Компьютерный практикум по статистическому анализу данных

Лабораторная работа № 7. Введение в работу с данными

Демидова Екатерина Алексеевна

Содержание

1 Введение 4

2 Теоретическое введение 5

3 Выполнение лабораторной работы 6

4 Выводы 20

Список литературы 21

Список иллюстраций

3.1 Примеры 6
3.2 Примеры 7
3.3 Примеры 8
3.4 Примеры 9
3.5 Примеры 10
3.6 Примеры 11
3.7 Примеры 12
3.8 Примеры 13
3.9 Задание 1 14
3.10 Задание 1 15
3.11 Задания 2 15
3.12 Задание 2 16
3.13 Задание 3 17
3.14 Задание 3 18
3.15 Задание 3 19

1 Введение

Цель работы

Основной целью работы является освоение специализированных пакетов Julia для обработки данных.

Задачи

1. Используя Jupyter Lab, повторите примеры.
2. Выполните задания для самостоятельной работы.

2 Теоретическое введение

Julia — высокоуровневый свободный язык программирования с динамической типизацией, созданный для математических вычислений[1]. Эффективен также и для написания программ общего назначения. Синтаксис языка схож с синтаксисом других математических языков, однако имеет некоторые существенные отличия.

Для выполнения заданий была использована официальная документация Julia[2].

3 Выполнение лабораторной работы

Выполним примеры из лабораторной работы для изучения циклов и функций(рис. 3.1 - 3.8)

```
[198] using CSV, DataFrames, DelimitedFiles, Plots, Statistics, GLM
```

Считывание данных

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

# Функция определения по названию языка программирования года его создания:
function language_created_year(P,language::String)
loc = findfirst(P[:,2].==language)
return P[loc,1]
end
# Пример вызова функции и определение даты создания языка Python:
language_created_year(P,"Python")
# Пример вызова функции и определение даты создания языка Julia:
language_created_year(P,"Julia")

# 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

language_created_year_v2(P,"julia")

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

Запись данных в файл

```
# Запись данных в CSV-файл:
CSV.write("programming_languages_data2.csv", P)
Можно задать тип файла и разделитель данных:
# Пример записи данных в текстовый файл с разделителем ',':
writedlm("programming_languages_data.txt", Tx, ',')
# Пример записи данных в текстовый файл с разделителем '-':
writedlm("programming_languages_data2.txt", Tx, '-')

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

Рис. 3.1: Примеры

Словари

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

# Инициализация словаря:
dict2 = Dict()

# 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]
```

DataFrames

```
# Подгружаем пакет DataFrames:
using DataFrames
# 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)
```

Рис. 3.2: Примеры

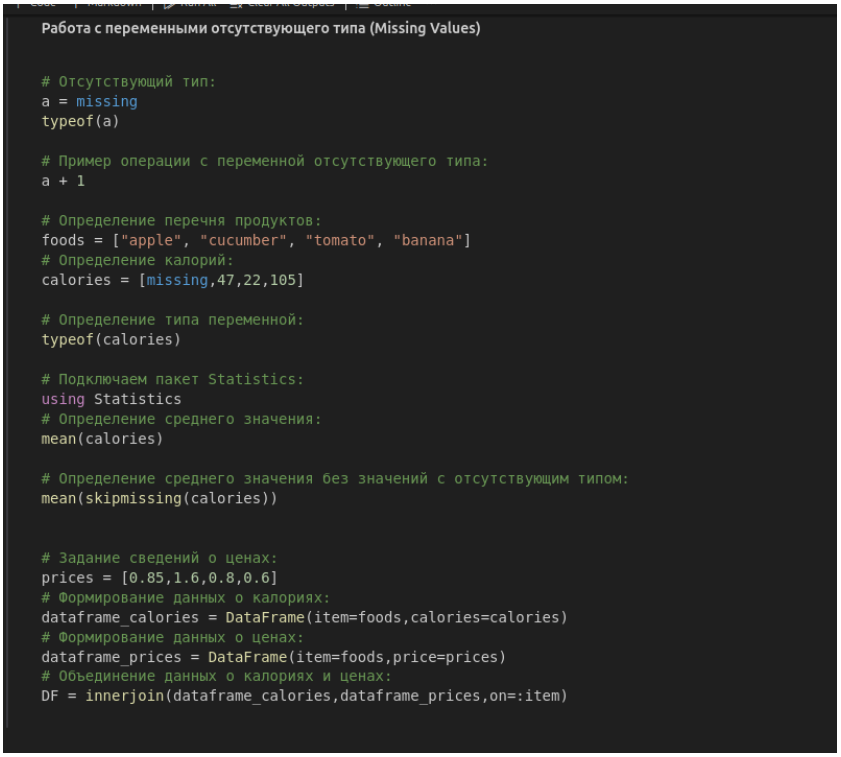


Рис. 3.3: Примеры

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

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

# 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)

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

using Clustering
# Добавление данных :latitude и :longitude в новый фрейм:
X = filter_houses[!,[:latitude,:longitude]]
# Конвертация данных в матричный вид:
X = Matrix(X)

# Транспонирование матрицы с данными:
X = X'

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

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

# 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")
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. 3.4: Примеры

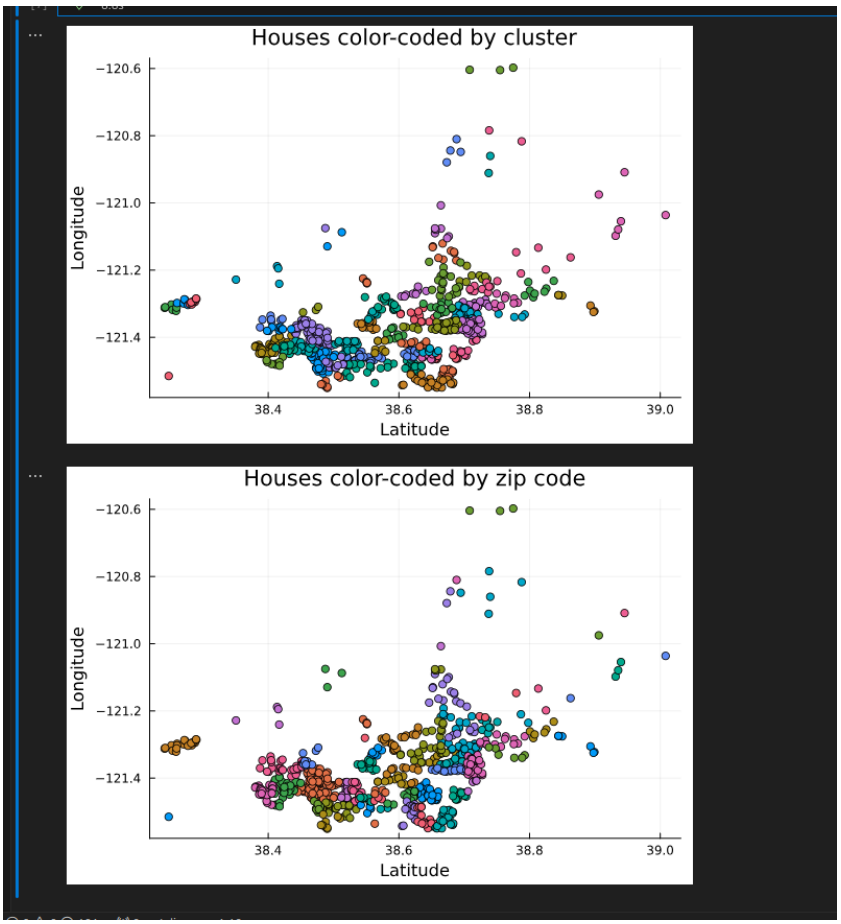


Рис. 3.5: Примеры

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

[30] ✓ 0.0s

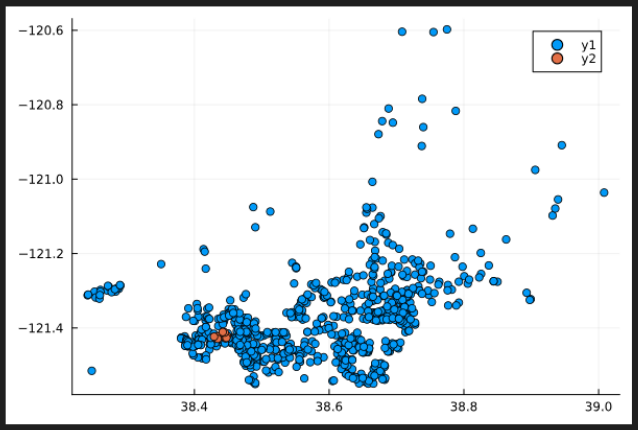


Рис. 3.6: Примеры

```
# Фрейм с указанием площади и цены недвижимости:
F = filter_houses[!,[:sq__ft,:price]]
y = filter_houses[!,:price]
# Конвертация данных в массив:
# F = convert(Array{Float64,2},F)'
F = Matrix(F)'
y = y'

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,:])
```

[69] ✓ 0.1s

...

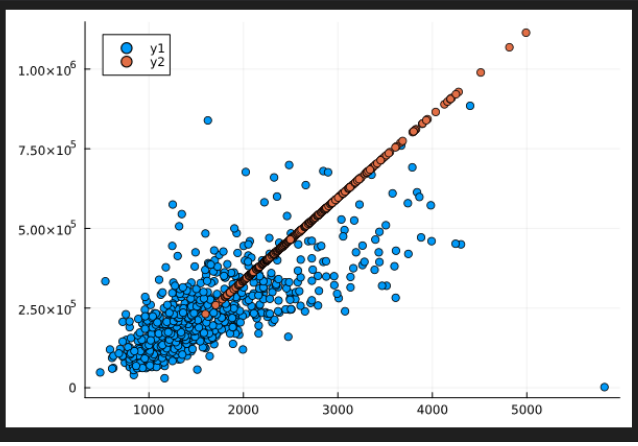
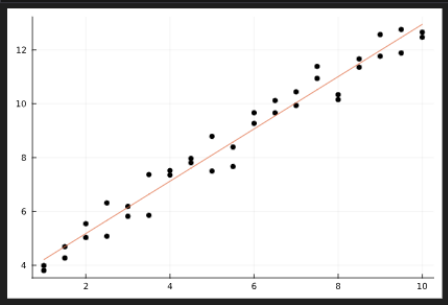


Рис. 3.7: Примеры

```
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

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




```
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)
```

```
size(xvals) = (300000,)
size(yvals) = (300000,)
  0.009181 seconds (2.56 k allocations: 191.812 KiB, 90.26% compilation time)

(1.0000000046681807, 3.000101072662801)
```

Рис. 3.8: Примеры

Затем выполним задания(рис. 3.9 - 3.15)

```
using RDatasets
iris = dataset("datasets", "iris") |> DataFrame
iris_spec = Dict("setosa"=>1,"versicolor"=>2, "virginica"=>3)
iris[!,:5] = [iris_spec[item] for item in iris[!,:5]];
```

✓ 5.5s

```
X = Matrix(iris)'
```

✓ 0.4s

```
5×150 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
 3.5  3.0  3.2  3.1  3.6  3.9  3.4  3.4     3.2  3.3  3.0  2.5  3.0  3.4  3.0
 1.4  1.4  1.3  1.5  1.4  1.7  1.4  1.5     5.9  5.7  5.2  5.0  5.2  5.4  5.1
 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
 1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0     3.0  3.0  3.0  3.0  3.0  3.0  3.0
```

```
# Задание количества кластеров:
k = length(unique(iris[!,:5]))
# Определение k-среднего:
C = kmeans(X,k)

# Формирование фрейма данных:
df = DataFrame(cluster = C.assignments,SepalLength = iris[!,:SepalLength],
    SepalWidth = iris[!,:SepalWidth], PetalLength = iris[!,:PetalLength],
    PetalWidth = iris[!,:PetalWidth], Species = iris[!,:Species]);
```

✓ 0.0s

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

✓ 2.0s

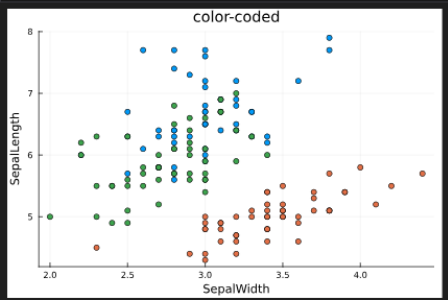


Рис. 3.9: Задание 1

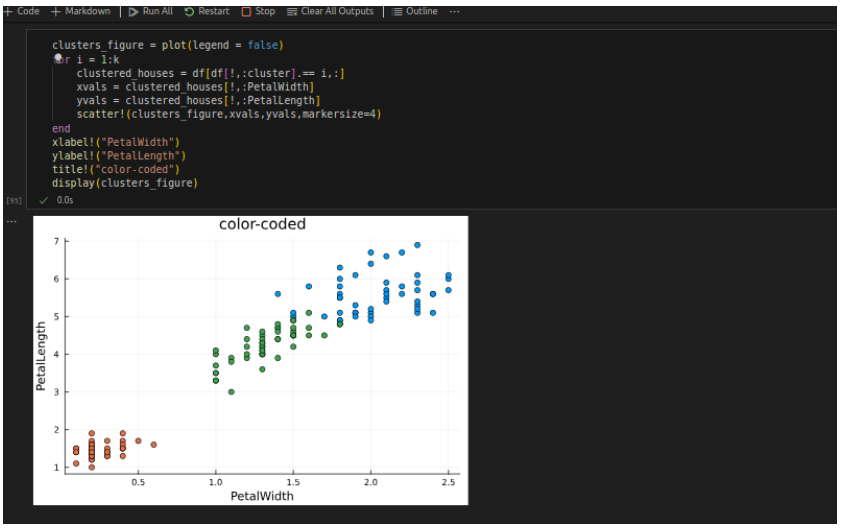


Рис. 3.10: Задание 1

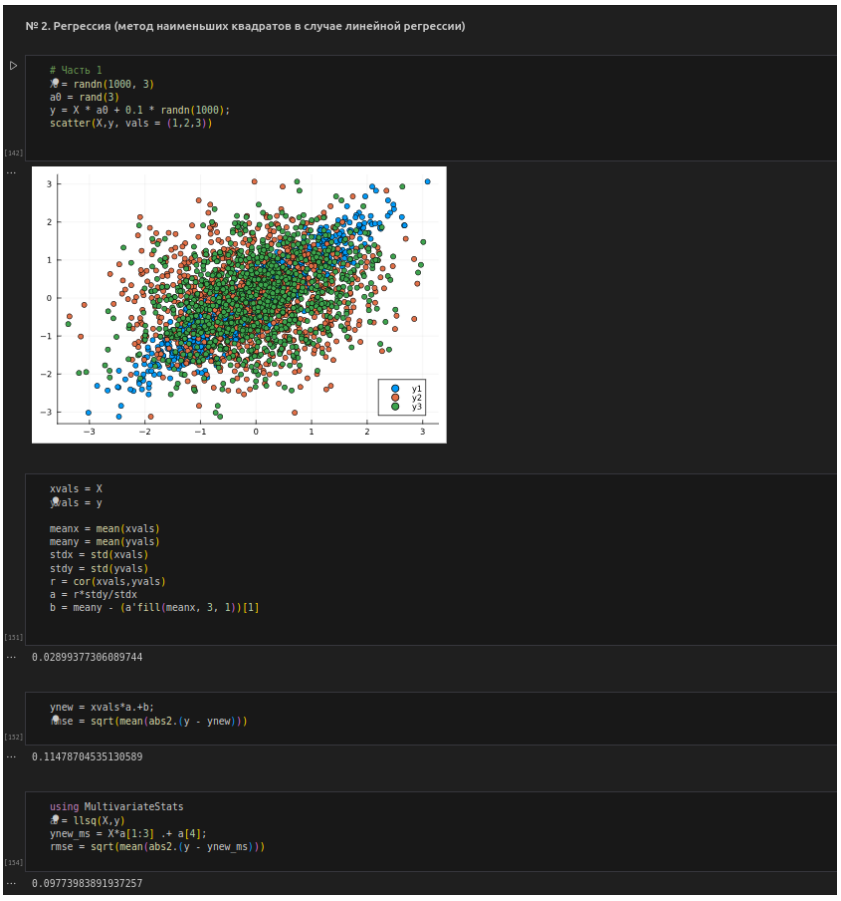


Рис. 3.11: Задания 2

Рис. 3.12: Задание 2

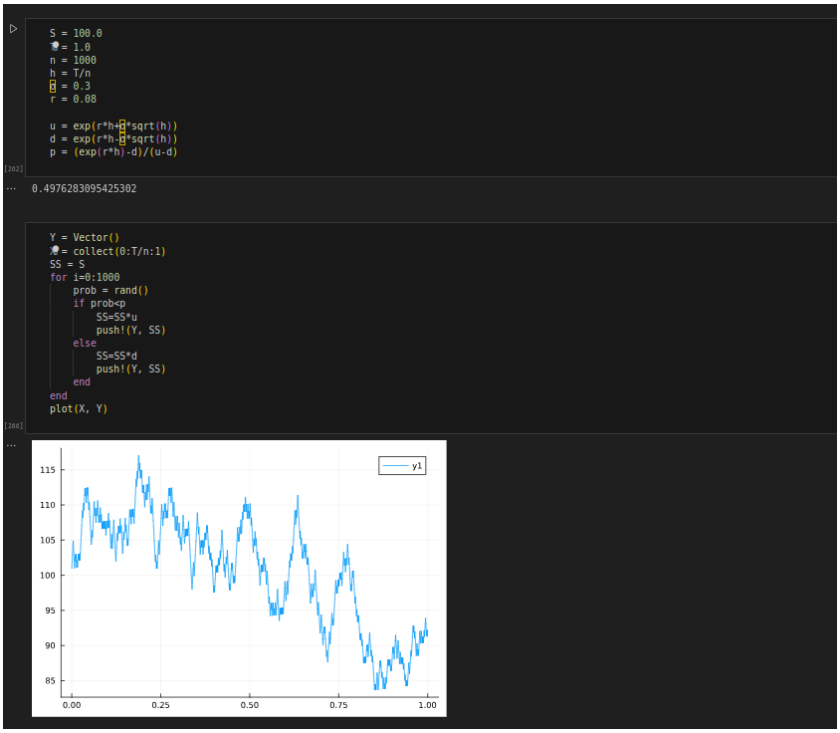


Рис. 3.13: Задание 3

```
function createPath(S::Float64, r::Float64, σ::Float64, T::Float64, n::Int64)
    Y = Vector()
    X = collect(0:T/n:1)
    u = exp(r*h+σ*sqrt(h))
    d = exp(r*h-σ*sqrt(h))
    p = (exp(r*h)-d)/(u-d)
    for i=0:1000
        prob = rand()
        if prob<p
            S=S*u
            push!(Y, S)
        else
            S=S*d
            push!(Y, S)
        end
    end
    return(X, Y)
end
```

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

```
p = plot()
for i=1:10
    plot!(p, createPath(S, r, σ, T, n))
end
p
```

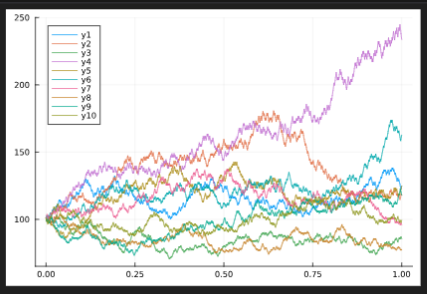


Рис. 3.14: Задание 3

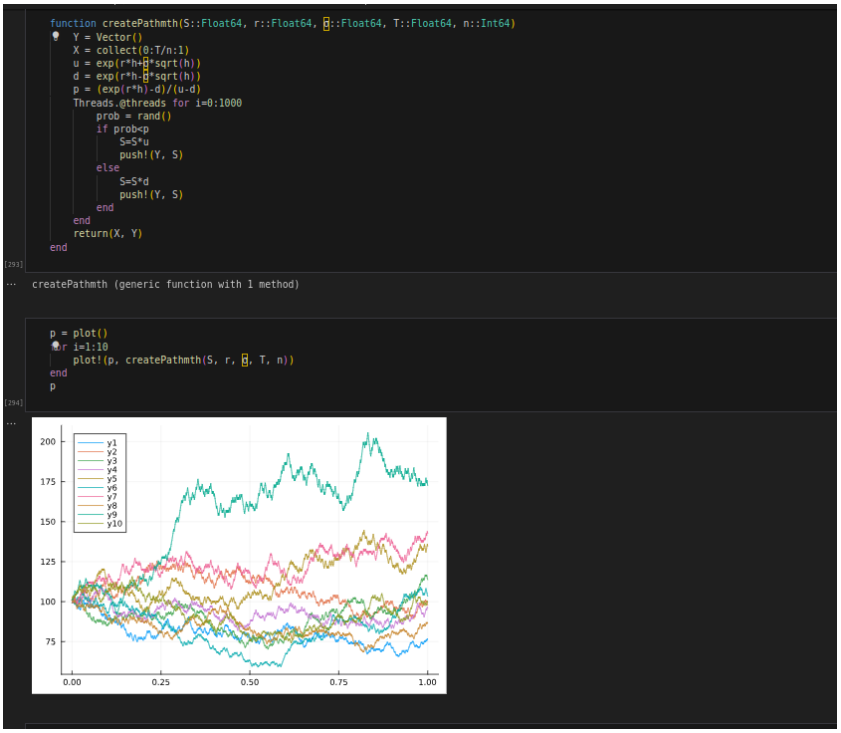


Рис. 3.15: Задание 3

4 Выводы

В результате выполнения работы освоили использование специализированных пакетов Julia для обработки данных.

Список литературы

1. JuliaLang [Электронный ресурс]. 2024 JuliaLang.org contributors. URL: https://julialang.org/ (дата обращения: 11.10.2024).
2. Julia 1.11 Documentation [Электронный ресурс]. 2024 JuliaLang.org contributors. URL: https://docs.julialang.org/en/v1/ (дата обращения: 11.10.2024).