Лабораторная работа № 7

Введение в работу сданными

Беличева Дарья Михайловна

Содержание

Список иллюстраций

1 Цель работы

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

2 Задание

1. Используя JupyterLab, повторите примерыи. При этом дополните графики обозначениями осей координат, легендой с названиями траекторий, названиями графиков и т.п.

2. Выполните задания для самостоятельной работы.

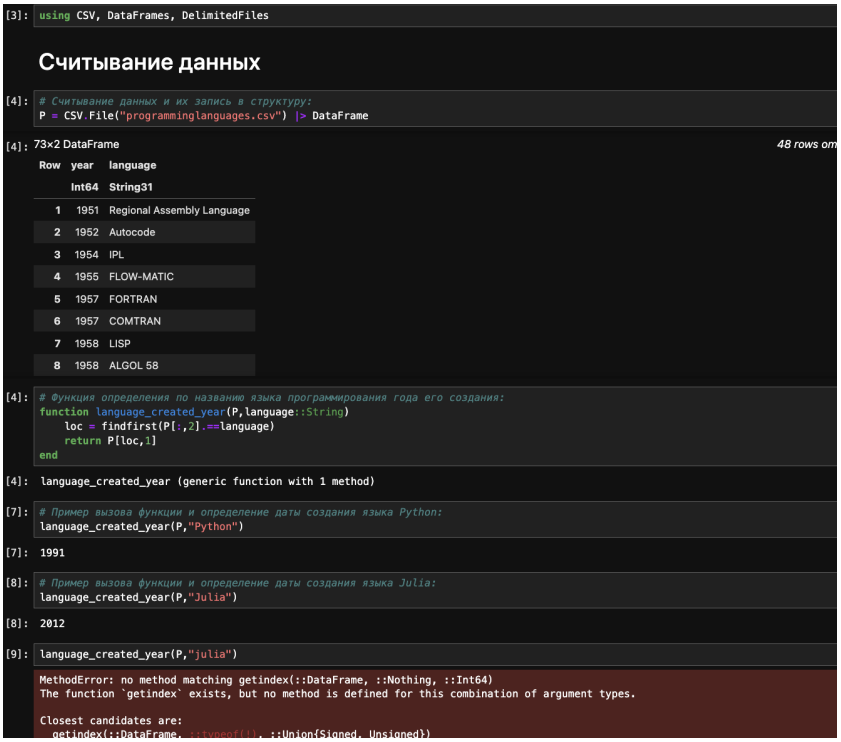
3 Теоретическое введение

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].

4 Выполнение лабораторной работы

Выполним примеры из лабораторной работы (рис. 4.1-4.13).



```
[3]: using CSV, DataFrames, DelimitedFiles
```

Считывание данных

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

[4]: 73×2 DataFrame — 48 rows om

Row	year	language
	Int64	String31
1	1951	Regional Assembly Language
2	1952	Autocode
3	1954	IPL
4	1955	FLOW-MATIC
5	1957	FORTRAN
6	1957	COMTRAN
7	1958	LISP
8	1958	ALGOL 58

```
[4]: # 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
```

```
[4]: language_created_year (generic function with 1 method)
```

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

```
[7]: 1991
```

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

```
[8]: 2012
```

```
[9]: 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})
```

Рис. 4.1: Считывание данных

Запись данных в файл

[14]:
```
# Запись данных в CSV-файл:
CSV.write("programming_languages_data2.csv", P)
```

[14]:
```
"programming_languages_data2.csv"
```

[15]:
```
# Пример записи данных в текстовый файл с разделителем ',':
writedlm("programming_languages_data.txt", Tx, ',')
```

[16]:
```
# Пример записи данных в текстовый файл с разделителем '-':
writedlm("programming_languages_data2.txt", Tx, '-')
```

[17]:
```
# Построчное считывание данных с указанием разделителя:
P_new_delim = readdlm("programming_languages_data2.txt",'-')
```

[17]:
```
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. 4.2: Запись данных в файл

Словари

[18]:
```
# Инициализация словаря:
dict = Dict{Integer,Vector{String}}()
# Инициализация словаря:
dict2 = Dict()
```

[18]:
```
Dict{Any, Any}()
```

[19]:
```
# 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
```

[20]:
```
# Пример определения в словаре языков программирования, созданных в 2003 году:
dict[2003]
```

[20]:
```
2-element Vector{String}:
 "Groovy"
 "Scala"
```

Рис. 4.3: Словари

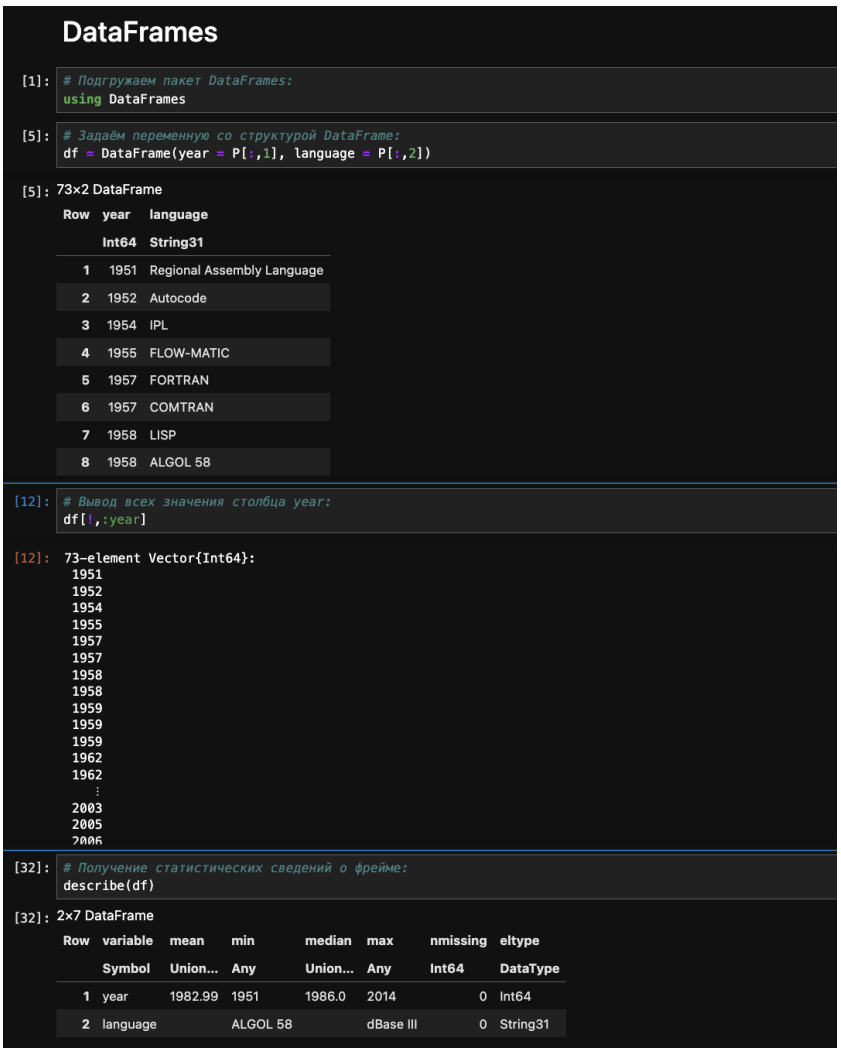


Рис. 4.4: DataFrames

RDatasets

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

```
[34]: # Задаём структуру данных в виде набора данных:
iris = dataset("datasets", "iris")
```

[34]: 150×5 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
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

```
[37]: # Определения типа переменной:
typeof(iris)
```

[37]: DataFrame

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

[38]: 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}

Рис. 4.5: RDatasets

Работа с переменными отсутствующего типа (Missing Values)

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

[40]: Missing

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

[41]: missing

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

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

```
[44]: # Подключаем пакет Statistics:
using Statistics
# Определение среднего значения:
mean(calories)
```

[44]: missing

```
[45]: # Определение среднего значения без значений с отсутствующим типом:
mean(skipmissing(calories))
```

[45]: 58.0

```
[49]: # 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 = outerjoin(dataframe_calories,dataframe_prices,on=:item)
```

[49]: 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. 4.6: Работа с переменными отсутствующеготипа (MissingValues)

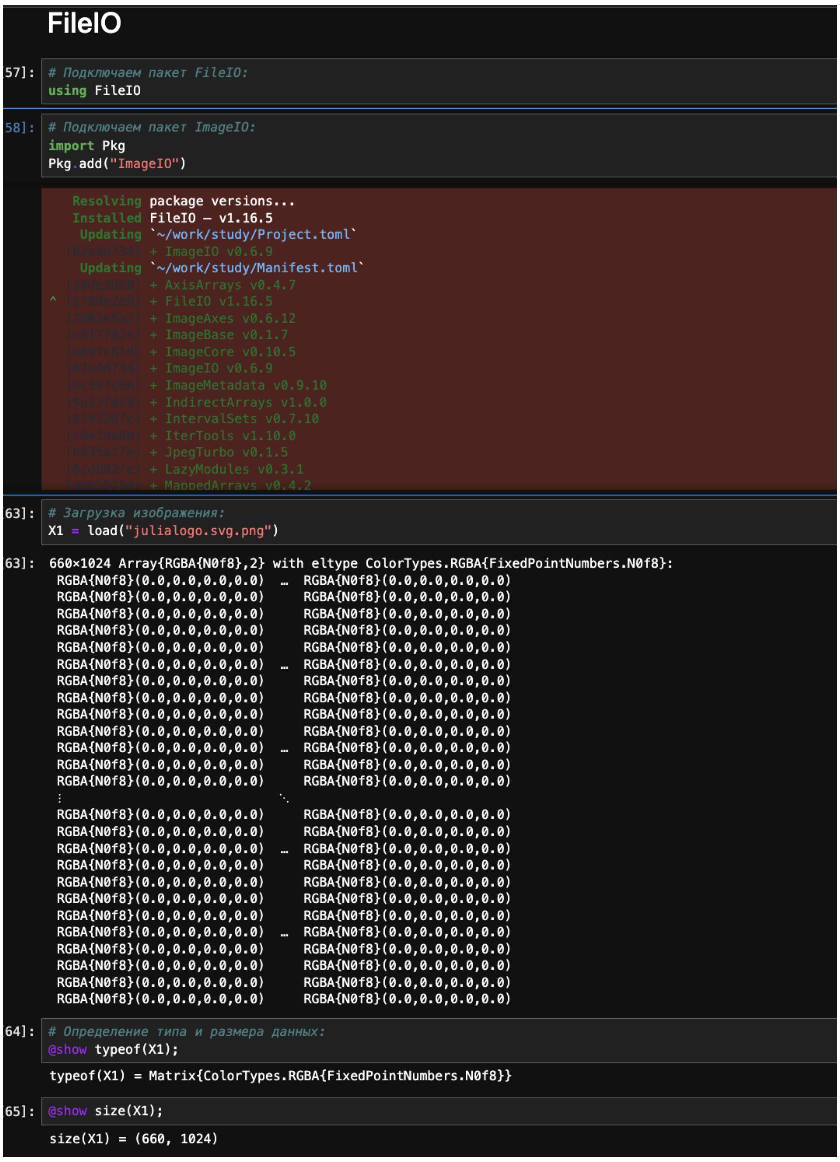


Рис. 4.7: FileIO

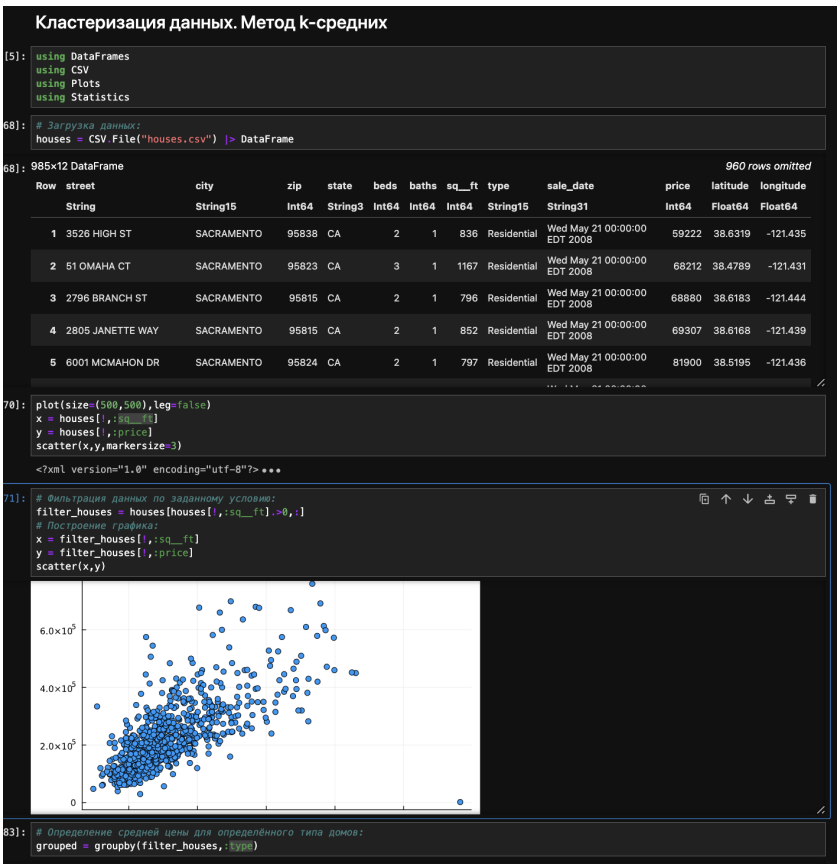


Рис. 4.8: Кластеризация данных. Метод k-средних

```
85]: using Clustering

48]: # 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)
```

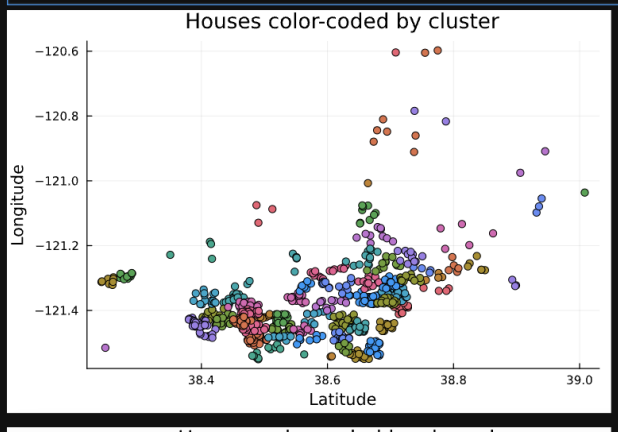


Рис. 4.9: Кластеризация данных. Метод k-средних

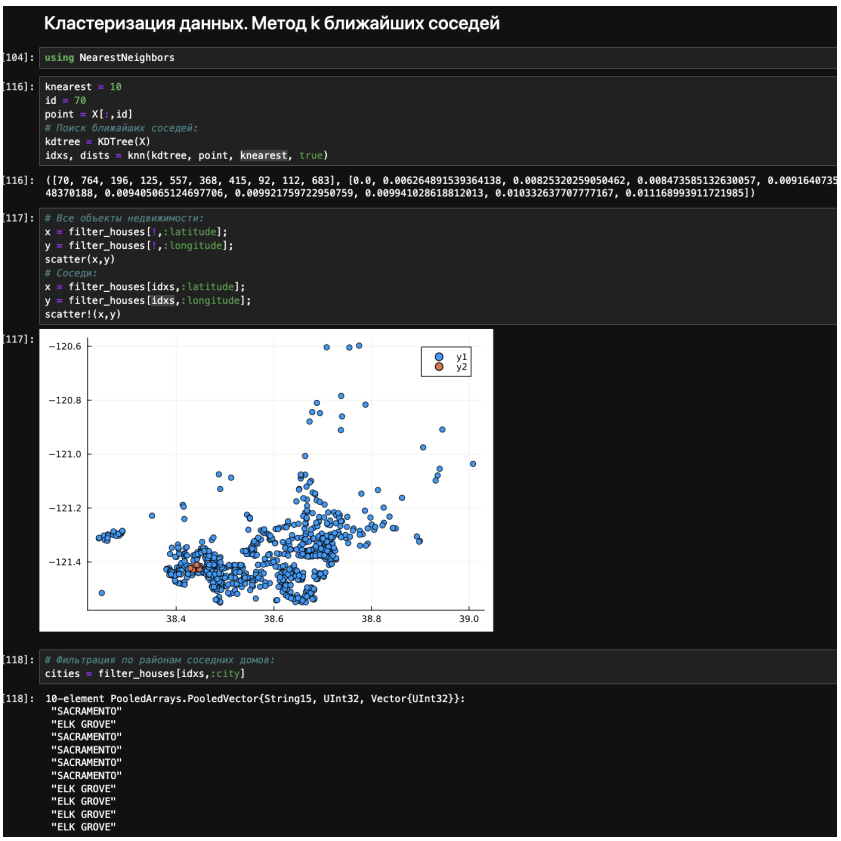


Рис. 4.10: ластеризация данных. Метод k ближайших соседей

Обработка данных. Метод главных компонент

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

```
[24]: 1×814 adjoint(::Vector{Int64}) with eltype Int64:
       59222  68212  68880  69307  81900  89921  …  234000  235000  235301  235738
```

```
[25]: using MultivariateStats
```

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

```
[26]: 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
      ──────────────────────────────────────
```

```
[27]: # 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,:])
```

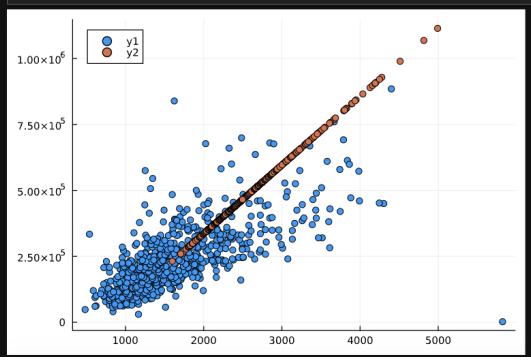


Рис. 4.11: Обработка данных. Метод главных компонент

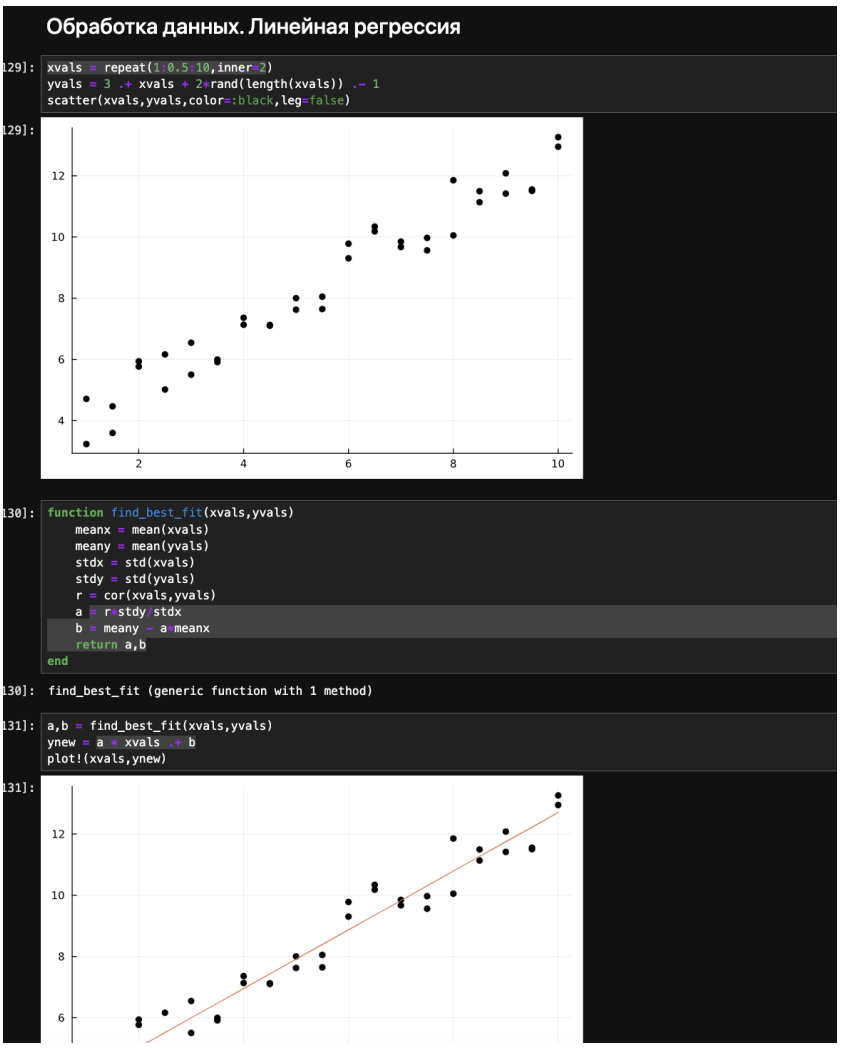


Рис. 4.12: Обработка данных. Линейная регрессия

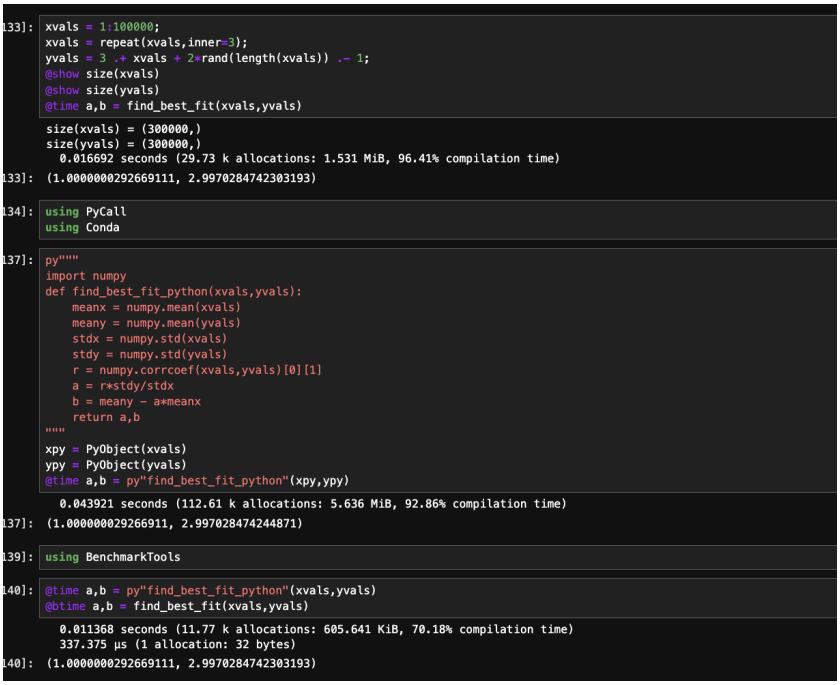


Рис. 4.13: Обработка данных. Линейная регрессия

Теперь выполним задания для самостоятельный работы.

Загрузим

```
using RDatasets
iris = dataset("datasets", "iris")
```

Используем Clustering.jl для кластеризации на основе k-средних. Сделаем точечную диаграмму полученных кластеров. Для этого проиндексируем фрейм данных, преобразуем его в массив и транспонируем (рис. 4.14).

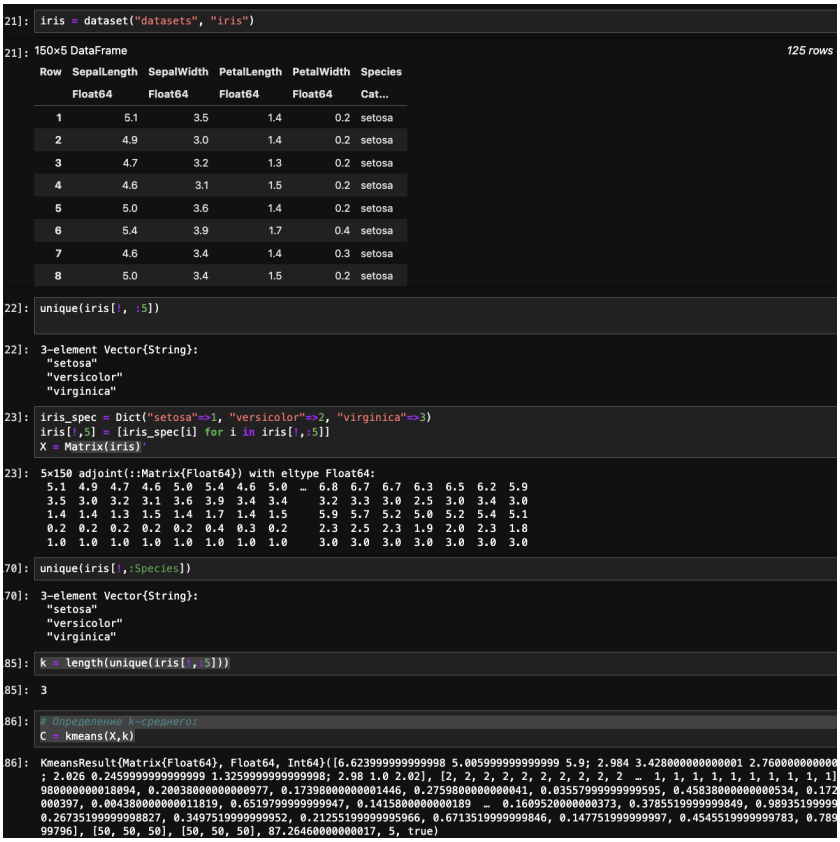


Рис. 4.14: Кластеризация

Рис. 4.15: Кластеризация

Пусть регрессионная зависимость являетсял инейной. Матрица наблюдений факторов X имеет размерность $N \times 3$ `randn (N, 3)`, массив результато в $N \times 1$, регрессионная зависимость является линейной.Найдем МНК-оценку для линейной модели.

- Сравним свои результаты с результатами использования `llsq` из `MultivariateStats.jl`.
- Сравним свои результаты с результатамии спользования регулярной регрессии наименьших квадратов из `GLM.jl`.

Создадим матрицу данных $X2$, которая добавляет столбец единиц в начало матрицы данных, и решим систему линейных уравнений.

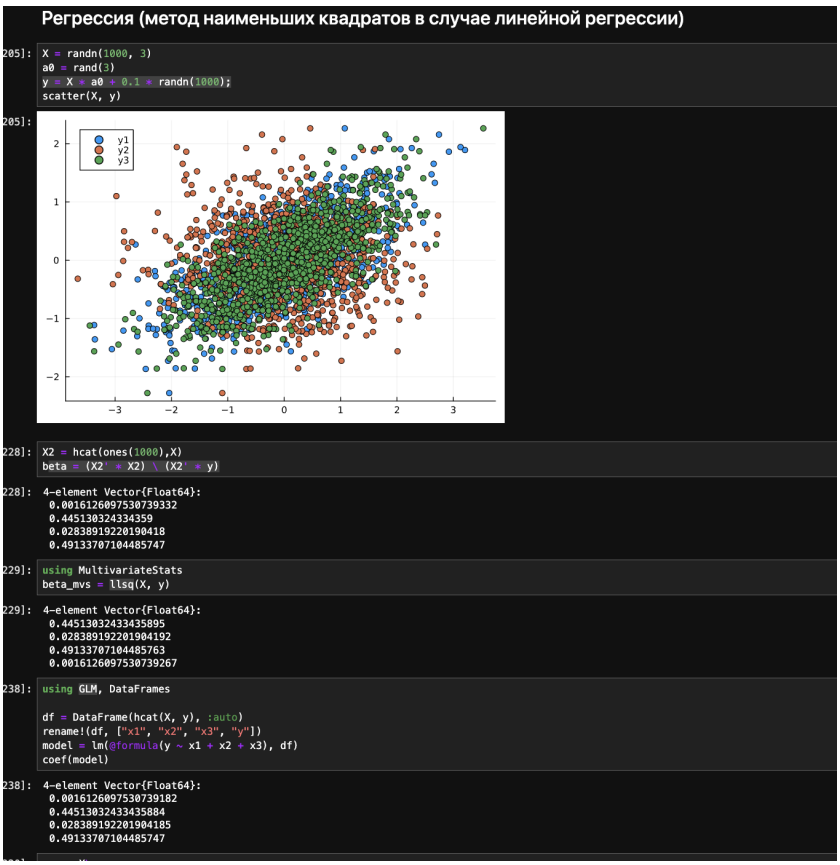


Рис. 4.16: Регрессия

Найдем линию регрессии, используя данные (X, y). Построем график (X, y), используя точечный график. Добавим линию регрессии, используя abline!. Добавим заголовок «График регрессии» и подпишим оси x и y.

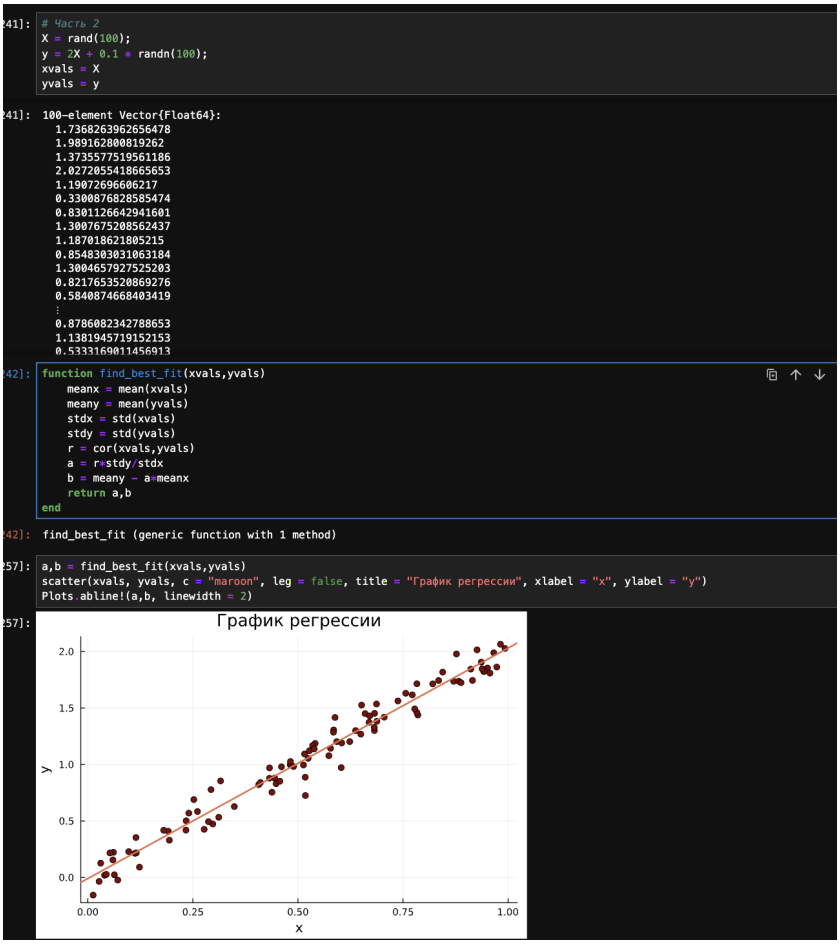


Рис. 4.17: Регрессия

Построим траекторию возможных цен на акции:

- S – начальная цена акции;
- T – длина биномиального дерева в годах;
- n – количество периодов;
- $h = Tn$ – длина одного периода;
- σ – волатильность акции;
- r – годовая процентная ставка;
- $u = \exp(rh + \sigma\sqrt{h})$;
- $d = \exp(rh - \sigma\sqrt{h})$;
- $p^* = \dfrac{\exp(rh) - d}{u - d}$;

Пусть $S = 100$, $T = 1$, $n = 10000$, $\sigma = 0.3$ и $r = 0.08$. Попробуем построить траекторию курса акций.

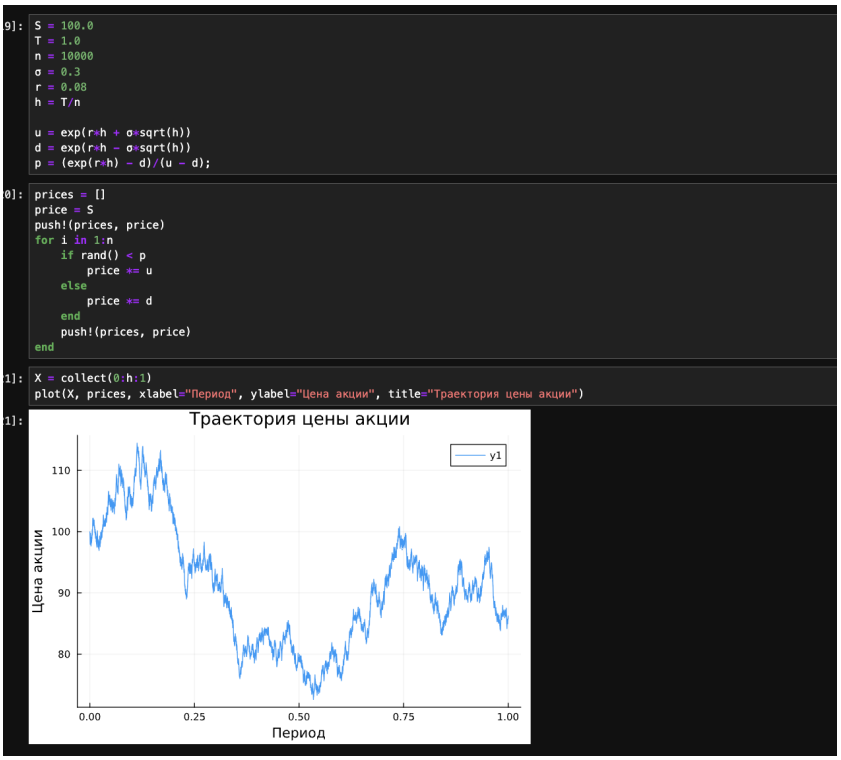


Рис. 4.18: Модель ценообразования биномиальных опционов

Создадим функцию `createPath (S ::Float64, r ::Float64, sigma ::Float64, T ::Float64, n ::Int64)`, которая создает траекторию цены ак- ции с учетом начальных параметров. Используем `createPath`, чтобы создать 10 разных траекторий и построим их все на одном графике.

```
[24]: function createPath(S::Float64, r::Float64, σ::Float64, T::Float64, n::Int64)
          h = T/n
          X = collect(0:h:1)
          u = exp(r*h + σ*sqrt(h))
          d = exp(r*h - σ*sqrt(h))
          p = (exp(r*h) - d)/(u - d);
          prices = []
          price = S
          push!(prices, price)
          for i in 1:n
              if rand() < p
                  price *= u
              else
                  price *= d
              end
              push!(prices, price)
          end
          return (X, prices)
      end

[24]: createPath (generic function with 1 method)

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

[25]:
```

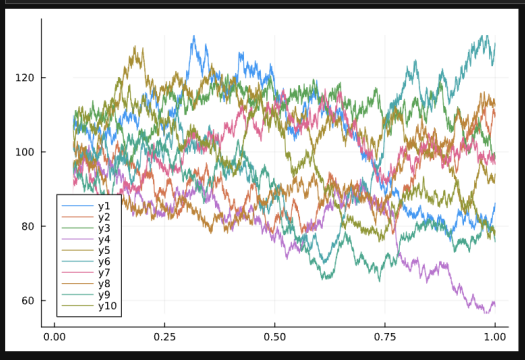


Рис. 4.19: Модель ценообразования биномиальных опционов

Распараллелим генерацию траектории. Можем использовать `Threads.@threads`, `pmap` и `@parallel`.

```
[26]: function createPath_parall(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 = (exp(r*h) - d)/(u - d);
          prices = []
          price = S
          push!(prices, price)
          Threads.@threads for i in 1:n
              if rand() < p
                  price *= u
              else
                  price *= d
              end
              push!(prices, price)
          end
          return prices
      end

[26]: createPath_parall (generic function with 1 method)

[28]: p = plot()
      for i=1:10
          plot!(p, createPath_parall(S, r, σ, T, n))
      end
      p
```

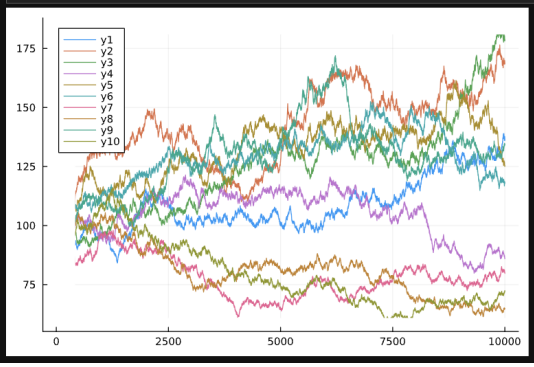


Рис. 4.20: Модель ценообразования биномиальных опционов

5 Выводы

В результате выполнения данной лабораторной работы я освоила специализированные пакеты Julia для обработки данных.

Список литературы

::: {#ref}s :::

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).