Отчет по лабораторной работе №7

Введение в работу с данными

Легиньких Галина Андреевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

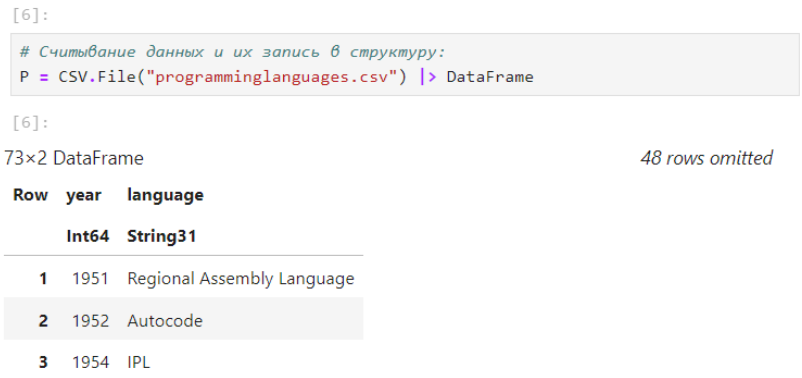
Основной целью работы является специализированных пакетов Julia для обработки данных.

2 Задание

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

3 Выполнение лабораторной работы

1. Для начала я повторила примеры по данной теме. Начала с раздела считывание данных. В Julia для работы с такого рода структурами данных используют пакеты CSV, DataFrames, RDatasets, FileIO. Скачала эти пакеты и попробовала прочитать данные из скаченного датафрейма. (рис. 3.1)



```
[6]:
# Считывание данных и их запись в структуру:
P = CSV.File("programminglanguages.csv") |> DataFrame
```

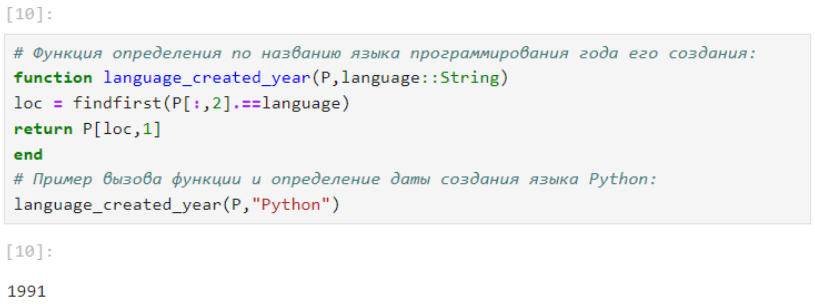
[6]:

73×2 DataFrame — *48 rows omitted*

Row	year	language
	Int64	String31
1	1951	Regional Assembly Language
2	1952	Autocode
3	1954	IPL

Рис. 3.1: Считывание данных

Далее приведём пример функции, в которой на входе указывается название языка программирования, а на выходе — год его создания. (рис. 3.2)



```
[10]:
# 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")
```

[10]:

```
1991
```

Рис. 3.2: Функция язык-год

При этом, чтобы убрать в функции зависимость данных от регистра, необходимо изменить исходную функцию следующим образом. (рис. 3.3)

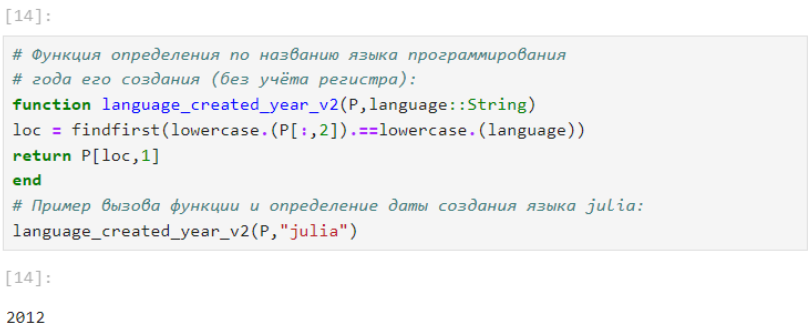


Рис. 3.3: Зависимость от регистра

2. Далее попробовала записать данные в файл, а так же задала при сохранении тип файла и разделитель.

3. При работе с данными бывает удобно записать их в формате словаря.При инициализации словаря можно задать конкретные типы данных для ключей и значений,а можно инициировать пустой словарь, не задавая строго структуру. (рис. 3.4)

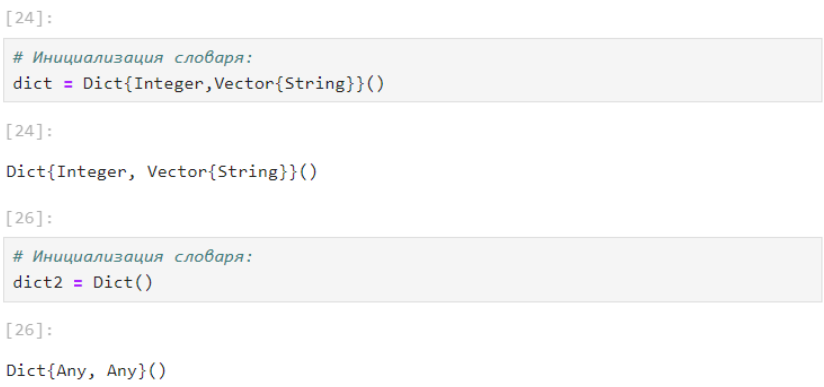


Рис. 3.4: Инициализация словаря

В результате при вызове словаря можно, выбрав любой год, узнать, какие языки программирования были созданы в этом году. (рис. 3.5)

[28]:

```
# 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[2003]
```

[28]:

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

Рис. 3.5: Работа со словарем

4. На примере с данными о языках программирования и годах их создания зададим структуру DataFrame. Попробовала вывести один столбец. (рис. 3.6)

[32]:

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

[32]:

73×2 DataFrame *48 rows omitted*

Row	year	language
	Int64	String31
1	1951	Regional Assembly Language
2	1952	Autocode

Рис. 3.6: Переменная со структурой датафрейм

А так же получила статистические сведения о фрейме. (рис. 3.7)

[36]:

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

[36]:

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. 3.7: Статистические сведения о фрейме

5. С данными можно работать также как с наборами данных через пакет RDatasets языка R.

6. Попробовала порабоать с переменными отсутствующего типа. (рис. 3.8)

```
[46]:
# Отсутствующий тип:
a = missing
typeof(a)

[46]:
Missing
```

Рис. 3.8: Переменная отсутсвующего типа

7. Перешла к обработке данных. И первый метод это метод k-средних. Задача кластеризации данных заключается в формировании однородной группы упорядоченных по какому-то признаку данных. Метод k-средних позволяет минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров. Рассмотрела задачу кластеризации данных на примере данных о недвижимости. (рис. 3.9) (рис. 3.10) (рис. 3.11)

```
[72]:
# Загрузка данных:
houses = CSV.File("houses.csv") |> DataFrame

[72]:
```

985×12 DataFrame *960 rows omitted*

Row	street	city	zip	state	beds	baths	sq__ft	type	s
	String	String15	Int64	String3	Int64	Int64	Int64	String15	S
1	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836	Residential	
2	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167	Residential	
3	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796	Residential	

Рис. 3.9: Кластеризация данных на примере данных о недвижимости

[74]:

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

[74]:

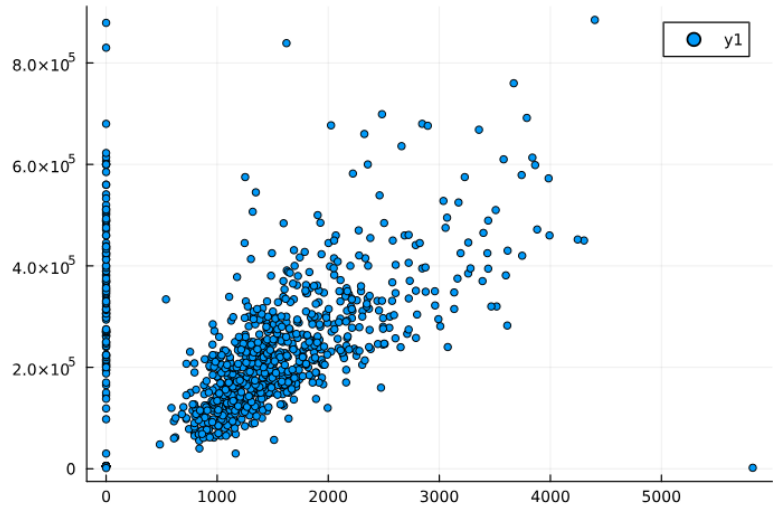


Рис. 3.10: Цены на недвижимость в зависимости от площади

[76]:

```
# 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)
```

[76]:

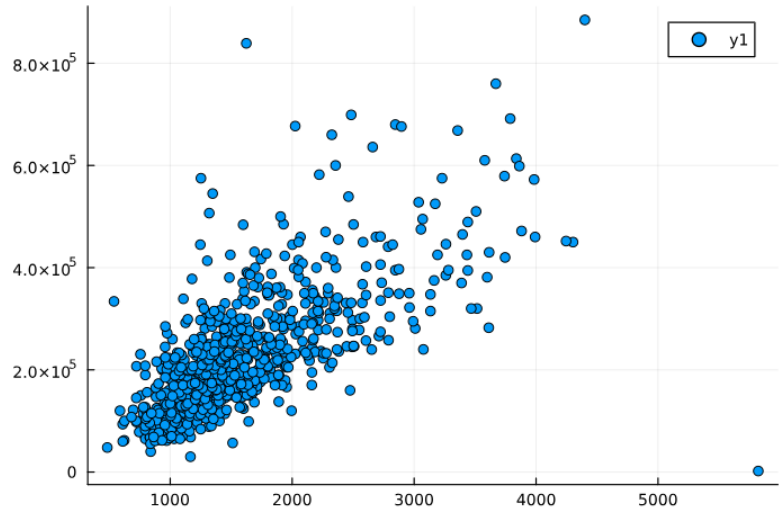


Рис. 3.11: Цены на недвижимость в зависимости от площади (исключены артефакты данных)

Сначала подключаем необходимые пакеты и формируем данные в нужном виде: (рис. 3.12) (рис. 3.13)

[86]:

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

[86]:

814×2 DataFrame — *789 rows omitted*

Row	latitude	longitude
	Float64	Float64
1	38.6319	-121.435
2	38.4789	-121.431

Рис. 3.12: Добавление данных

[92]:

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

[92]:

```
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
```

Рис. 3.13: Конвертация данных в матричный вид

В качестве критерия для формирования кластеров данных и определения количества кластеров попробуем использовать количество почтовых индексов: (рис. 3.14)

[96]:

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

[96]:

```
66
```

Рис. 3.14: Задание количества кластеров

Далее сформируем новый фрейм, включающий исходные данные о недвижимости и столбец с данными о назначенном каждому дому кластере (рис. 3.15)

[100]:

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

[100]:

814×5 DataFrame — *789 rows omitted*

Row	cluster	city	latitude	longitude	zip
	Int64	String15	Float64	Float64	Int64
1	28	SACRAMENTO	38.6319	-121.435	95838
2	34	SACRAMENTO	38.4789	-121.431	95823

Рис. 3.15: Формирование фрейма данных

Плстроила график, обозначив каждый кластер отдельным цветом (рис. 3.16)

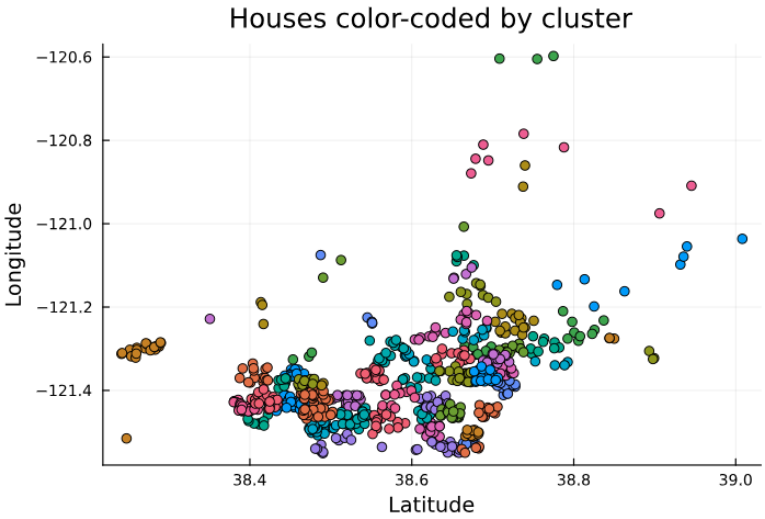


Рис. 3.16: График кластеров

Построила график, раскрасив кластеры по почтовому индексу (рис. 3.17)

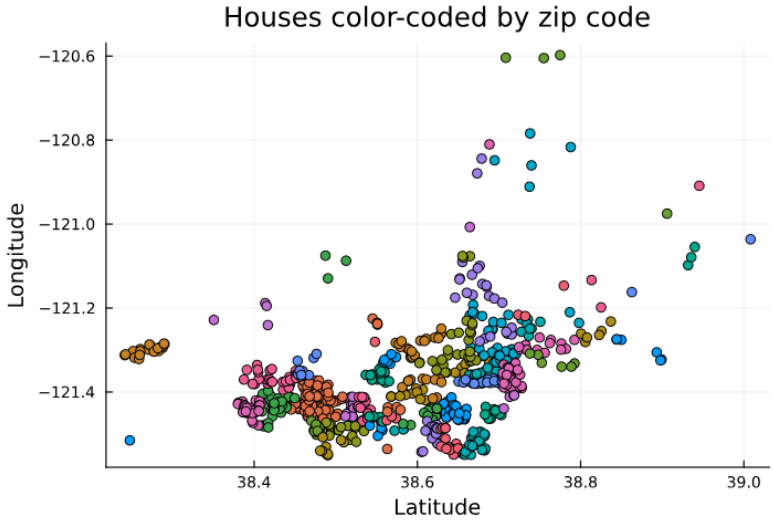


Рис. 3.17: График кластеров 2

8. Кластеризация данных. Метод k ближайших соседей. Данный метод заключается в отнесении объекта к тому из известных классов, который является наиболее распространённым среди ⊠ соседей данного элемента. В случае использования метода для регрессии, объекту присваивается среднее значение по ⊠ ближайшим к нему объектам. (рис. 3.18) (рис. 3.19)

[112]:

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

[112]:

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

[114]:

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

[114]:

```
([70, 764, 196, 125, 557, 368, 415, 92, 112, 683], [0.0, 0.006264891539364138, 0.
00825320259050462, 0.008473585132630057, 0.009164073548370188, 0.0094050651246977
06, 0.009921759722950759, 0.009941028618812013, 0.010332637707777167, 0.011168993
911721985])
```

Рис. 3.18: Метод главных компонент

[116]:

```
# 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)
```

[116]:

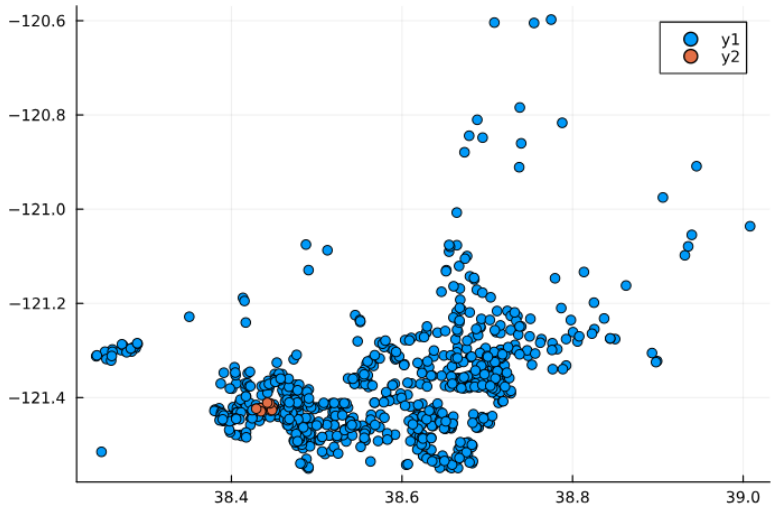
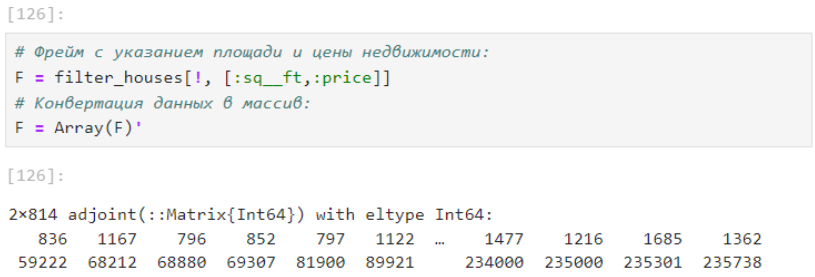


Рис. 3.19: Определение соседей объекта недвижимости

9. Обработка данных. Метод главных компонент. Метод главных компонент

(Principal Components Analysis, PCA) позволяет уменьшить размерность данных, потеряв наименьшее количество полезной информации. Метод имеет широкое применение в различных областях знаний, например, при визуализации данных, компрессии изображений, в эконометрике, некоторых гуманитарных предметных областях, например, в социологии или в политологии.

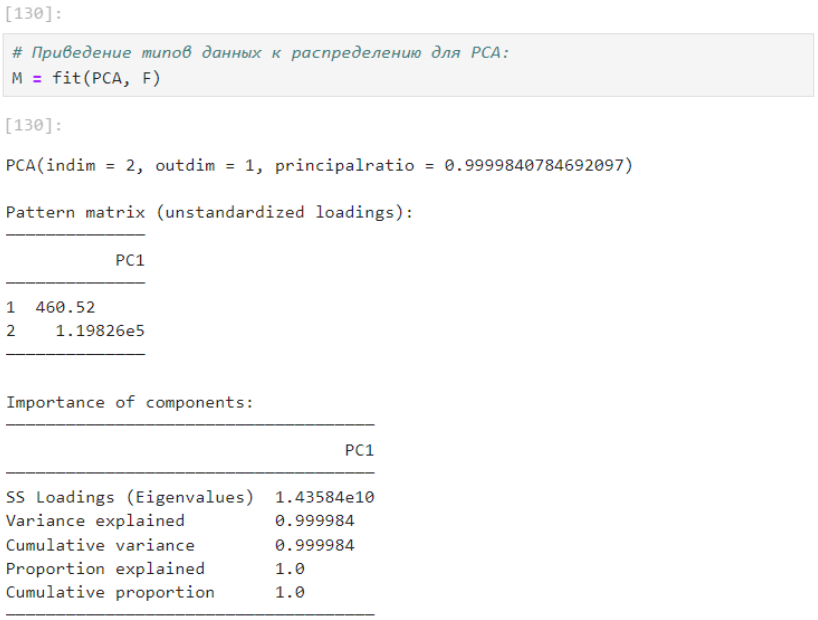
На примере с данными о недвижимости попробуем уменьшить размеры данных о цене и площади из набора данных домов (рис. 3.20) (рис. 3.21) (рис. 3.22)



```
[126]:
# Фрейм с указанием площади и цены недвижимости:
F = filter_houses[!, [:sq__ft,:price]]
# Конвертация данных в массив:
F = Array(F)'
```

```
[126]:
2×814 adjoint(::Matrix{Int64}) with eltype Int64:
   836   1167    796    852    797   1122  …    1477    1216    1685    1362
 59222  68212  68880  69307  81900  89921     234000  235000  235301  235738
```

Рис. 3.20: Создание фрейма



```
[130]:
# Приведение типов данных к распределению для PCA:
M = fit(PCA, F)
```

```
[130]:
PCA(indim = 2, outdim = 1, principalratio = 0.9999840784692097)

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

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

Рис. 3.21: Приведение типов данных к распределению для PCA

[140]:

```
# 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,:])
```

[140]:

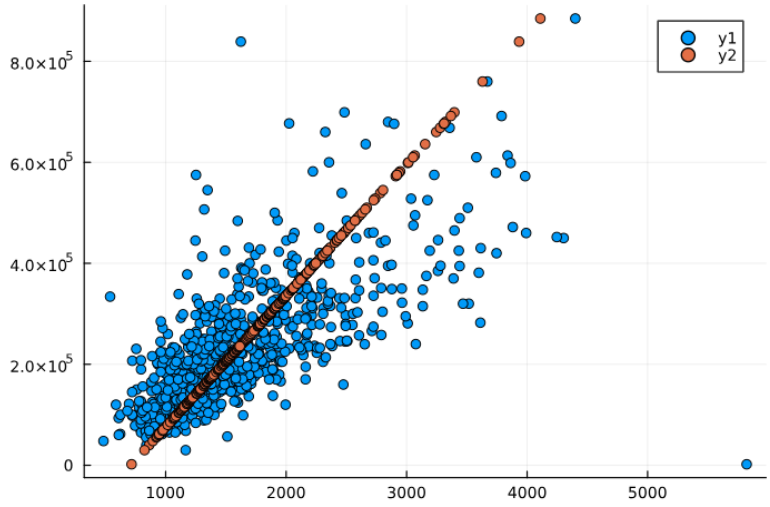


Рис. 3.22: Определение главных компонент для данных по объектам недвижимости

10. Обработка данных. Линейная регрессия. Регрессионный анализ представляет собой набор статистических методов исследования влияния одной или нескольких независимых переменных (регрессоров) на зависимую (критериальная) переменную. Терминология зависимых и независимых переменных отражает лишь математическую зависимость переменных, а не причинноследственные отношения.

Зададим случайный набор данных (можно использовать и полученные экспериментальным путём какие-то данные). Попробуем найти для данных лучшее соответствие (рис. 3.23)

[142]:

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

[142]:

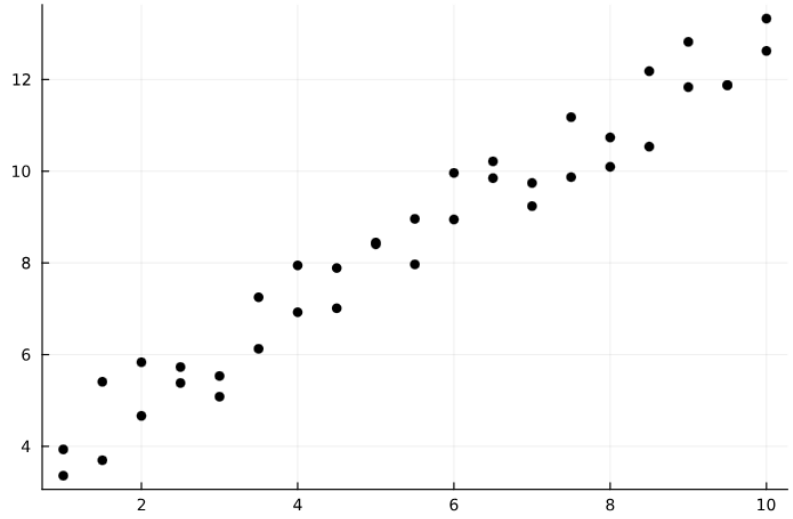


Рис. 3.23: Исходные данные

Применим функцию линейной регрессии для построения соответствующего графика значений (рис. 3.24)

[146]:

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

[146]:

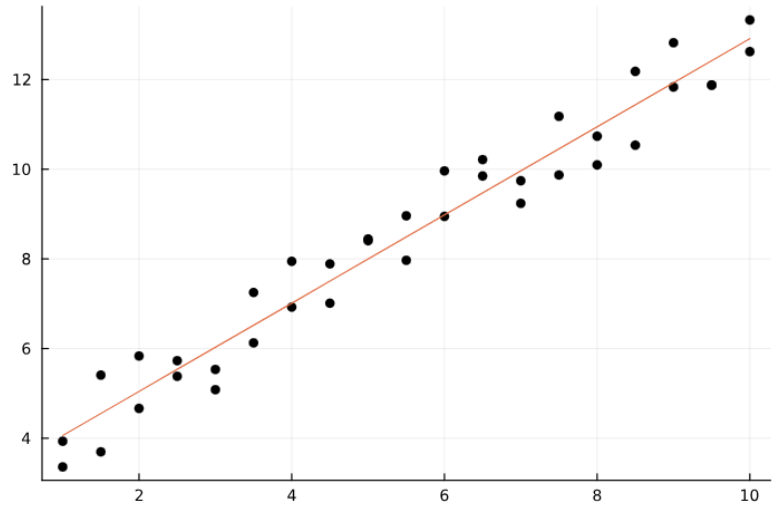


Рис. 3.24: Линейная регрессия

11. Перешла к заданиям для самостоятельного выполнения. Нумерация соответствует.

- Задание на Кластеризацию. (рис. 3.25) (рис. 3.26)

[162]:

```
# Загрузка данных iris
iris = dataset("datasets", "iris")

# Извлечение числовых данных (первые четыре столбца) и преобразование в массив
data = Matrix(iris[:, 1:4])'
# Транспонируем массив для использования в k-средних (объекты — столбцы)

# Кластеризация методом k-средних на 3 кластера (так как в данных есть 3 вида ир
k = 3
result = kmeans(data, k)

# Получение меток кластеров
labels = result.assignments

# Построение точечной диаграммы (используем первые два признака для визуализации
scatter(
    data[1, :], data[2, :],
    group = labels,
    title = "K-means Clustering of Iris Dataset",
    xlabel = "Feature 1: Sepal Length",
    ylabel = "Feature 2: Sepal Width",
    legend = :topright,
    markersize = 5
)
```

Рис. 3.25: Задание 1. Код

[162]:

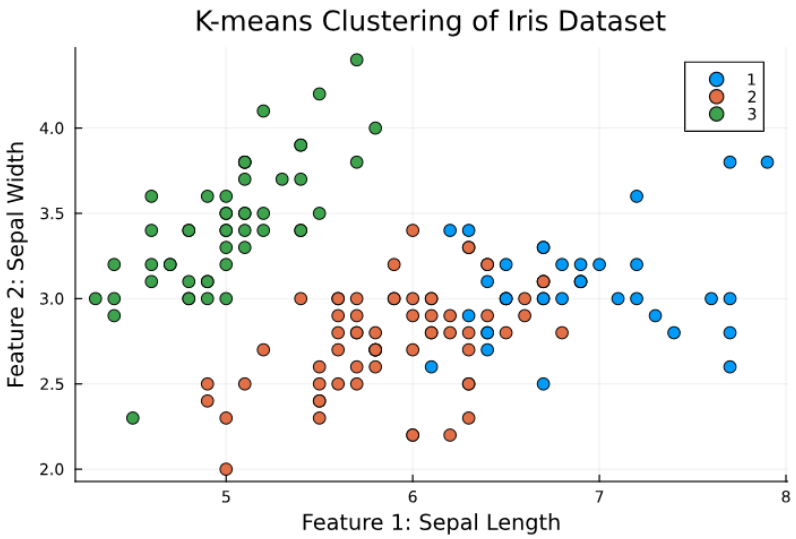


Рис. 3.26: Задание 1. График

- Задание на регрессию (часть 1) (рис. 3.27) (рис. 3.28)

[172]:
```julia
using LinearAlgebra
using DataFrames
using GLM

# Генерация данных
N = 1000  # Количество наблюдений
X = randn(N, 3)  # Матрица факторов (N x 3)
a0 = rand(3)  # Истинные коэффициенты
y = X * a0 + 0.1 * randn(N)  # Линейная зависимость с добавлением шума

# Добавляем столбец единиц для свободного члена
X2 = hcat(ones(N), X)

# Проверка матрицы X^T X
XtX = X2' * X2
if !isposdef(XtX)
    println("Матрица X^T X не положительно определена. Добавляем регуляризацию.")
end

# Регуляризация вручную: Ridge Regression
λ = 1e-6  # Маленькое значение регуляризации
XtX_reg = XtX + λ * I  # Добавляем λI к X^T X
beta_ridge = XtX_reg \ (X2' * y)  # Решение системы
println("Оценка коэффициентов (с регуляризацией): ", beta_ridge)

# Сравнение с GLM.jl
df = DataFrame(hcat(X, y), [:X1, :X2, :X3, :Y])  # Создаем DataFrame
model = lm(@formula(Y ~ X1 + X2 + X3), df)  # Модель линейной регрессии
println("Оценка с использованием GLM.jl: ", coef(model))

# Проверка сходства коэффициентов
println("Разница между оценками (GLM - Ridge): ", coef(model) - beta_ridge)
```

Рис. 3.27: Задание 2.1. Код

```
Оценка коэффициентов (с регуляризацией): [-0.004361583263215564, 0.643371334441835
4, 0.6787211781696519, 0.6094844818961423]
Оценка с использованием GLM.jl: [-0.0043615833086847935, 0.643371335071556, 0.67872
11788728496, 0.6094844825440442]
Разница между оценками (GLM - Ridge): [-4.546922907633366e-11, 6.297206089911356e-1
0, 7.031976112514826e-10, 6.479018432870021e-10]
```

Рис. 3.28: Задание 2.1. Решение

- Задание на регрессию (часть 2) (рис. 3.29) (рис. 3.30)

[188]:
```julia
# Часть 2: Линейная регрессия для одномерного случая
# Создание данных
X = rand(100)
y = 2X + 0.1 * randn(100)

# Добавляем столбец единиц
X2 = hcat(ones(length(X)), X)

# МНК-оценка
beta_mnk = (X2' * X2) \ (X2' * y)
println("МНК-оценка коэффициентов: ", beta_mnk)

# Построение графика
scatter(X, y, label="Данные", title="График регрессии", xlabel="X", ylabel="Y")
```

```
МНК-оценка коэффициентов: [0.006898153545410991, 1.9878127109267747]
```

Рис. 3.29: Задание 2.2. Код

[188]:

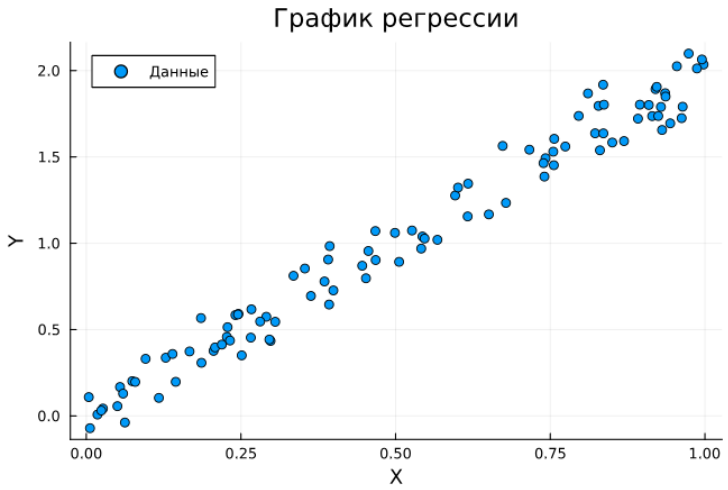


Рис. 3.30: Задание 2.2. График

- Модель ценообразования биномиальных опционов (a) (рис. 3.31) (рис. 3.32)

[192]:

```julia
using Plots
using Random

# Параметры
S = 100.0      # Начальная цена акции
T = 1.0        # Длина биномиального дерева в годах
n = 10000      # Количество периодов
σ = 0.3        # Волатильность
r = 0.08       # Годовая процентная ставка

# Вычисления
h = T / n                # Длина одного периода
u = exp(r * h + σ * sqrt(h))  # Рост
d = exp(r * h - σ * sqrt(h))  # Падение
p_star = (exp(r * h) - d) / (u - d)  # Вероятность роста

# Построение одной траектории
function generate_path(S, u, d, p_star, n)
    path = [S]  # Начальная цена
    for _ in 1:n
        rand() < p_star ? push!(path, path[end] * u) : push!(path, path[end] * d)
    end
    return path
end

# Генерация и построение траектории
path = generate_path(S, u, d, p_star, n)
plot(0:n, path, xlabel="Периоды", ylabel="Цена акции",
    title="Траектория курса акций")
```

Рис. 3.31: Задание 3.1. Код

[192]:

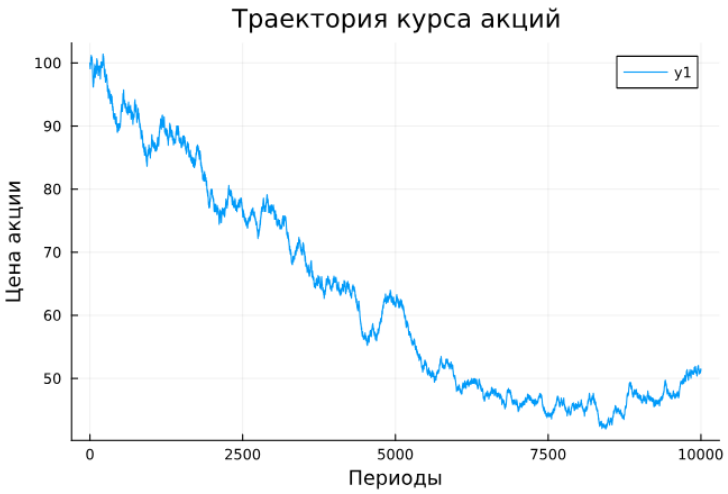


Рис. 3.32: Задание 3.1. График

- Модель ценообразования биномиальных опционов (b) (рис. 3.33)

•[214]:

```julia
# Функция для создания траектории
function createPath(S::Float64, r::Float64, σ::Float64, T::Float64, n::Int64)
    h = T / n
    u = exp(r * h + σ * sqrt(h))
    d = exp(r * h - σ * sqrt(h))
    p_star = (exp(r * h) - d) / (u - d)
    return generate_path(S, u, d, p_star, n)
end

# Генерация 10 траекторий
plot(title="10 траекторий курса акций", xlabel="Периоды", ylabel="Цена акции")
for _ in 1:10
    path = createPath(S, r, σ, T, n)
    plot!(0:n, path, legend=false)
end
```

Рис. 3.33: Задание 3.2. Код

- Модель ценообразования биномиальных опционов (c) (рис. 3.34) (рис. 3.35)

```julia
using Base.Threads
# Функция для генерации траектории
function createPath(S, r, σ, T, n)
    dt = T / n
    path = zeros(Float64, n+1)
    path[1] = S
    for i in 2:n+1
        path[i] = path[i-1] * exp((r - 0.5 * σ^2) * dt + σ * sqrt(dt) * randn())
    end
    return path
end

# Параметры для генерации траекторий
S = 100.0  # Начальная цена акции
r = 0.05   # Безрисковая ставка
σ = 0.2    # Волатильность
T = 1.0    # Время до экспирации (в годах)
n = 252    # Количество шагов (дней в году)

# Многопоточная генерация траекторий
n_trajectories = 10
trajectories = Vector{Vector{Float64}}(undef, n_trajectories)

# Параллельный расчет траекторий
Threads.@threads for i in 1:n_trajectories
    trajectories[i] = createPath(S, r, σ, T, n)
end

# Построение графика всех траекторий
p = plot(title="Распараллеленные траектории курса акций", xlabel="Периоды", ylabel="Цена акции")

# Добавление траекторий на график
for path in trajectories
    plot!(p, 0:n, path, legend=false)
end

# Явный вывод графика
display(p)
```

Рис. 3.34: Задание 3.3. Код

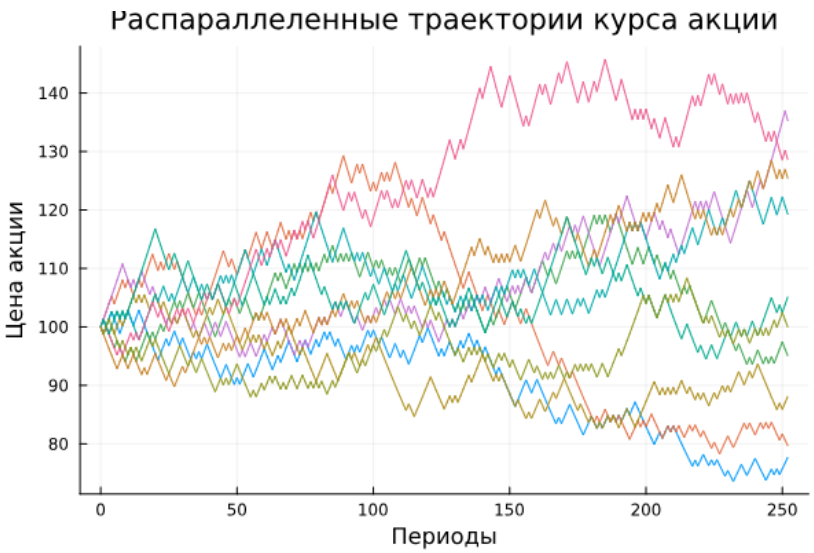


Рис. 3.35: Задание 3.3. График

- Модель ценообразования биномиальных опционов (d) (рис. 3.36)

[224]:
```
# Параметры
S = 100.0     # Начальная цена акции
T = 1.0       # Длина биномиального дерева в годах
n = 10000     # Количество периодов
σ = 0.3       # Волатильность
r = 0.08      # Годовая процентная ставка

# Вычисления
h = T / n               # Длина одного периода
u = exp(r * h + σ * sqrt(h))  # Рост
d = exp(r * h - σ * sqrt(h))  # Падение
p_star = (exp(r * h) - d) / (u - d)  # Вероятность роста
path = createPath(S, r, σ, T, n)
plot(0:n, path, xlabel="Периоды", ylabel="Цена акции", title="Одна траектория курса акций")
```

[224]:

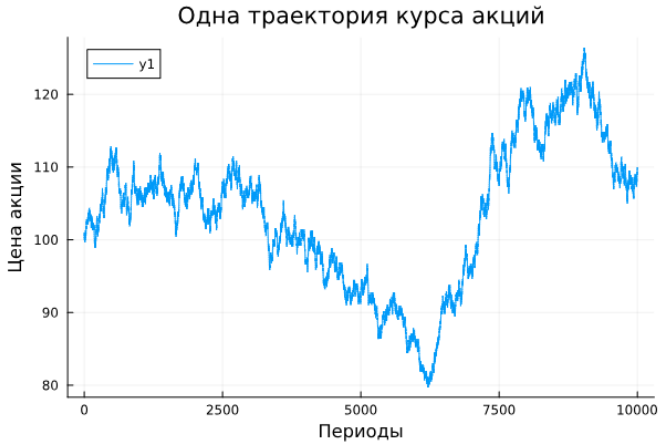


Рис. 3.36: Задание 3.4. Код и График

4 Вывод

Изучила специализированные пакеты Julia для обработки данных.