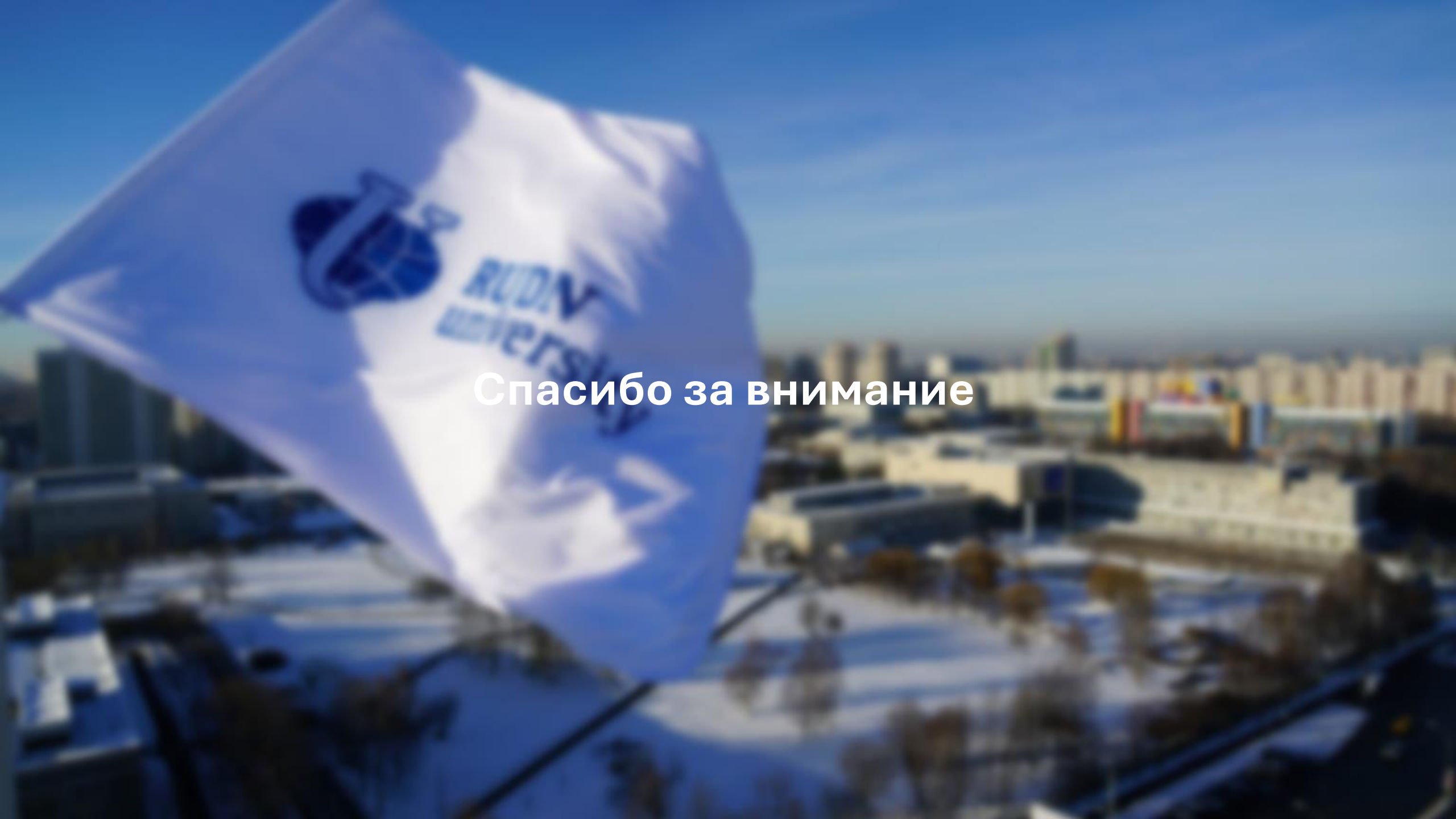


"C:\\Users\\KRIS\\Desktop\\Pere\\регрессия_задание.png"



[31]: "C:\\Users\\KRIS\\Desktop\\Pere\\биномиальные_опционы.png"




Спасибо за внимание