Отчёт по лабораторной работе №7

Введение в работу с данными

Ким Реачна

Содержание

1 Цель работы **5**

2 Выполнение лабораторной работы **6**
2.1 Julia для науки о данных 6
2.1.1 Считывание данных 6
2.1.2 Запись данных в файл 9
2.1.3 Словари 9
2.1.4 DataFrames 10
2.1.5 RDatasets 12
2.1.6 Работа с переменными отсутствующего типа (Missing Values) 13
2.1.7 FileIO 14
2.2 Обработка данных: стандартные алгоритмы машинного обучения в Julia 15
2.2.1 Кластеризация данных. Метод k-средних 15
2.2.2 Кластеризация данных. Метод k ближайших соседей 20
2.2.3 Обработка данных. Метод главных компонент 22
2.2.4 Обработка данных. Линейная регрессия 24
2.3 Задания для самостоятельного выполнения 26
2.3.1 Кластеризация 26
2.3.2 Регрессия (метод наименьших квадратов в случае линейной регрессии) 30
2.3.3 Модель ценообразования биномиальных опционов 32

3 Листинги программы **36**

4 Вывод **51**

Список иллюстраций

2.1 Установка пакетов для будущей работы 6
2.2 Примеры считывания данных 7
2.3 Примеры считывания данных 8
2.4 Примеры записи данных в файл 9
2.5 Примеры словаря 9
2.6 Примеры DataFrames 10
2.7 Примеры DataFrames 11
2.8 Примеры RDatesets 12
2.9 Примеры RDatesets 12
2.10 Примеры работы с Missing Values 13
2.11 Примеры работы с Missing Values 13
2.12 Установка пакет FileIO 14
2.13 Примеры FileIO 14
2.14 Примеры кластеризации метода k-средних 15
2.15 График цен на недвижимость в зависимости от площади 16
2.16 График цен на недвижимость в зависимости от площади (исключены артефакты данных) 16
2.17 Примеры кластеризации метода k-средних 17
2.18 Примеры кластеризации метода k-средних 18
2.19 Примеры кластеризации метода k-средних 19
2.20 Пример кластеризации объектов недвижимости по географическому расположению 19
2.21 Пример кластеризации объектов недвижимости по почтовому индексу 20
2.22 Примеры кластеризации метода k ближайших соседей 20
2.23 График определение соседей объекта недвижимости 21
2.24 Примеры метода главных компонент 22
2.25 Примеры метода главных компонент 23
2.26 Определение главных компонент для данных по объектам недвижимости 23
2.27 Исходные данные 24
2.28 Линейная регрессия 25
2.29 Примеры линейная регрессия 25
2.30 Загрузка данных 26
2.31 График распределения признаков SepalLength и PetalLength . . . 27
2.32 Добавление данных в новый фрейм 27
2.33 Преобразование данных и транспонирование 28

2.34 Создание фрейма данных с указанием кластера 29
2.35 График Iris с цветовой кодировкой по кластеру 29
2.36 График Iris color-coded by species 30
2.37 Данные и функция для решения систем уравнений 31
2.38 Установка необходимый пакет и создаем фрейм данных 31
2.39 Линия регрессии 32
2.40 Решения и график 33
2.41 Функция createPath 34
2.42 График 10 разных траекторий 34
2.43 Распараллеливание генерации траектории 35

1 Цель работы

Основной целью работы является специализированных пакетов Julia для обработки данных.

2 Выполнение лабораторной работы

2.1 Julia для науки о данных

2.1.1 Считывание данных

[1]:
```
# Обновление окружения:
using Pkg
Pkg.update

# Установка пакетов:
using Pkg
for p in ["CSV", "DataFrames", "RDatasets", "FileIO"]
Pkg.add(p)
end
```

```
    Updating registry at `C:\Users\Reachna\.julia\registries\General.toml`
   Resolving package versions...
   Installed InlineStrings ─── v1.4.0
   Installed PooledArrays ──── v1.4.3
   Installed WorkerUtilities ─ v1.6.1
   Installed WeakRefStrings ── v1.4.2
   Installed SentinelArrays ── v1.4.1
   Installed FilePathsBase ─── v0.9.21
   Installed CSV ────────────── v0.10.11
    Updating `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
  [336ed68f] + CSV v0.10.11
    Updating `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
  [336ed68f] + CSV v0.10.11
  [944b1d66] + CodecZlib v0.7.3
  [48062228] + FilePathsBase v0.9.21
  [842dd82b] + InlineStrings v1.4.0
  [2dfb63ee] + PooledArrays v1.4.3
  [91c51154] + SentinelArrays v1.4.1
  [3bb67fe8] + TranscodingStreams v0.10.2
  [ea10d353] + WeakRefStrings v1.4.2
  [76eceee3] + WorkerUtilities v1.6.1
Precompiling project...
  ✓ WorkerUtilities
  ✓ PooledArrays
```

Рис. 2.1: Установка пакетов для будущей работы

```
[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. 2.2: Примеры считывания данных

```
[6]: language_created_year(P,"julia")
```

```
MethodError: no method matching getindex(::DataFrame, ::Nothing, ::Int64)

Closest candidates are:
  getindex(::DataFrame, ::typeof(!), ::Union{Signed, Unsigned})
   @ DataFrames C:\Users\Reachna\.julia\packages\DataFrames\58MUJ\src\dataframe\dataframe.jl:548
  getindex(::DataFrame, ::Colon, ::Union{AbstractString, Signed, Symbol, Unsigned})
   @ DataFrames C:\Users\Reachna\.julia\packages\DataFrames\58MUJ\src\dataframe\dataframe.jl:542
  getindex(::DataFrame, ::InvertedIndex, ::Union{AbstractString, Signed, Symbol, Unsigned})
   @ DataFrames C:\Users\Reachna\.julia\packages\DataFrames\58MUJ\src\dataframe\dataframe.jl:538
  ...


Stacktrace:
 [1] language_created_year(P::DataFrame, language::String)
   @ Main .\In[4]:4
 [2] top-level scope
   @ In[6]:1
```

```
[7]: # 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
```

```
[7]: language_created_year_v2 (generic function with 1 method)
```

```
[8]: # Пример вызова функции и определение даты создания языка julia:
     language_created_year_v2(P,"julia")
```

```
[8]: 2012
```

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

```
[9]: 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"
```

Рис. 2.3: Примеры считывания данных

2.1.2 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. 2.4: Примеры записи данных в файл

2.1.3 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. 2.5: Примеры словаря

2.1.4 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. 2.6: Примеры DataFrames

```
[20]: # Вывод всех значения столбца year:
df[!,:year]
```

```
[20]: 73-element Vector{Int64}:
 1951
 1952
 1954
 1955
 1957
 1957
 1958
 1958
 1959
 1959
 1959
 1962
 1962
    ⋮
 2003
 2005
 2006
 2007
 2009
 2010
 2011
 2011
 2011
 2011
 2012
 2014
```

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

[21]: 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. 2.7: Примеры DataFrames

2.1.5 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. 2.8: Примеры RDatesets

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

[23]: DataFrame

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

[24]: 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. 2.9: Примеры RDatesets

2.1.6 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]

[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. 2.10: Примеры работы с Missing Values

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

[30]: 58.0

[31]: # 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)

[31]: 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. 2.11: Примеры работы с Missing Values

2.1.7 FileIO

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

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

```
  Resolving package versions...
  Installed OpenEXR ──────── v0.3.2
  Installed ImageIO ──────── v0.6.7
  Installed Sixel ────────── v0.1.3
  Installed ProgressMeter ── v1.9.0
  Installed PNGFiles ─────── v0.4.3
  Installed JpegTurbo ────── v0.1.5
  Installed Netpbm ───────── v1.1.1
  Installed AxisArrays ───── v0.4.7
  Installed TiffImages ───── v0.6.8
  Installed QOI ──────────── v1.0.0
  Installed ImageMetadata ── v0.9.9
  Installed PkgVersion ───── v0.3.3
  Installed MappedArrays ─── v0.4.2
  Installed PaddedViews ──── v0.5.12
  Installed LazyModules ──── v0.3.1
  Installed MosaicViews ──── v0.3.4
  Installed Imath_jll ────── v3.1.7+0
  Installed ImageCore ────── v0.10.1
```

Рис. 2.12: Установка пакет 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)

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

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

Рис. 2.13: Примеры FileIO

2.2 Обработка данных: стандартные алгоритмы машинного обучения в Julia

2.2.1 Кластеризация данных. Метод k-средних

[36]:
```
# 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 `C:\Users\Reachna\.julia\environments\v1.9\Project.toml`
 No Changes to `C:\Users\Reachna\.julia\environments\v1.9\Manifest.toml`
  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`
  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`
```

[37]:
```
# Загрузка данных:
houses = CSV.File("houses.csv") |> DataFrame
```

[37]: 985×12 DataFrame — *960 rows omitted*

Row	street	city	zip	state	beds	baths	sqft	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

Рис. 2.14: Примеры кластеризации метода 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]:

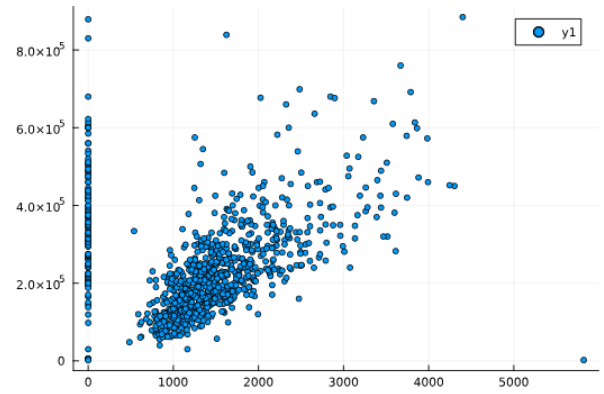


Рис. 2.15: График цен на недвижимость в зависимости от площади

```
[39]: # Фильтрация данных по заданному условию:
filter_houses = houses[houses[!,:sqft].>0,:]
# Построение графика:
x = filter_houses[!,:sqft]
y = filter_houses[!,:price]
scatter(x,y)
```

[39]:

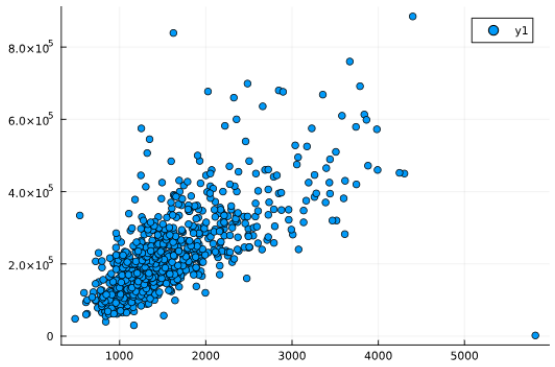


Рис. 2.16: График цен на недвижимость в зависимости от площади (исключены артефакты данных)

[40]:
```
# Подключение пакета Statistics:
using Statistics
# Определение средней цены для определённого типа домов:
combine(groupby(filter_houses,:type),filter_houses->mean(filter_houses[!,:price]))
```

[40]: 3×2 DataFrame

Row	type	x1
	String15	Float64
1	Residential	2.34802e5
2	Condo	1.34213e5
3	Multi-Family	2.24535e5

[41]:
```
# Подключение пакета Clustering:
import Pkg
Pkg.add("Clustering")
using Clustering
```

```
  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`
```

[42]:
```
# Добавление данных :Latitude и :Longitude в новый фрейм:
X = filter_houses[!, [:latitude,:longitude]]
```

[42]: 814×2 DataFrame *789 rows omitted*

Row	latitude	longitude
	Float64	Float64
1	38.6319	-121.435
2	38.4789	-121.431
3	38.6183	-121.444
4	38.6168	-121.439
5	38.5195	-121.436
6	38.6626	-121.328
7	38.6817	-121.352
8	38.5351	-121.481
9	38.6212	-121.271
10	38.7009	-121.443

Рис. 2.17: Примеры кластеризации метода k-средних

```
[43]: # Конвертация данных в матричный вид:
      X = Matrix{Float64}(X)
```

```
[43]: 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
       38.6817  -121.352
       38.5351  -121.481
       38.6212  -121.271
       38.7009  -121.443
       38.6377  -121.452
       38.4707  -121.459
       38.6187  -121.436
        ⋮
       38.7035  -121.375
       38.7031  -121.235
       38.3898  -121.446
       38.8978  -121.325
       38.4679  -121.445
       38.4453  -121.442
       38.4174  -121.484
       38.4577  -121.36
       38.4999  -121.459
       38.7088  -121.257
       38.417   -121.397
       38.6552  -121.076
```

```
[44]: # Транспонирование матрицы с данными:
      X = X'
```

```
[44]: 2×814 adjoint(::Matrix{Float64}) with eltype Float64:
         38.6319    38.4789    38.6183  …     38.7088    38.417    38.6552
       -121.435   -121.431   -121.444       -121.257   -121.397  -121.076
```

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

```
[45]: 66
```

```
[46]: # Определение k-среднего:
      C = kmeans(X,k)
```

```
[46]: KmeansResult{Matrix{Float64}, Float64, Int64}([38.63384813333333 38.46755533333334 … 38.524875375 38.4712824; -121.53378246666665 -121.31786866666668 … -121.4249398125 -121.4590456], [54, 57, 13, 13, 65, 15, 31, 44, 58, 10  …  27, 22, 57, 33, 56, 48, 63, 9, 14, 16], [7.917070615803823e-5, 0.000155379999341676, 7.903687219368294e-6, 2.1914860553806648e-5, 0.00014646772251580842, 0.00017304826178587973, 0.0001565350394230336, 0.00010155051859328523, 0.00043306466977810487, 0.00011686658399412408  …  5.8604102378012261e-5, 5.241325197857804 6e-5, 6.081747051212005e-5, 0.00020461086387513205, 0.00013415837747743353, 8.298793545691296e-5, 4.5475157094188035e-5, 0.00017908342124428 6, 0.0003265923005528748, 7.06132996128872e-5], [15, 3, 9, 3, 13, 6, 3, 9, 11, 13  …  13, 11, 13, 20, 25, 12, 17, 7, 16, 10], [15, 3, 9, 3, 13, 6, 3, 9, 11, 13  …  13, 11, 13, 20, 25, 12, 17, 7, 16, 10], 0.19741556713415775, 21, true)
```

Рис. 2.18: Примеры кластеризации метода k-средних

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

[47]: 814×5 DataFrame — *789 rows omitted*

Row	cluster	city	latitude	longitude	zip
	Int64	String15	Float64	Float64	Int64
1	54	SACRAMENTO	38.6319	-121.435	95838
2	57	SACRAMENTO	38.4789	-121.431	95823
3	13	SACRAMENTO	38.6183	-121.444	95815
4	13	SACRAMENTO	38.6168	-121.439	95815
5	65	SACRAMENTO	38.5195	-121.436	95824
6	15	SACRAMENTO	38.6626	-121.328	95841
7	31	SACRAMENTO	38.6817	-121.352	95842
8	44	SACRAMENTO	38.5351	-121.481	95820
9	58	RANCHO CORDOVA	38.6212	-121.271	95670
10	10	RIO LINDA	38.7009	-121.443	95673
11	62	SACRAMENTO	38.6377	-121.452	95838
12	66	SACRAMENTO	38.4707	-121.459	95823
13	13	SACRAMENTO	38.6187	-121.436	95815
⋮	⋮	⋮	⋮	⋮	⋮
803	30	NORTH HIGHLANDS	38.7035	-121.375	95660
804	9	ORANGEVALE	38.7031	-121.235	95662
805	27	ELK GROVE	38.3898	-121.446	95757
806	22	LINCOLN	38.8978	-121.325	95648
807	57	SACRAMENTO	38.4679	-121.445	95823
808	33	SACRAMENTO	38.4453	-121.442	95823
809	56	ELK GROVE	38.4174	-121.484	95758
810	48	SACRAMENTO	38.4577	-121.36	95829
811	63	SACRAMENTO	38.4999	-121.459	95823

Рис. 2.19: Примеры кластеризации метода k-средних

```
[48]: 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)
```

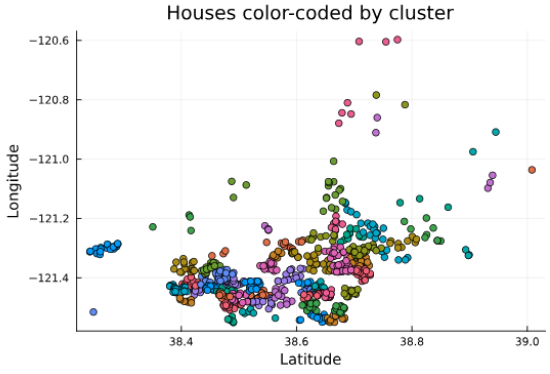


Рис. 2.20: Пример кластеризации объектов недвижимости по географическому расположению

```
[49]: 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)
```

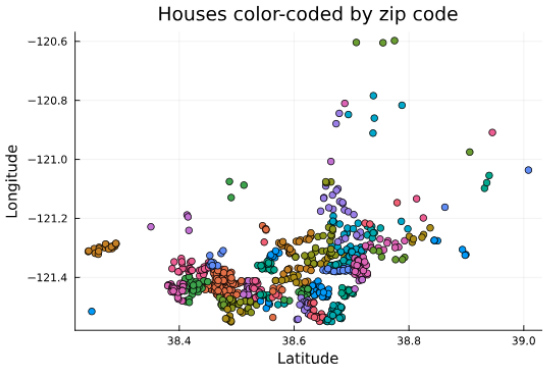


Рис. 2.21: Пример кластеризации объектов недвижимости по почтовому индексу

2.2.2 Кластеризация данных. Метод k ближайших соседей

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

```
  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`
```

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

```
[51]: 2-element Vector{Float64}:
   38.44004
 -121.421012
```

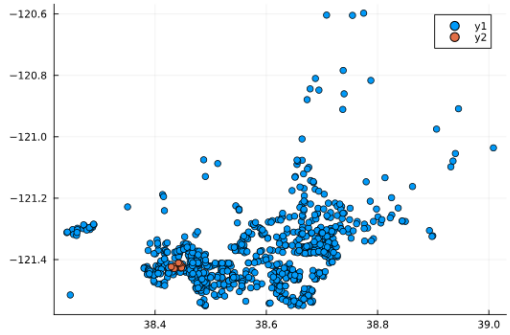
```
[52]: # Поиск ближайших соседей:
kdtree = KDTree(X)
idxs, dists = knn(kdtree, point, knearest, true)
```

```
[52]: ([70, 764, 196, 125, 557, 368, 415, 92, 112, 683], [0.0, 0.0062648915393641 38, 0.00825320259050462, 0.008473585132630057, 0.009164073548370188, 0.009405065124697706, 0.009921759722950759, 0.009941028618812013, 0.010332637707777167, 0.011168993911721985])
```

Рис. 2.22: Примеры кластеризации метода 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. 2.23: График определение соседей объекта недвижимости

2.2.3 Обработка данных. Метод главных компонент

```
[55]: # Фрейм с указанием площади и цены недвижимости:
      F = filter_houses[!, [:sqft,:price]]
```

[55]: 814×2 DataFrame *789 rows omitted*

Row	sqft	price
	Int64	Int64
1	836	59222
2	1167	68212
3	796	68880
4	852	69307
5	797	81900
6	1122	89921
7	1104	90895
8	1177	91002
9	941	94905
10	1146	98937
11	909	100309
12	1289	106250
13	871	106852
⋮	⋮	⋮
803	960	224252
804	1456	225000
805	1450	228000
806	1358	229027
807	1329	229500
808	1715	230000
809	1262	230000
810	2280	232425

Рис. 2.24: Примеры метода главных компонент

```
[56]: # Конвертация данных в массив:
      F = Matrix{Float64}(F)'

[56]: 2×814 adjoint(::Matrix{Float64}) with eltype Float64:
        836.0   1167.0    796.0    852.0    797.0  …    1216.0    1685.0    1362.0
      59222.0  68212.0  68880.0  69307.0  81900.0     235000.0  235301.0  235738.0

[57]: # Подключение пакета MultivariateStats:
      import Pkg
      Pkg.add("MultivariateStats")

        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`

[58]: using MultivariateStats
      # Приведение типов данных к распределению для PCA:
      M = fit(PCA, F)

[58]: 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
      ──────────────────────────────────────

[59]: # Выделение значений главных компонент в отдельную переменную:
      y = MultivariateStats.transform(M, F)

      Xr = reconstruct(M, y)

[59]: 2×814 Matrix{Float64}:
        936.922    971.477    974.039    975.681  …  1613.64         1615.32
      59221.6    68212.8    68879.3    69306.5          2.35301e5  235737.0
```

Рис. 2.25: Примеры метода главных компонент

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

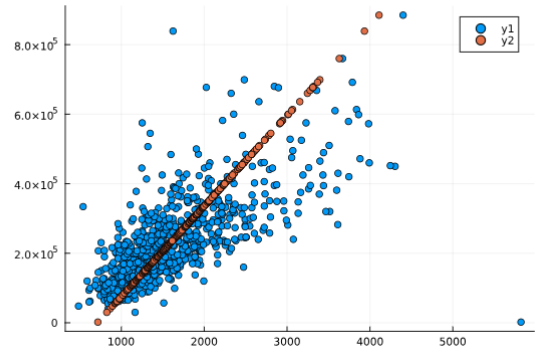
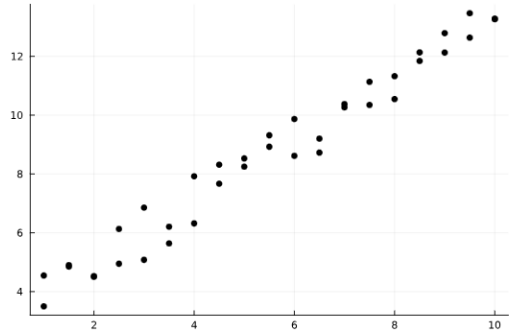


Рис. 2.26: Определение главных компонент для данных по объектам недвижимости

2.2.4 Обработка данных. Линейная регрессия

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

[61]:



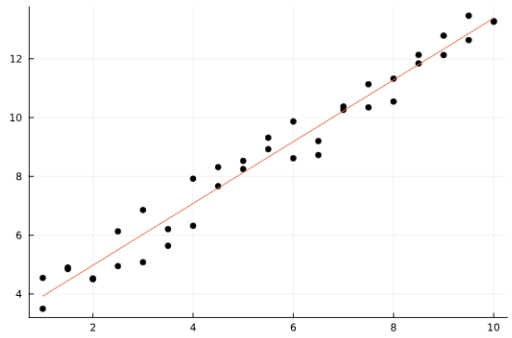

```
[62]: function find_best_fit(xvals,yvals)
          meanx = mean(xvals)
          meany = mean(yvals)
          stdx = std(xvals)
          stdy = std(yvals)
          r = cor(xvals,yvals)
          a = r*stdy/stdx
          b = meany - a*meanx
          return a,b
      end
```

```
[62]: find_best_fit (generic function with 1 method)
```

Рис. 2.27: Исходные данные

```
[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. 2.28: Линейная регрессия

```
[68]: 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
      """

[69]: find_best_fit_python = py"find_best_fit_python"

[69]: PyObject <function find_best_fit_python at 0x000001AD8A4901F0>

[70]: xpy = PyObject(xvals)
      ypy = PyObject(yvals)
      @time a,b = find_best_fit_python(xpy,ypy)

        0.132207 seconds (73.55 k allocations: 5.060 MiB, 75.61% compilation time)
[70]: (1.0000000248843, 2.9988226057903375)

[71]: import Pkg
      Pkg.add("BenchmarkTools")
      using BenchmarkTools

        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`

[72]: @btime a,b = find_best_fit_python(xvals,yvals)
      @btime a,b = find_best_fit(xvals,yvals)

        7.022 ms (27 allocations: 864 bytes)
        1.103 ms (1 allocation: 32 bytes)
[72]: (1.000000022488431, 2.9988226057685097)
```

Рис. 2.29: Примеры линейная регрессия

2.3 Задания для самостоятельного выполнения

2.3.1 Кластеризация

- Загрузка данных

```
[73]: using RDatasets
      iris = dataset("datasets", "iris")
```

[73]: 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

Рис. 2.30: Загрузка данных

- Для кластеризации выбираются 2 атрибута - SepalLength и PetalLength и построение точечного графика их распределения в начале:

```
[74]: plot(size=(600, 600), leg=false)

      x = iris[!, :SepalLength]
      y = iris[!, :PetalLength]
      scatter(x, y, markersize=3, title="Распределение признаков SepalLength и PetalLength",
          xlabel="SepalLength", ylabel="PetalLength", leg=false)
```

[74]:

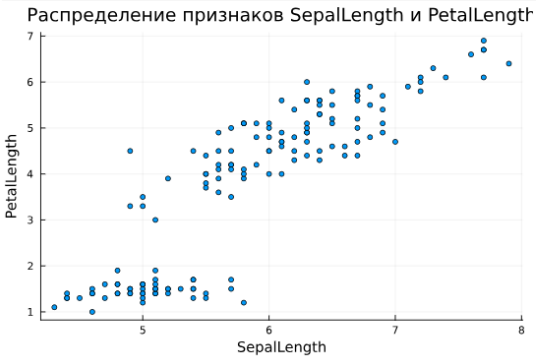


Рис. 2.31: График распределения признаков SepalLength и PetalLength

- Добавим данные в новый фрейм:

```
[75]: X = iris[!, [:SepalLength, :PetalLength]]
```

[75]: 150×2 DataFrame — *125 rows omitted*

Row	SepalLength	PetalLength
	Float64	Float64
1	5.1	1.4
2	4.9	1.4
3	4.7	1.3
4	4.6	1.5
5	5.0	1.4
6	5.4	1.7
7	4.6	1.4
8	5.0	1.5
9	4.4	1.4
10	4.9	1.5
11	5.4	1.5
12	4.8	1.6
13	4.8	1.4
⋮	⋮	⋮
139	6.0	4.8
140	6.9	5.4
141	6.7	5.6
142	6.9	5.1
143	5.8	5.1
144	6.8	5.9
145	6.7	5.7

Рис. 2.32: Добавление данных в новый фрейм

- Преобразуем данные в матричный видб транспонируем и выдвигаем гипотезу, что эти признаки могут зависеть от типа радужной оболочки. Поэтому

подсчитаем количество уникальных видов и возьмем это количество кластеров.

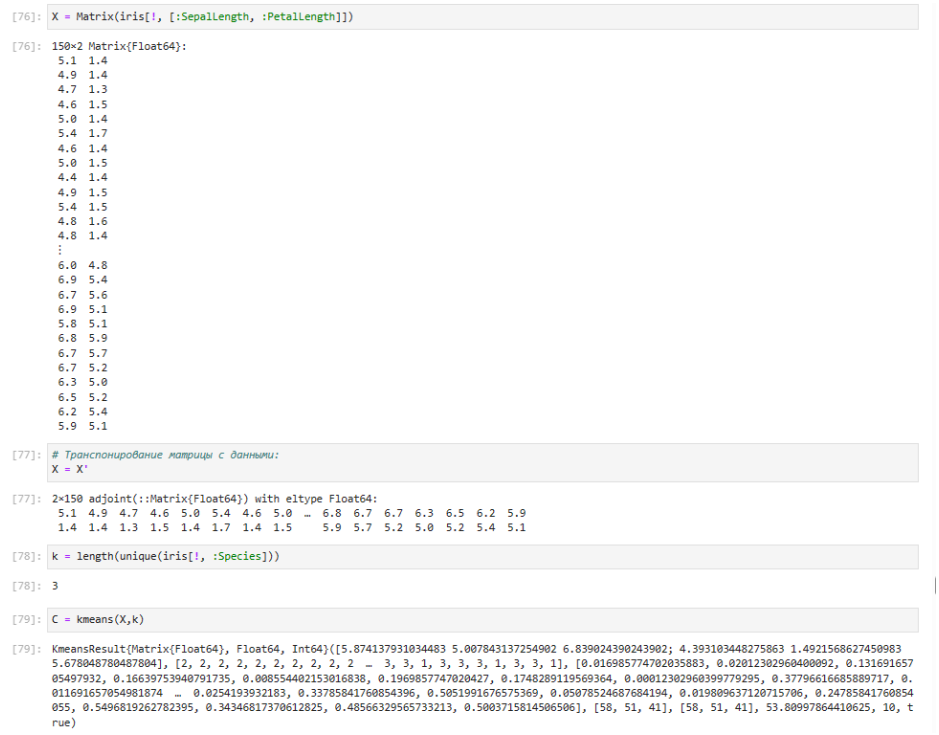


Рис. 2.33: Преобразование данных и транспонирование

- Создание фрейма данных с указанием кластера, построение графика датафрейма с распределением цветов по кластерам и построение графиков с распределением цветов по самому виду Ирисов:

```
[80]: iris_new = DataFrame(cluster = C.assignments,
          SepalLength=iris[!, :SepalLength], PetalLength=iris[!, :PetalLength], Spicies = iris[!, :Species])
```

[80]: 150×4 DataFrame *125 rows omitted*

Row	cluster	SepalLength	PetalLength	Spicies
	Int64	Float64	Float64	Cat...
1	2	5.1	1.4	setosa
2	2	4.9	1.4	setosa
3	2	4.7	1.3	setosa
4	2	4.6	1.5	setosa
5	2	5.0	1.4	setosa
6	2	5.4	1.7	setosa
7	2	4.6	1.4	setosa
8	2	5.0	1.5	setosa
9	2	4.4	1.4	setosa
10	2	4.9	1.5	setosa
11	2	5.4	1.5	setosa
12	2	4.8	1.6	setosa
13	2	4.8	1.4	setosa
⋮	⋮	⋮	⋮	⋮
139	1	6.0	4.8	virginica
140	3	6.9	5.4	virginica
141	3	6.7	5.6	virginica
142	3	6.9	5.1	virginica
143	1	5.8	5.1	virginica
144	3	6.8	5.9	virginica
145	3	6.7	5.7	virginica
146	3	6.7	5.2	virginica
147	1	6.3	5.0	virginica

Рис. 2.34: Создание фрейма данных с указанием кластера

```
[81]: cluster_figure = plot(legend = false)
for i = 1:k
    iris_new_clustered = iris_new[iris_new[!, :cluster].==i,:]
    xvals = iris_new_clustered[!, :SepalLength]
    yvals = iris_new_clustered[!, :PetalLength]
    scatter!(cluster_figure, xvals, yvals, markersize=4)
end
xlabel!("SepalLength")
ylabel!("PetalLength")
title!("Iris color-coded by cluster")
display(cluster_figure)
```

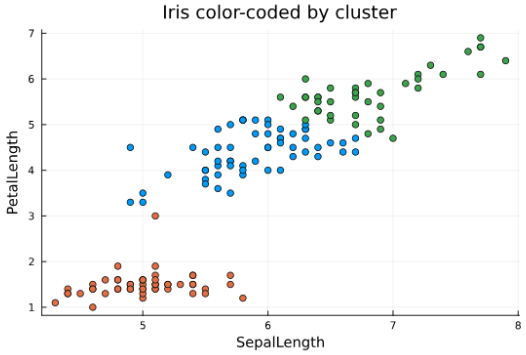


Рис. 2.35: График Iris с цветовой кодировкой по кластеру

```
[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)
```

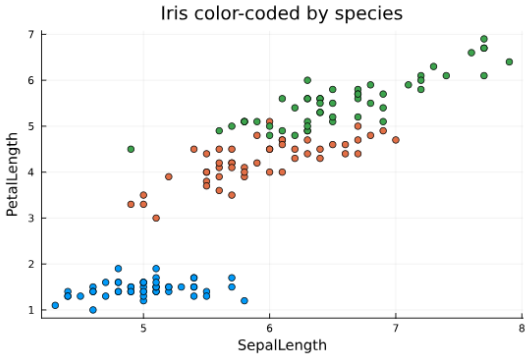


Рис. 2.36: График Iris color-coded by species

2.3.2 Регрессия (метод наименьших квадратов в случае линейной регрессии)

Часть 1: Пусть регрессионная зависимость является линейной. Матрица наблюдений факторов X имеет размерность $N \times 3$ randn $(N, 3)$, массив результатов $N \times 1$, регрессионная зависимость является линейной. Найдите МНК-оценку для линейной модели:

- Записываем данные, затем реализуем функцию linear_regression_model, которая изначально создает матрицу X2, состоящую из единиц. Далее присоединяю этот столбец к X. А затем решаю систему линейных уравнений.

```
[83]: # Часть 1
      X = randn(1000, 3)
      a0 = rand(3)
      print(a0)
      y = X * a0 + 0.1 * randn(1000);

      [0.27569630619990987, 0.11064418518813057, 0.6420620293621074]

[84]: function linear_regression_model(X,y)
          X2 = ones(1000)
          X = hcat(X,X2)
          return X \ y
      end

[84]: linear_regression_model (generic function with 1 method)

[85]: a = linear_regression_model(X,y)
      print(a)

      [0.27186121611415687, 0.11565127842117802, 0.6453067691942215, -0.0006286184372975464]
```

Рис. 2.37: Данные и функция для решения систем уравнений

Результат решения линейных уравнений : -0.000628618.

- Сравню полученные результаты с результатами использования llsq из MultivariateStats.jl и с результатами использования регулярной регрессии наименьших квадратов из GLM.jl. После установки пакета создадим датафрейм, в который запишем y и разобьем X на 3 столбца - x1, x2 и x3.

```
[86]: using MultivariateStats
      # using llsq
      a = llsq(X,y)
      print(a)

      [0.2718612161141568, 0.11565127842117806, 0.6453067691942221, -0.0006286184372975258]

[87]: Pkg.add("GLM")

        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`

[88]: using GLM, DataFrames

[89]: X1 = X[:, 1]
      X2 = X[:, 2]
      X3 = X[:, 3]

      data = DataFrame(y=y, x1=X1, x2=X2, x3=X3);

      lm(@formula(y ~ x1 + x2 + x3), data)

[89]: StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Vector{Float64}}, GLM.DensePredChol{Float64, LinearAlgebra.CholeskyPivoted{Float64,
      Matrix{Float64}, Vector{Int64}}}}, Matrix{Float64}}

      y ~ 1 + x1 + x2 + x3

      Coefficients:
      ──────────────────────────────────────────────────────────────────────────
                          Coef.  Std. Error       t  Pr(>|t|)    Lower 95%   Upper 95%
      ──────────────────────────────────────────────────────────────────────────
      (Intercept)  -0.000628618  0.00316643   -0.20    0.8427  -0.00684225  0.00558501
      x1            0.271861     0.00313727   86.66    <1e-99   0.265705    0.278018
      x2            0.115651     0.00324786   35.61    <1e-99   0.109278    0.122025
      x3            0.645307     0.00330034  195.53    <1e-99   0.63883     0.651783
      ──────────────────────────────────────────────────────────────────────────
```

Рис. 2.38: Установка необходимый пакет и создаем фрейм данных

Как мы видим результат практически идентичен и Результат Intercept во всех 3 случаях также одинаковое -0.000628618.

Часть 2: Найдите линию регрессии, используя данные (X, y). Постройте график (X, y), используя точечный график. Добавьте линию регрессии, используя abline!. Добавьте заголовок «График регрессии» и подпишите оси x и y.

```
[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)
```

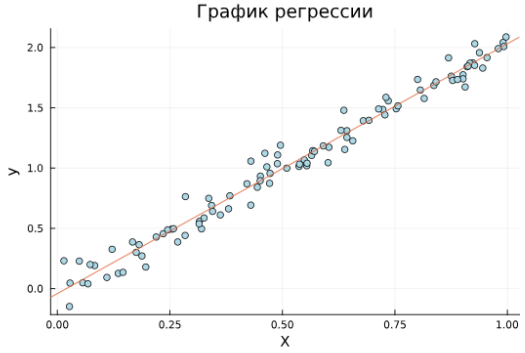


Рис. 2.39: Линия регрессии

2.3.3 Модель ценообразования биномиальных опционов

а) Пусть $S = 100, T = 1, n = 10000, \sigma = 0.3$ и $r = 0.08$. Попробуйте построить траекторию курса акций. Функция rand () генерирует случайное число от 0 до 1:

[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]:

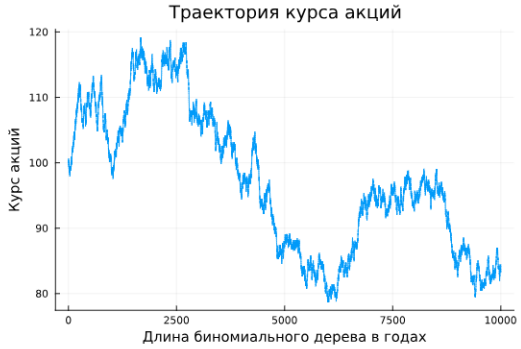


Рис. 2.40: Решения и график

b) Создайте функцию createPath (S :: Float64, r :: Float64, sigma ::Float64, T :: Float64, n :: Int64), которая создает траекторию цены акции с учетом начальных параметров. Используйте createPath, чтобы создать 10 разных траекторий и построить их все на одном графике:

```
[94]: function createPath(S::Float64, T::Float64, n::Int64, sigma::Float64, r::Float64)
          # S - начальная цена акции;
          # T - длина биномиального дерева в годах;
          # n - длина одного периода;
          # sigma - волатильность акции;
          # r – годовая процентная ставка;

          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)
          stockTree = []
          append!(stockTree, S)
          j = 0

          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
          return stockTree
      end

[94]: createPath (generic function with 1 method)
```

Рис. 2.41: Функция createPath

```
[95]: for i in 1:10
          IJulia.clear_output(true)
          traj = createPath(100.0, 1.0, 10000, 0.3, 0.08)
          if i == 1
              p = plot(traj, title="Траектория курса акций", xlabel="Длина биномиального дерева в годах",
                  ylabel="Курс акций", leg=false)
          end
          p = plot!(traj)
          display(p)
      end
```

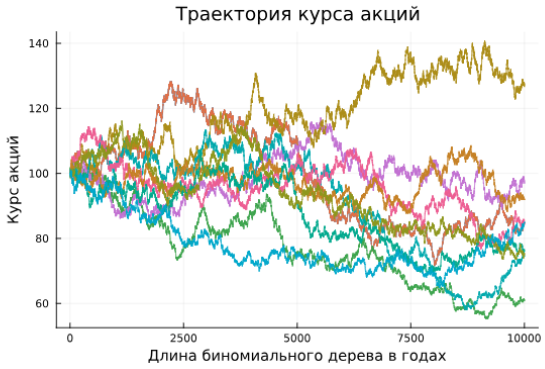


Рис. 2.42: График 10 разных траекторий

c) Распараллельте генерацию траектории. Можете использовать `Threads.@threads`, `pmap` и `@parallel`.

[96]:
```
using Base.Threads
Threads.@threads for i in 1:10
    IJulia.clear_output(true)
    traj = createPath(100.0, 1.0, 10000, 0.3, 0.08)
    if i == 1
        g = plot(traj, title="Траектория курса акций", xlabel="Длина биномиального дерева в годах",
            ylabel="Курс акций", leg=false)
    end
    g = plot!(traj)
    display(g)
end
```

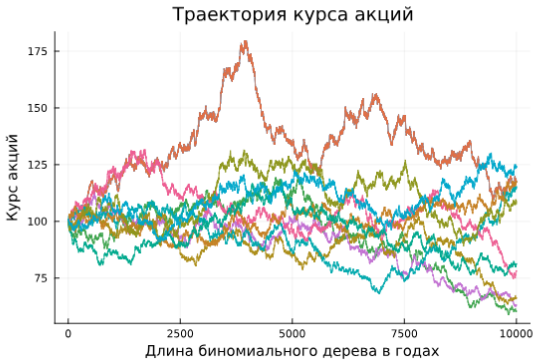


Рис. 2.43: Распараллеливание генерации траектории

3 Листинги программы

```
# 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
# 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:
print(language_created_year(P,"Python"))
# Пример вызова функции и определение даты создания языка Julia:
language_created_year(P,"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
# Пример вызова функции и определение даты создания языка julia:
language_created_year_v2(P,"julia")
# Построчное считывание данных с указанием разделителя:
Tx = readdlm("programminglanguages.csv", ',')
# Запись данных в CSV-файл:
CSV.write("programming_languages_data2.csv", P)
# 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", '-')
# 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]
# 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)
# Подгружаем пакет RDatasets:
using RDatasets
# Задаём структуру данных в виде набора данных:
iris = dataset("datasets", "iris")
# Определения типа переменной:
typeof(iris)
describe(iris)
# Отсутствующий тип:
a = missing
typeof(a)
# Пример операции с переменной отсутствующего типа:
a + 1
# Определение перечня продуктов:
foods = ["apple", "cucumber", "tomato", "banana"]
# Определение калорий:
calories = [missing,47,22,105]
# Определение типа переменной:
typeof(calories)
```

```
# Подключаем пакет Statistics:
using Statistics
# Определение среднего значения:
mean(calories)
# Определение среднего значения без значений с отсутствующим типом:
mean(skipmissing(calories))
# 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)
# Подключаем пакет FileIO:
using FileIO
# Подключаем пакет ImageIO:
import Pkg
Pkg.add("ImageIO")
# Загрузка изображения:
X1 = load("julialogo.png")
# Определение типа и размера данных:
@show typeof(X1);
@show size(X1);
# Загрузка пакетов:
import Pkg
Pkg.add("DataFrames")
Pkg.add("Statistics")
using DataFrames
```

```
using CSV
import Pkg
Pkg.add("Plots")
# Загрузка данных:
houses = CSV.File("houses.csv") |> DataFrame
# Построение графика:
using Plots
plot(size=(500,500),leg=false)
x = houses[!,:sqft]
y = houses[!,:price]
scatter(x,y,markersize=3)
# Фильтрация данных по заданному условию:
filter_houses = houses[houses[!,:sqft].>0,:]
# Построение графика:
x = filter_houses[!,:sqft]
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]))
# Подключение пакета Clustering:
import Pkg
Pkg.add("Clustering")
using Clustering
# Добавление данных :latitude и :longitude в новый фрейм:
X = filter_houses[!, [:latitude,:longitude]]
# Конвертация данных в матричный вид:
```

```
X = Matrix{Float64}(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)
# 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)
# Фильтрация по районам соседних домов:
cities = filter_houses[idxs,:city]
# Фрейм с указанием площади и цены недвижимости:
F = filter_houses[!, [:sqft,:price]]
# Конвертация данных в массив:
F = Matrix{Float64}(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:
y = MultivariateStats.transform(M, F)

Xr = reconstruct(M, y)
# Построение графика с выделением главных компонент:
scatter(F[1,:],F[2,:])
scatter!(Xr[1,:],Xr[2,:])
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)
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
"""
find_best_fit_python = py"find_best_fit_python"
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)
Задания для самостоятельного выполнения
Кластеризация
using RDatasets
iris = dataset("datasets", "iris")
plot(size=(600, 600), leg=false)

x = iris[!, :SepalLength]
y = iris[!, :PetalLength]
scatter(x, y, markersize=3, title="Распределение признаков
        SepalLength и PetalLength",
    xlabel="SepalLength", ylabel="PetalLength", leg=false)
X = iris[!, [:SepalLength, :PetalLength]]
X = Matrix(iris[!, [:SepalLength, :PetalLength]])
# Транспонирование матрицы с данными:
X = X'
k = length(unique(iris[!, :Species]))
C = kmeans(X,k)
iris_new = DataFrame(cluster = C.assignments,
    SepalLength=iris[!, :SepalLength], PetalLength=iris[!, :PetalLength],
                    Spicies = iris[!, :Species])
cluster_figure = plot(legend = false)
for i = 1:k
    iris_new_clustered = iris_new[iris_new[!, :cluster].==i,:]
    xvals = iris_new_clustered[!, :SepalLength]
    yvals = iris_new_clustered[!, :PetalLength]
    scatter!(cluster_figure, xvals, yvals, markersize=4)
```

```
end
xlabel!("SepalLength")
ylabel!("PetalLength")
title!("Iris color-coded by cluster")
display(cluster_figure)
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)
Регрессия (метод наименьших квадратов в случае линейной регрессии)
# Часть 1
X = randn(1000, 3)
a0 = rand(3)
print(a0)
y = X * a0 + 0.1 * randn(1000);
function linear_regression_model(X,y)
    X2 = ones(1000)
    X = hcat(X,X2)
    return X \ y
end
a = linear_regression_model(X,y)
```

```
print(a)
using MultivariateStats
# using llsq
a = llsq(X,y)
print(a)
Pkg.add("GLM")
using GLM, DataFrames
X1 = X[:, 1]
X2 = X[:, 2]
X3 = X[:, 3]

data = DataFrame(y=y, x1=X1, x2=X2, x3=X3);

lm(@formula(y ~ x1 + x2 + x3), data)
# Часть 2
X = rand(100);
y = 2X + 0.1 * randn(100);
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)
Модель ценообразования биномиальных опционов
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
using Plots
plot(stockTree, title="Траектория курса акций",
    xlabel="Длина биномиального дерева в годах",
    ylabel="Курс акций", leg=false)
function createPath(S::Float64, T::Float64, n::Int64, sigma::Float64, r::Float64)
    # S - начальная цена акции;
    # T - длина биномиального дерева в годах;
    # n - длина одного периода;
    # sigma - волатильность акции;
    # ☒ – годовая процентная ставка;

    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)
    stockTree = []
    append!(stockTree, S)
    j = 0

    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
    return stockTree
end
for i in 1:10
    IJulia.clear_output(true)
    traj = createPath(100.0, 1.0, 10000, 0.3, 0.08)
    if i == 1
        p = plot(traj, title="Траектория курса акций",
            xlabel="Длина биномиального дерева в годах",
            ylabel="Курс акций", leg=false)
    end
    p = plot!(traj)
    display(p)
end
```

```
using Base.Threads
Threads.@threads for i in 1:10
    IJulia.clear_output(true)
    traj = createPath(100.0, 1.0, 10000, 0.3, 0.08)
    if i == 1
        g = plot(traj, title="Траектория курса акций",
            xlabel="Длина биномиального дерева в годах",
            ylabel="Курс акций", leg=false)
    end
    g = plot!(traj)
    display(g)
end
```

4 Вывод

Я специализировала пакетов Julia для обработки данных.