Компьютерный практикум по статистическому анализу данных

Лабораторная работа №7: Введение в Data Science

Кармацкий Никита Сергеевич

Российский университет дружбы народов, Москва, Россия

Цель лабораторной работы

- Изучить специализированные пакеты Julia для обработки данных.

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

Считывание данных

[1]:
```
# 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
```

```
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.11/Project.toml`
 No Changes to `~/.julia/environments/v1.11/Manifest.toml`
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.11/Project.toml`
 No Changes to `~/.julia/environments/v1.11/Manifest.toml`
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.11/Project.toml`
 No Changes to `~/.julia/environments/v1.11/Manifest.toml`
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.11/Project.toml`
 No Changes to `~/.julia/environments/v1.11/Manifest.toml`
```

Рис. 1: Установка пакетов

[2]:
```
# 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")
```

[2]: 1991

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

```
[3]: 2012
```

[4]:
```
# Пример вызова функции и определение даты создания языка Julia:
language_created_year(P,"julia")
```

```
MethodError: no method matching getindex(::DataFrame, ::Nothing, ::Int64)
The function `getindex` exists, but no method is defined for this combination of argument types

Closest candidates are:
  getindex(::DataFrame, ::typeof(!), ::Union{Signed, Unsigned})
   @ DataFrames ~/.julia/packages/DataFrames/kcA9R/src/dataframe/dataframe.jl:548
  getindex(::DataFrame, ::Colon, ::Union{AbstractString, Signed, Symbol, Unsigned})
   @ DataFrames ~/.julia/packages/DataFrames/kcA9R/src/dataframe/dataframe.jl:542
  getindex(::DataFrame, ::InvertedIndex, ::Union{AbstractString, Signed, Symbol, Unsigned})
   @ DataFrames ~/.julia/packages/DataFrames/kcA9R/src/dataframe/dataframe.jl:538
  ...
```

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


[5]: 2012

Считывание данных

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

```
[6]: 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"
```

1.2. Запись данных в файл ¶

Рис. 6: Построчное считывание данных

1.2. Запись данных в файл

[8]:
```
# Запись данных в CSV-файл:
CSV.write("programming_languages_data2.csv", P)
```

[8]:
```
"programming_languages_data2.csv"
```

[9]:
```
# Пример записи данных в текстовый файл с разделителем ',':
writedlm("programming_languages_data.txt", Tx, ',')
```

[10]:
```
# Пример записи данных в текстовый файл с разделителем '-':
writedlm("programming_languages_data2.txt", Tx, '-')
```

Запись данных в файл

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

```
[11]: 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"
```

Рис. 9: Проверка корректности считывания созданного текстового файла

1.3. Словари

[13]:
```
# Инициализация словаря:
dict = Dict{Integer,Vector{String}}()
```

[13]:
```
Dict{Integer, Vector{String}}()
```

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

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

[15]:
```
# 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
```

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

```
[16]: 2-element Vector{String}:
       "Groovy"
       "Scala"
```

1.4. DataFrames

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

[19]:
```
# 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)
```

[19]: 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
```


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

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

[23]: 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}

1.6. Работа с переменными отсутствующего типа (Missing Values

[25]:
```
# Отсутствующий тип:
a = missing
typeof(a)
```

[25]: Missing

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

```
[26]: missing
```

[27]:
```
# Определение перечня продуктов:
foods = ["apple", "cucumber", "tomato", "banana"]
# Определение калорий:
calories = [missing,47,22,105]
# Определение типа переменной:
typeof(calories)
```

[27]:
```
Vector{Union{Missing, Int64}} (alias for Array{Union{Missing, Int64}, 1})
```

[28]:
```
# Подключаем пакет Statistics:
using Statistics
# Определение среднего значения:
mean(calories)
# Определение среднего значения без значений с отсутствующим типом:
mean(skipmissing(calories))
```

[28]: 58.0

[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

[54]:
```
# Подключаем пакет FileIO:
using FileIO
```

[55]:
```
# Подключаем пакет ImageIO:
import Pkg
Pkg.add("ImageIO")
```

```
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.11/Project.toml`
 No Changes to `~/.julia/environments/v1.11/Manifest.toml`
```

[56]:
```
X1 = load("figure_karmatskiy.png")
display(X1)
```

```
400×600 Array{RGBA{N0f8},2} with eltype ColorTypes.RGBA{FixedPointNumbers.N0f8}:
 RGBA{N0f8}(1.0,1.0,1.0,1.0)  …  RGBA{N0f8}(1.0,1.0,1.0,1.0)
 RGBA{N0f8}(1.0,1.0,1.0,1.0)     RGBA{N0f8}(1.0,1.0,1.0,1.0)
 RGBA{N0f8}(1.0,1.0,1.0,1.0)     RGBA{N0f8}(1.0,1.0,1.0,1.0)
 RGBA{N0f8}(1.0,1.0,1.0,1.0)     RGBA{N0f8}(1.0,1.0,1.0,1.0)
```

[110]:
```
@show typeof(X1);
@show size(X1);
```

```
typeof(X1) = Matrix{RGBA{FixedPointNumbers.N0f8}}
size(X1) = (400, 600)
```

2. Обработка данных: стандартные алгоритмы машинного обученияв Julia

2.1. Кластеризация данных. Метод k-средних

[63]:
```
# Загрузка пакетов:
import Pkg
Pkg.add("DataFrames")
Pkg.add("Statistics")
using DataFrames
using CSV
import Pkg
Pkg.add("Plots")
```

```
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.11/Project.toml`
  No Changes to `~/.julia/environments/v1.11/Manifest.toml`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.11/Project.toml`
  No Changes to `~/.julia/environments/v1.11/Manifest.toml`
   Resolving package versions...
```

```
[116]: houses = CSV.File("houses.csv")|>DataFrame
```

[116]: 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

Рис. 26: Загрузка данных

[64]:
```
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)
```

[64]:

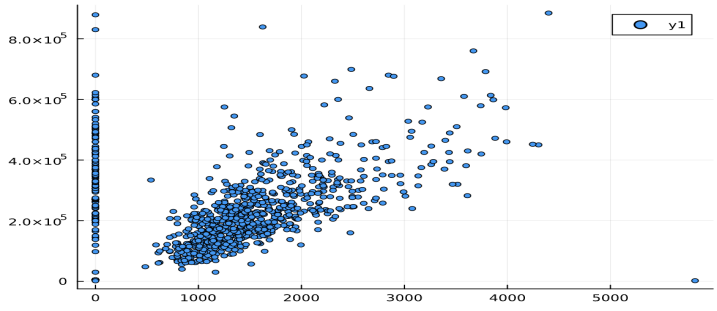

Рис. 27: Построение графика цен на недвижимость в зависимости от площади

[65]:
```
# 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)
```

[65]:

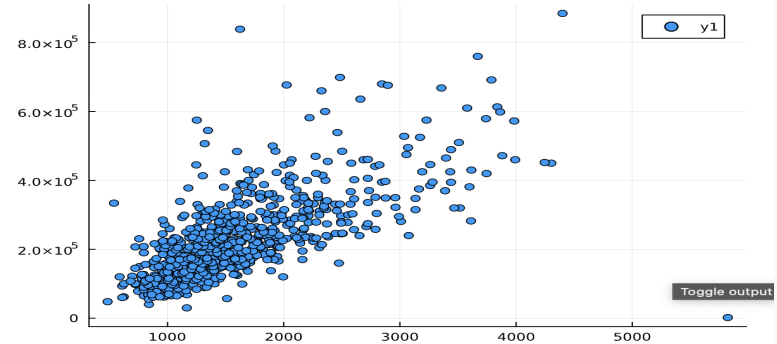

Рис. 28: Построение графика без “артефактов”

Средние цены для каждого типа домов:

4×2 DataFrame

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

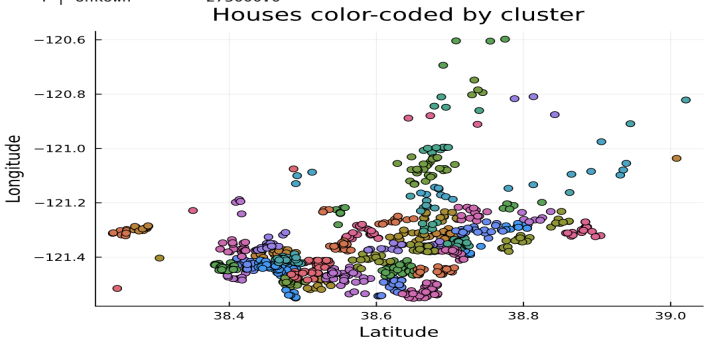

Рис. 29: Построение графика с кластерами разных цветов

[67]:
```
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)
```

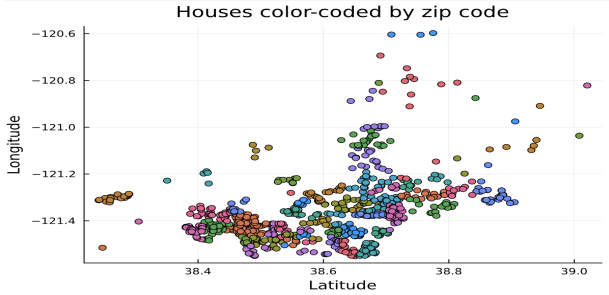

Рис. 30: Построение графика с кластерами разных цветов по почтовому индексу

2.2. Кластеризация данных. Метод k ближайших соседей

[69]:
```
# Подключение пакета NearestNeighbors:
import Pkg
Pkg.add("NearestNeighbors")
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)
```

```
  Resolving package versions...
 No Changes to `~/.julia/environments/v1.11/Project.toml`
 No Changes to `~/.julia/environments/v1.11/Manifest.toml`
```

[69]:

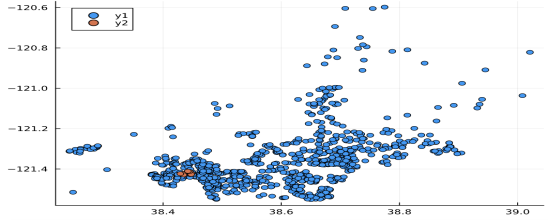

Рис. 31: Отображение на графике соседей выбранного объекта недвижимости

[70]:
```
# Фильтрация по районам соседних домов:
cities = filter_houses[idxs,:city]
```

[70]:
```
10-element PooledArrays.PooledVector{String15, UInt32, Vector{UInt32}}:
 "SACRAMENTO"
 "ELK GROVE"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "ELK GROVE"
 "ELK GROVE"
 "ELK GROVE"
```

2.3. Обработка данных. Метод главных компонент

[72]:
```
# 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

[74]:
```
xvals = repeat(1:0.5:10,inner=2)
yvals = 3 .+ xvals + 2*rand(length(xvals)) .- 1
scatter(xvals,yvals,color=:black,leg=false)
```

[74]:

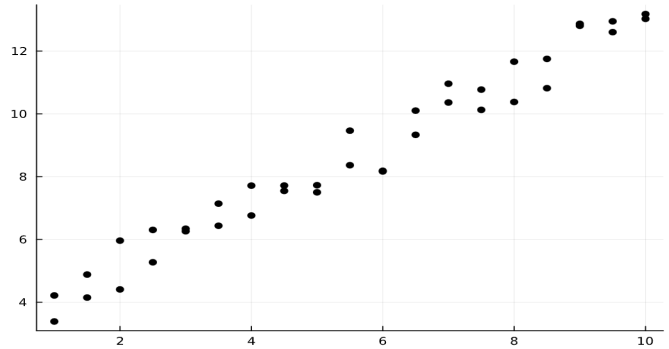

Рис. 34: Исходные данные

[75]:
```
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)
```

[75]:

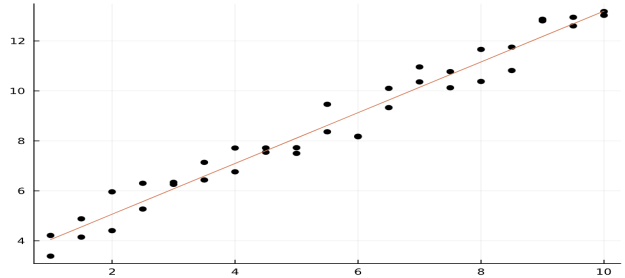

Рис. 35: Применение функции для построения графика

[76]:
```
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)
```

```
size(xvals) = (300000,)
size(yvals) = (300000,)
  0.098020 seconds (29.73 k allocations: 1.527 MiB, 95.77% compilation time)
```

Рис. 36: Сравнение

Самостоятельная работа

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]:

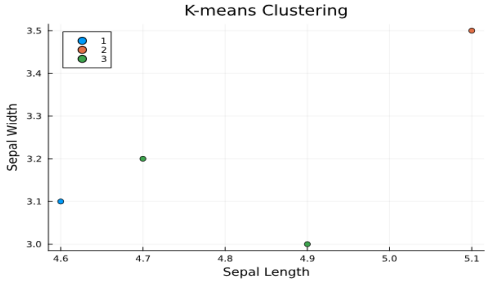

Рис. 37: Решение задания №1

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

Самостоятельная работа

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)
```

[49]:

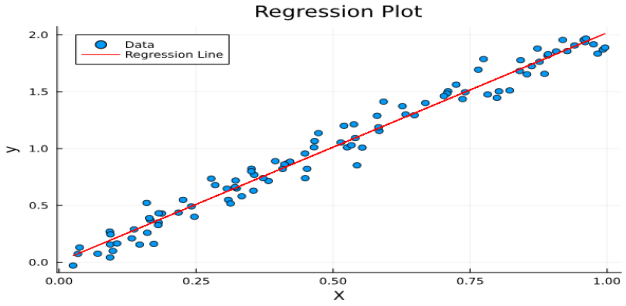

Рис. 39: Решение задания №2

Самостоятельная работа

a.

[50]:
```
# Параметры модели
S = 100       # Начальная цена акции
T = 1         # Длительность в годах
n = 10000     # Количество периодов
sigma = 0.3   # Волатильность
r = 0.08      # Годовая процентная ставка
h = T / n     # Длина одного периода

# Расчет u и d
u = exp(r * h + sigma * sqrt(h))
d = exp(r * h - sigma * sqrt(h))

# Цена акции на каждом шаге
function create_path(S, r, sigma, T, n)
    path = zeros(Float64, n+1)
    path[1] = S
    for i in 2:n+1
        if rand() > 0.5  # Пример случайного выбора направления
            path[i] = path[i-1] * u
        else
            path[i] = path[i-1] * d
        end
    end
    return path
end
# Генерация и построение траектории
path = create_path(S, r, sigma, T, n)
plot(path, label="Stock Price Path", xlabel="Time Step", ylabel="Price", title="Stock Price Trajectory")
```

[50]:

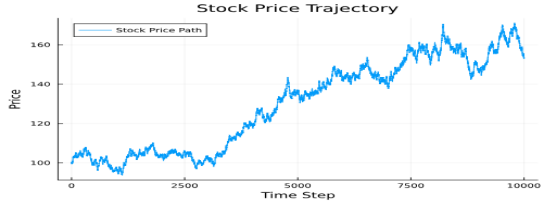

Рис. 40: Решение задания №3

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
```

c.

```
[ ]: 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
```

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/