Лабораторная работа №7

Введение в работу с данными

Ким Реачна[1]

20 декабря, 2023, Москва, Россия

[1]Российский Университет Дружбы Народов

Цели и задачи

Цель лабораторной работы

Основной целью работы является специализированных пакетов Julia для обработки данных.

Задание

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

Процесс выполнения лабораторной работы

Считывание данных

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

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

[3]: 73×2 DataFrame *48 rows omitted*

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)

[5]: # Пример вызова функции и определение даты создания языка Python:
     print(language_created_year(P,"Python"))
     # Пример вызова функции и определение даты создания языка Julia:
     language_created_year(P,"Julia")

     1991
[5]: 2012
```

Рис. 1: Примеры считывания данных

Запись данных в файл

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

[10]: "programming_languages_data2.csv"

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

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

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

[13]: 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. 2: Примеры записи данных в файл

Словари

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

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

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

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

```
[16]: # 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
```

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

```
[17]: 2-element Vector{String}:
       "Groovy"
       "Scala"
```

Рис. 3: Примеры словаря

DataFrames

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

[19]:
```
# Задаём переменную со структурой DataFrame:
df = DataFrame(year = P[:,1], language = P[:,2])
```

[19]: 73×2 DataFrame *48 rows omitted*

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
9	1959	FACT
10	1959	COBOL
11	1959	RPG
12	1962	APL
13	1962	Simula
⋮	⋮	⋮
62	2003	Scala
63	2005	F#
64	2006	PowerShell
65	2007	Clojure
66	2009	Go
67	2010	Rust
68	2011	Dart
69	2011	Kotlin
70	2011	Red
71	2011	Elixir
72	2012	Julia
73	2014	Swift

Рис. 4: Примеры DataFrames

RDatasets

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

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

[22]: 150×5 DataFrame *125 rows omitted*

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
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
⋮	⋮	⋮	⋮	⋮	⋮
139	6.0	3.0	4.8	1.8	virginica
140	6.9	3.1	5.4	2.1	virginica
141	6.7	3.1	5.6	2.4	virginica
142	6.9	3.1	5.1	2.3	virginica
143	5.8	2.7	5.1	1.9	virginica
144	6.8	3.2	5.9	2.3	virginica
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

Рис. 5: Примеры RDatesets

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]

[27]: 4-element Vector{Union{Missing, Int64}}:
          missing
       47
       22
      105

[28]: # Определение типа переменной:
      typeof(calories)

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

[29]: # Подключаем пакет Statistics:
      using Statistics

      # Определение среднего значения:
      mean(calories)

[29]: missing
```

Рис. 6: Примеры работы с Missing Values

FileIO

```
[35]: # Загрузка изображения:
      X1 = load("julialogo.png")
```

```
[35]: 460×460 Array{RGBA{N0f8},2} with eltype ColorTypes.RGBA{FixedPointNumbers.N0f8}:
 RGBA{N0f8}(0.0,0.0,0.0,0.0)  …  RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)  …  RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)  …  RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 ⋮                            ⋱
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)  …  RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)  …  RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
 RGBA{N0f8}(0.0,0.0,0.0,0.0)     RGBA{N0f8}(0.0,0.0,0.0,0.0)
```

```
[36]: # Определение типа и размера данных:
      @show typeof(X1);
      @show size(X1);
```

```
typeof(X1) = Matrix{ColorTypes.RGBA{FixedPointNumbers.N0f8}}
size(X1) = (460, 460)
```

Рис. 7: Примеры FileIO

Кластеризация данных. Метод k-средних

[38]:
```
# Построение графика:
using Plots
plot(size=(500,500),leg=false)
x = houses[!,:sqft]
y = houses[!,:price]
scatter(x,y,markersize=3)
```

[38]:

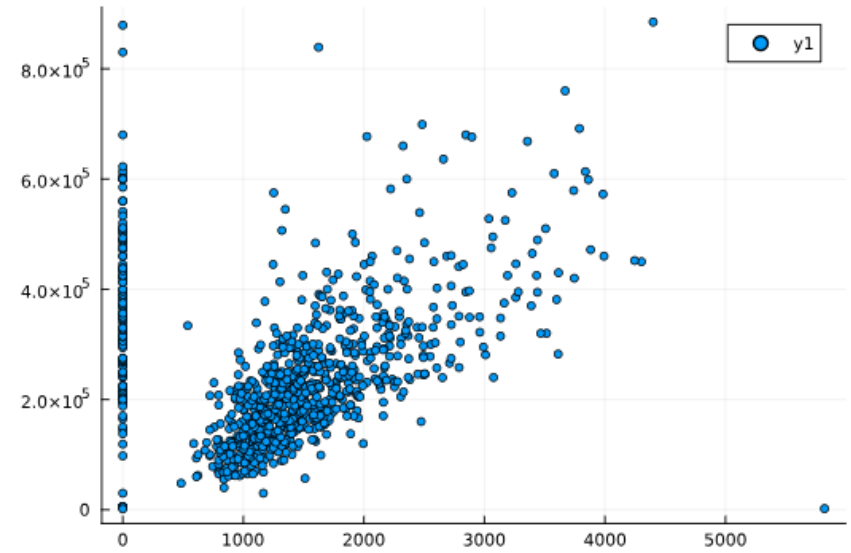
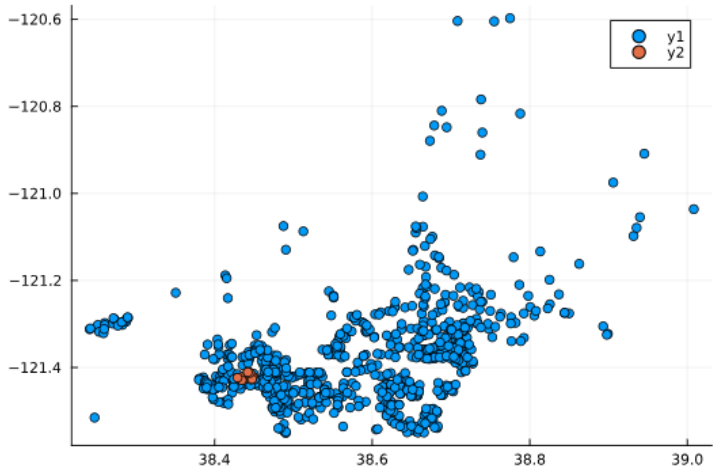


Рис. 8: График цен на недвижимость в зависимости от площади

Кластеризация данных. Метод k ближайших соседей

```
[53]: # 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)
```

[53]:




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

```
[54]: 10-element PooledArrays.PooledVector{String15, UInt32, Vector{UInt32}}:
 "SACRAMENTO"
 "ELK GROVE"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "ELK GROVE"
 "ELK GROVE"
 "ELK GROVE"
 "ELK GROVE"
```

Рис. 9: График определение соседей объекта недвижимости

Обработка данных. Метод главных компонент

```
[60]: # Построение графика с выделением главных компонент:
scatter(F[1,:],F[2,:])
scatter!(Xr[1,:],Xr[2,:])
```

[60]:

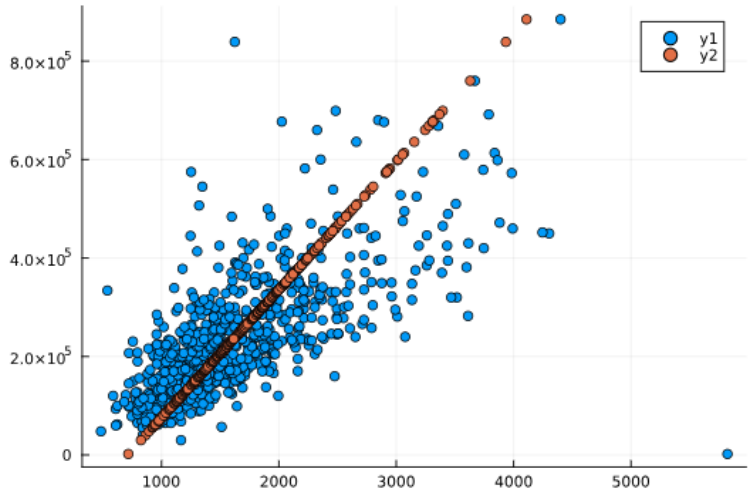
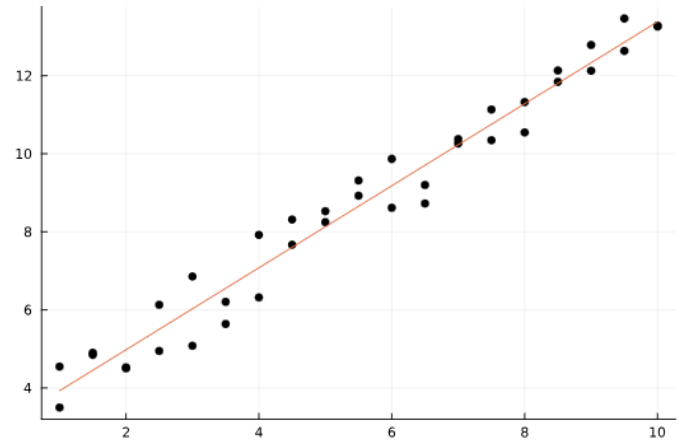


Рис. 10: Определение главных компонент для данных по объектам недвижимости

Обработка данных. Линейная регрессия

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

[63]:




```
[64]: xvals = 1:100000;
      xvals = repeat(xvals,inner=3);
      yvals = 3 .+ xvals + 2*rand(length(xvals)) .- 1;
```

```
[65]: @show size(xvals)
      @show size(yvals)
```

```
size(xvals) = (300000,)
size(yvals) = (300000,)
```

[65]: (300000,)

```
[66]: @time a,b = find_best_fit(xvals,yvals)
```

```
  0.034630 seconds (23.13 k allocations: 1.506 MiB, 95.08% compilation time)
```

[66]: (1.000000022488431, 2.9988226057685097)

```
[67]: import Pkg
      Pkg.add("PyCall")
      Pkg.add("Conda")
      using PyCall
      using Conda
```

```
  Resolving package versions...
 No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
 No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
```

Рис. 11: Линейная регрессия

Задания для самостоятельного выполнения - Кластеризация

[82]:
```
unique_species = unique(iris[!, :Species])
species_figure = plot(legend=false)
for uspecies in unique_species
    iris_sp = iris[iris[!, :Species].==uspecies,:]
    x = iris_sp[!, :SepalLength]
    y = iris_sp[!, :PetalLength]
    scatter!(species_figure, x, y)
end
xlabel!("SepalLength")
ylabel!("PetalLength")
title!("Iris color-coded by species")
display(species_figure)
```

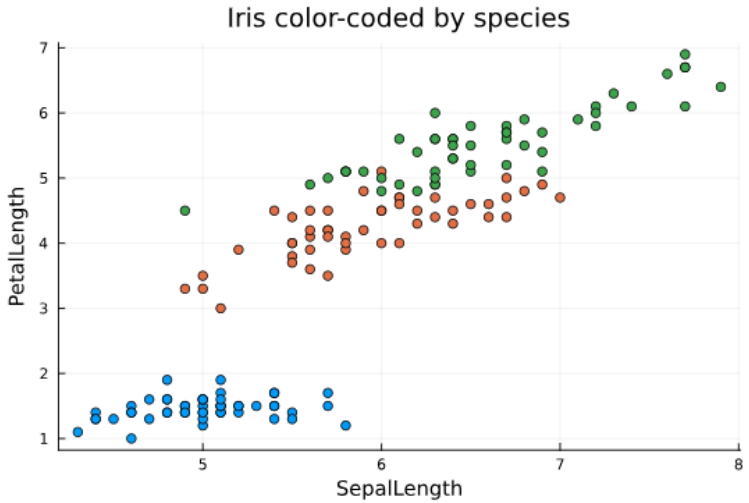


Рис. 12: График Iris color-coded by species

Задания для самостоятельного выполнения - Регрессия

```
[90]: # Часть 2
X = rand(100);
y = 2X + 0.1 * randn(100);
```

```
[91]: a, b = find_best_fit(X,y)
ynew = a * X .+ b
scatter(X, y, title="График регрессии", xlabel="X", ylabel="y", color=:lightblue, leg=false, line=:scatter)
Plots.abline!(a, b, line=:solid)
```

[91]:

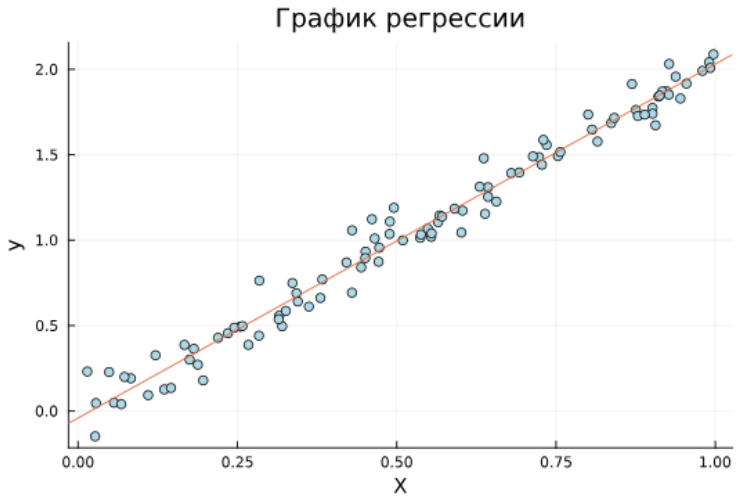


Рис. 13: Линия регрессии

Модель ценообразования биномиальных опционов

[92]:
```
S = 100
T = 1
n = 10000
sigma = 0.3
r = 0.08

h = T / n # длина одного периода;
u = exp(r*h + sigma * sqrt(h))
d = exp(r*h - sigma * sqrt(h))
p = (exp(r*h) - d)/(u - d)

j = 0
stockTree = []
append!(stockTree, S)
for i in 1:n
    k = rand()
    if k < p
        append!(stockTree, S * (u^(i - j)) * (d ^ j))
    else
        j = j + 1
        append!(stockTree, S * (u^(i - j)) * (d ^ j))
    end
end
```

[93]:
```
using Plots
plot(stockTree, title="Траектория курса акций", xlabel="Длина биномиального дерева в годах",
    ylabel="Курс акций", leg=false)
```

[93]:

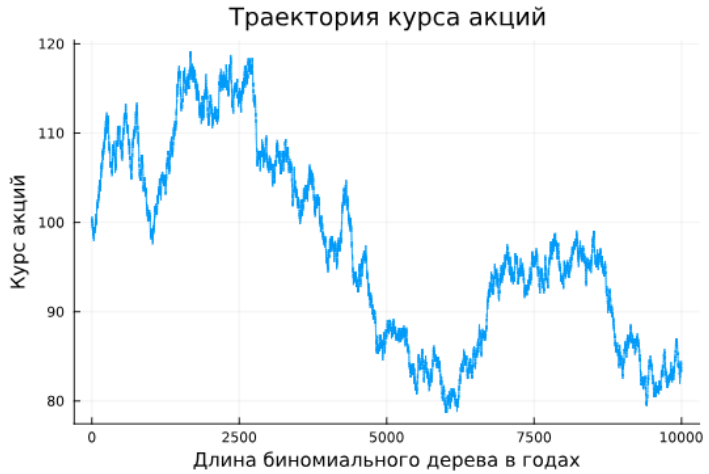


Рис. 14: Решения и график

Выводы по проделанной работе

Вывод

Я специализировала пакетов Julia для обработки данных.