Лабораторная работа №7

Компьютерный практикум по статистическому анализу данных

Николаев Дмитрий Иванович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Повторение примеров

Повторим примеры, представленные в лабораторной работе ([1]).

2.1.1 Julia для науки о данных

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

2.1.1.1 Считывание данных

Перед тем, как начать проводить какие-либо операции над данными, необходимо их откуда-то считать и возможно сохранить в определённой структуре.

Довольно часто данные для обработки содержаться в csv-файле, имеющим текстовый формат, в котором данные в строке разделены, например, запятыми, и соответствуют ячейкам таблицы, а строки данных соответствуют строкам таблицы. Также данные могут быть представлены в виде фреймов или множеств.

В Julia для работы с такого рода структурами данных используют пакеты CSV, DataFrames, RDatasets, FileIO:

Рассмотрим возможности пакетов Julia по считыванию данных ([2.1-2.5]).

Лабораторная работа №7. Введение в работу с данными

Повторение примеров

7.2.1. Julia для науки о данных

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

7.2.1.1. Считывание данных

Довольно часто данные для обработки содержаться в csv-файле, имеющим текстовый формат, в котором данные в строке разделены, например, запятыми, и соответствуют ячейкам таблицы, а строки данных соответствуют строкам таблицы. Также данные могут быть представлены в виде фреймов или множеств.

В Julia для работы с такого рода структурами данных используют пакеты CSV, DataFrames, RDatasets, FileIO:

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

```
[1]: up (generic function with 7 methods)
```

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

```
   Updating registry at `C:\Users\User\.julia\registries\General.toml`
  Resolving package versions...
  Installed libpng_jll ─ v1.6.38+1
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
    Updating `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
^ [b53b4c65] ↑ libpng_jll v1.6.38+0 ⇒ v1.6.38+1
        Info Packages marked with ^ have new versions available and may be upgradable.
Precompiling project...
  ✓ libpng_jll
  ✓ libsixel_jll
```

Рис. 2.1: Считывание данных (1)

```
[b53b4c65] ↑ libpng_jll v1.6.38+0 ⇒ v1.6.38+1
        Info Packages marked with ^ have new versions available and may be upgradable.
Precompiling project...
  ✓ libpng_jll
  ✓ libsixel_jll
  ✓ ImageMagick_jll
  ✓ PNGFiles
  ✓ Sixel
  ✓ Cairo_jll
  ✓ HarfBuzz_jll
  ✓ ImageIO
  ✓ ImageMagick
  ✓ libass_jll
  ✓ FFMPEG_jll
  ✓ FFMPEG
  ✓ GR_jll
  ✓ GR
  ✓ Plots
  ✓ Images
  16 dependencies successfully precompiled in 217 seconds. 350 already precompiled. 75 skipped during auto due to previous errors.
   Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
   Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
   Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
```

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

```
[ Info: Precompiling CSV [336ed68f-0bac-5ca0-87d4-7b16caf5d00b]
[ Info: Precompiling DataFrames [a93c6f00-e57d-5684-b7b6-d8193f3e46c0]
```

Рис. 2.2: Считывание данных (2)

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

[4]: 73×2 DataFrame

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. 2.3: Считывание данных (3)

[5]:
```
# 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
```

[5]:
```
language_created_year (generic function with 1 method)
```

[6]:
```
# Пример вызова функции и определение даты создания языка Python:
language_created_year(P,"Python")
```

[6]:
```
1991
```

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

[7]:
```
2012
```

[8]:
```
language_created_year(P,"julia")
```

```
MethodError: no method matching getindex(::DataFrame, ::Nothing, ::Int64)
Closest candidates are:
  getindex(::DataFrame, ::typeof(!), ::Union{Signed, Unsigned}) at C:\Users\User\.julia\packages\DataFrames\S8MUJ\src\dataframe\dataframe.jl:548
  getindex(::DataFrame, ::Colon, ::Union{AbstractString, Signed, Symbol, Unsigned}) at C:\Users\User\.julia\packages\DataFrames\S8MUJ\src\dataframe\dataframe.jl:542
  getindex(::DataFrame, ::InvertedIndex, ::Union{AbstractString, Signed, Symbol, Unsigned}) at C:\Users\User\.julia\packages\DataFrames\S8MUJ\src\dataframe\dataframe.jl:538
  ...

Stacktrace:
 [1] language_created_year(P::DataFrame, language::String)
   @ Main .\In[5]:4
 [2] top-level scope
   @ In[8]:1
```

Рис. 2.4: Считывание данных (4)

```
[9]: # 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

[9]: language_created_year_v2 (generic function with 1 method)

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

[10]: 2012

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

[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. 2.5: Считывание данных (5)

2.1.1.2 Запись данных в файл

Рассмотрим возможности пакетов Julia по записи данных в файл ([2.6,2.7]).

7.2.1.2. Запись данных в файл

Предположим, что требуется записать имеющиеся данные в файл. Для записи данных в формате CSV можно воспользоваться следующим вызовом:

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

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

Можно задать тип файла и разделитель данных:

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

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

Можно проверить, используя readdlm, корректность считывания созданного текстового файла:

Рис. 2.6: Запись данных в файл (1)

Можно проверить, используя readdlm, корректность считывания созданного текстового файла:

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

```
[15]: 74x2 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"
```

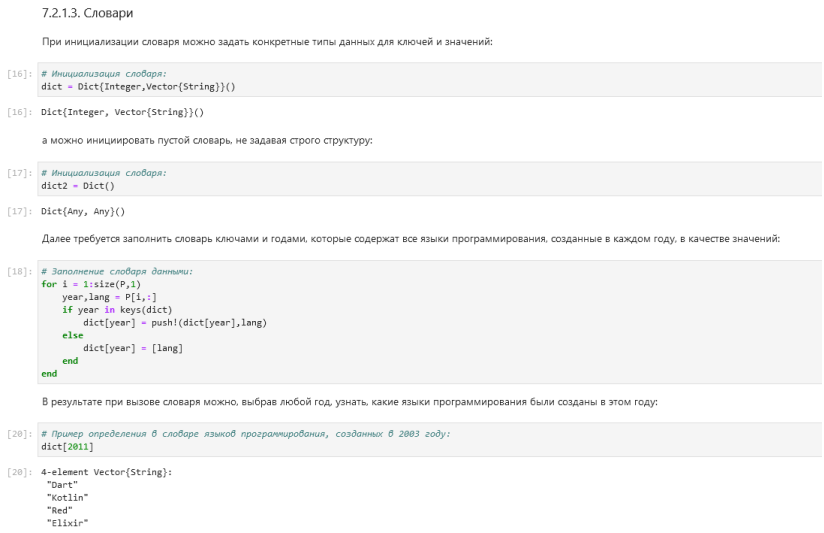
Рис. 2.7: Запись данных в файл (2)

2.1.1.3 Словари

При работе с данными бывает удобно записать их в формате словаря.

Предположим, что словарь должен содержать перечень всех языков программирования и года их создания, при этом при указании года выводить все языки программирования, созданные в этом году.

Создадим словарь с теми же данными ([2.8]).



7.2.1.3. Словари

При инициализации словаря можно задать конкретные типы данных для ключей и значений:

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

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

а можно инициировать пустой словарь, не задавая строго структуру:

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

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

Далее требуется заполнить словарь ключами и годами, которые содержат все языки программирования, созданные в каждом году, в качестве значений:

```
[18]: # 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
```

В результате при вызове словаря можно, выбрав любой год, узнать, какие языки программирования были созданы в этом году:

```
[20]: # Пример определения в словаре языков программирования, созданных в 2003 году:
      dict[2011]
```

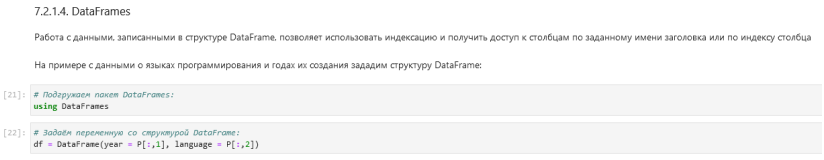
```
[20]: 4-element Vector{String}:
       "Dart"
       "Kotlin"
       "Red"
       "Elixir"
```

Рис. 2.8: Словари

2.1.1.4 DataFrames

Работа с данными, записанными в структуре DataFrame, позволяет использовать индексацию и получить доступ к столбцам по заданному имени заголовка или по индексу столбца.

Рассмотрим работу с DataFrames ([2.9-2.12]).



7.2.1.4. DataFrames

Работа с данными, записанными в структуре DataFrame, позволяет использовать индексацию и получить доступ к столбцам по заданному имени заголовка или по индексу столбца

На примере с данными о языках программирования и годах их создания зададим структуру DataFrame:

```
[21]: # Подружаем пакет DataFrames:
      using DataFrames
```

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

Рис. 2.9: DataFrames (1)

[22]: 73×2 DataFrame

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. 2.10: DataFrames (2)

Если требуется получить доступ к столбцам по имени заголовка, то необходимо добавить к имени заголовка двоеточие:

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

```
[23]: 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
```

Пакет DataFrames предоставляет возможность с помощью description получить основные статистические сведения о каждом столбце во фрейме данных:

Рис. 2.11: DataFrames (3)

Пакет DataFrames предоставляет возможность с помощью description получить основные статистические сведения о каждом столбце во фрейме данных:

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

```
[24]: 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

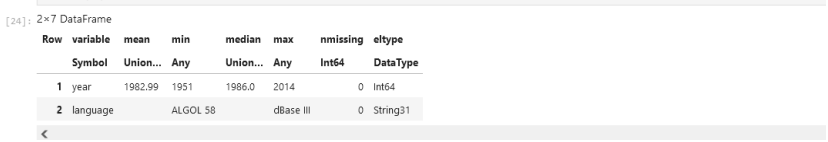


Рис. 2.12: DataFrames (4)

2.1.1.5 RDatasets

Рассмотрим работу с RDatasets ([2.13-2.15]).

7.2.1.5. RDatasets

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

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

```
[ Info: Precompiling RDatasets [ce6b1742-4840-55fa-b093-852dadbb1d8b]
```

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

Рис. 2.13: RDatasets (1)

[26]: 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
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. 2.14: RDatasets (2)

В данном случает набор данных содержит сведения о цветах. При этом следует иметь в виду, что данные, загруженные с помощью набора данных, хранятся в виде DataFrame:

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

```
[27]: DataFrame
```

Пакет RDatasets также предоставляет возможность с помощью description получить основные статистические сведения о каждом столбце в наборе данных:

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

[28]: 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. 2.15: RDatasets (3)

2.1.1.6 Работа с переменными отсутствующего типа (Missing Values)

Рассмотрим особенности работы с переменными отсутствующего типа ([2.16-2.19]).

7.2.1.6. Работа с переменными отсутствующего типа (Missing Values)

Пакет DataFrames позволяет использовать так называемый «отсутствующий» тип:

```
[29]: # Отсутствующий тип:
a = missing
```

```
[29]: missing
```

```
[30]: typeof(a)
```

```
[30]: Missing
```

В операции сложения числа и переменной с отсутствующим типом значение также будет иметь отсутствующий тип:

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

```
[31]: missing
```

Приведём пример работы с данными, среди которых есть данные с отсутствующим типом. Предположим есть перечень продуктов, для которых заданы калории:

```
[32]: # Определение перечня продуктов:
foods = ["apple", "cucumber", "tomato", "banana"]
```

```
[32]: 4-element Vector{String}:
 "apple"
 "cucumber"
 "tomato"
 "banana"
```

Рис. 2.16: Работа с Missing Values (1)

```
[33]: # Определение калорий:
calories = [missing,47,22,105]
```

```
[33]: 4-element Vector{Union{Missing, Int64}}:
      missing
       47
       22
      105
```

В массиве значений калорий есть значение с отсутствующим типом:

```
[34]: # Определение типа переменной:
typeof(calories)
```

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

При попытке получить среднее значение калорий, ничего не получится из-за наличия переменной с отсутствующим типом:

```
[35]: # Подключаем пакет Statistics:
using Statistics
```

```
[36]: # Определение среднего значения:
mean(calories)
```

```
[36]: missing
```

Для решения этой проблемы необходимо игнорировать отсутствующий тип:

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

```
[37]: 58.0
```

Рис. 2.17: Работа с Missing Values (2)

Далее показано, как можно сформировать таблицы данных и объединить их в один фрейм:

```
[38]: # Задание сведений о ценах:
prices = [0.85,1.6,0.8,0.6]
```

```
[38]: 4-element Vector{Float64}:
       0.85
       1.6
       0.8
       0.6
```

```
[39]: # Формирование данных о калориях:
dataframe_calories = DataFrame(item=foods,calories=calories)
```

[39]: 4×2 DataFrame

Row	item	calories
	String	Int64?
1	apple	*missing*
2	cucumber	47
3	tomato	22
4	banana	105

Рис. 2.18: Работа с Missing Values (3)

```
[40]: # Формирование данных о ценах:
dataframe_prices = DataFrame(item=foods,price=prices)
```

[40]: 4×2 DataFrame

Row	item	price
	String	Float64
1	apple	0.85
2	cucumber	1.6
3	tomato	0.8
4	banana	0.6

```
[42]: # Объединение данных о калориях и ценах:
DF = innerjoin(dataframe_calories,dataframe_prices,on=:item)
```

[42]: 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. 2.19: Работа с Missing Values (4)

2.1.1.7 FileIO

Рассмотрим работу с так называемыми “сырыми данными” с помощью пакета FileIO ([2.20,2.21]).

7.2.1.7. FileIO

В Julia можно работать с так называемыми «сырыми» данными, используя пакет FileIO:

[43]:
```
# Подключаем пакет FileIO:
using FileIO
```

Попробуем посмотреть, как Julia работает с изображениями.

Подключим соответствующий пакет:

[44]:
```
# Подключаем пакет ImageIO:
import Pkg
Pkg.add("ImageIO")
```

```
   Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
```

Загрузим изображение (в данном случае логотип Julia):

[48]:
```
# Загрузка изображения:
X1 = load("Julialogo.png")
```

Рис. 2.20: FileIO (1)

```
X1 = load("Julialogo.png")
```

[48]:
```
200×320 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)
```

Julia хранит изображение в виде множества цветов:

[49]:
```
# Определение типа и размера данных:
@show typeof(X1);
@show size(X1);
```

```
typeof(X1) = Matrix{ColorTypes.RGBA{FixedPointNumbers.N0f8}}
size(X1) = (200, 320)
```

Рис. 2.21: FileIO (2)

2.1.2 Обработка данных: стандартные алгоритмы машинного обучения в Julia

2.1.2.1 Кластеризация данных. Метод k-средних

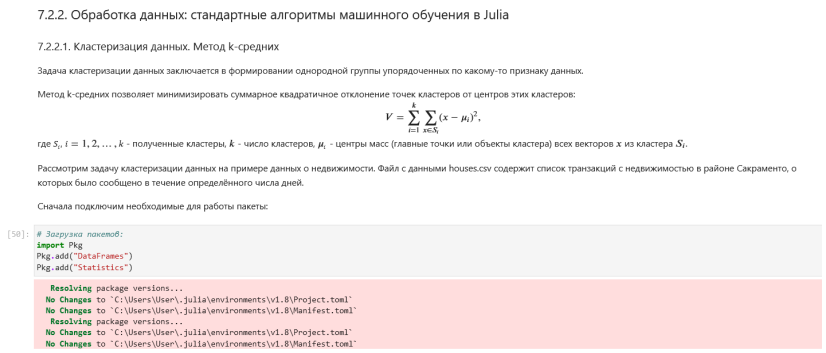
Задача кластеризации данных заключается в формировании однородной группы упорядоченных по какому-то признаку данных.

Метод k-средних позволяет минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров:

$$V = \sum_{i=1}^{k} \sum_{x \in S_i} (x - \mu_i)^2,$$

где $S_i, i = 1, 2, \dots, k$ — полученные кластеры, k — число кластеров, μ_i — центры масс (главные точки или объекты кластера) всех векторов x из кластера S_i.

Реализуем кластеризация данных методом k-средних ([2.22-2.36]).



7.2.2. Обработка данных: стандартные алгоритмы машинного обучения в Julia

7.2.2.1. Кластеризация данных. Метод k-средних

Задача кластеризации данных заключается в формировании однородной группы упорядоченных по какому-то признаку данных.

Метод k-средних позволяет минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров:

$$V = \sum_{i=1}^{k} \sum_{x \in S_i} (x - \mu_i)^2,$$

где $S_i, i = 1, 2, \dots, k$ - полученные кластеры, k - число кластеров, μ_i - центры масс (главные точки или объекты кластера) всех векторов x из кластера S_i.

Рассмотрим задачу кластеризации данных на примере данных о недвижимости. Файл с данными houses.csv содержит список транзакций с недвижимостью в районе Сакраменто, о которых было сообщено в течение определённого числа дней.

Сначала подключим необходимые для работы пакеты:

```
[50]: # Загрузка пакетов:
      import Pkg
      Pkg.add("DataFrames")
      Pkg.add("Statistics")
```

```
  Resolving package versions...
 No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
 No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
  Resolving package versions...
 No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
 No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
```

Рис. 2.22: Кластеризация данных. Метод k-средних (1)

```
[51]: using DataFrames
      using CSV
```

```
[52]: import Pkg
      Pkg.add("Plots")
```

```
  Resolving package versions...
 No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
 No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
```

Затем загрузим данные:

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

Рис. 2.23: Кластеризация данных. Метод k-средних (2)

```
houses = CSV.File("houses.csv") |> DataFrame
```

[53]: 985×12 DataFrame

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
13	2930 LA ROSA RD	SACRAMENTO	95815	CA	1	1	871	Residential	Wed May 21 00:00:00 EDT 2008	106852	38.6187	-121.436
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
974	2181 WINTERHAVEN CIR	CAMERON PARK	95682	CA	3	2	0	Residential	Thu May 15 00:00:00 EDT 2008	224500	38.6976	-120.996
975	7540 HICKORY AVE	ORANGEVALE	95662	CA	3	1	1456	Residential	Thu May 15 00:00:00 EDT 2008	225000	38.7031	-121.235
976	5024 CHAMBERLIN CIR	ELK GROVE	95757	CA	3	2	1450	Residential	Thu May 15 00:00:00 EDT 2008	228000	38.3898	-121.446
977	2400 INVERNESS DR	LINCOLN	95648	CA	3	2	1358	Residential	Thu May 15 00:00:00 EDT 2008	229027	38.8978	-121.325
978	5 BISHOPGATE CT	SACRAMENTO	95823	CA	4	2	1329	Residential	Thu May 15 00:00:00 EDT 2008	229500	38.4679	-121.445
979	5601 REXLEIGH DR	SACRAMENTO	95823	CA	4	2	1715	Residential	Thu May 15 00:00:00 EDT 2008	230000	38.4453	-121.442
980	1909 YARNELL WAY	ELK GROVE	95758	CA	3	2	1262	Residential	Thu May 15 00:00:00 EDT 2008	230000	38.4174	-121.484
981	9169 GARLINGTON CT	SACRAMENTO	95829	CA	4	3	2280	Residential	Thu May 15 00:00:00 EDT 2008	232425	38.4577	-121.36
982	6932 RUSKUT WAY	SACRAMENTO	95823	CA	3	2	1477	Residential	Thu May 15 00:00:00 EDT 2008	234000	38.4999	-121.459
983	7933 DAFFODIL WAY	CITRUS HEIGHTS	95610	CA	3	2	1216	Residential	Thu May 15 00:00:00 EDT 2008	235000	38.7088	-121.257
984	8304 RED FOX WAY	ELK GROVE	95758	CA	4	2	1685	Residential	Thu May 15 00:00:00 EDT 2008	235301	38.417	-121.397
985	3882 YELLOWSTONE LN	EL DORADO HILLS	95762	CA	3	2	1362	Residential	Thu May 15 00:00:00 EDT 2008	235738	38.6552	-121.076

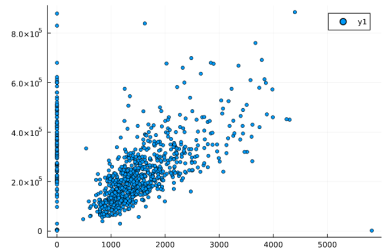
Рис. 2.24: Кластеризация данных. Метод k-средних (3)

Построим график цен на недвижимость в зависимости от площади:

[54]:
```
# Построение графика:
using Plots
```

[55]:
```
plot(size=(500,500),leg=false)
x = houses[!,:sq__ft]
y = houses[!,:price]
scatter(x,y,markersize=3)
```

[55]:



Как видно из графика, имеются так называемые «артефакты», т.е. проявляются отсутствующие или невозможные сведения в исходных данных, например, цены на недвижимость нулевой площади.

Рис. 2.25: Кластеризация данных. Метод k-средних (4)

Для того чтобы избавиться от такого эффекта, можно отфильтровать и исключить такие значения, получить более корректный график цен:

[56]:
```
# Фильтрация данных по заданному условию:
filter_houses = houses[houses[!,:sq__ft].>0,:]
```

Рис. 2.26: Кластеризация данных. Метод k-средних (5)

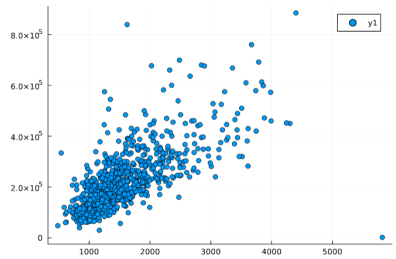
[56]: 814×12 DataFrame

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
13	2930 LA ROSA RD	SACRAMENTO	95815	CA	1	1	871	Residential	Wed May 21 00:00:00 EDT 2008	106852	38.6187	-121.436
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
803	7381 WASHBURN WAY	NORTH HIGHLANDS	95660	CA	3	1	960	Residential	Thu May 15 00:00:00 EDT 2008	224252	38.7035	-121.375
804	7540 HICKORY AVE	ORANGEVALE	95662	CA	3	1	1456	Residential	Thu May 15 00:00:00 EDT 2008	225000	38.7031	-121.235
805	5024 CHAMBERLIN CIR	ELK GROVE	95757	CA	3	2	1450	Residential	Thu May 15 00:00:00 EDT 2008	228000	38.3898	-121.446
806	2400 INVERNESS DR	LINCOLN	95648	CA	3	2	1358	Residential	Thu May 15 00:00:00 EDT 2008	229027	38.8978	-121.325
807	5 BISHOPGATE CT	SACRAMENTO	95823	CA	4	2	1329	Residential	Thu May 15 00:00:00 EDT 2008	229500	38.4679	-121.445
808	5601 REXLEIGH DR	SACRAMENTO	95823	CA	4	2	1715	Residential	Thu May 15 00:00:00 EDT 2008	230000	38.4453	-121.442
809	1909 YARNELL WAY	ELK GROVE	95758	CA	3	2	1262	Residential	Thu May 15 00:00:00 EDT 2008	230000	38.4174	-121.484
810	9169 GARLINGTON CT	SACRAMENTO	95829	CA	4	3	2280	Residential	Thu May 15 00:00:00 EDT 2008	232425	38.4577	-121.36
811	6932 RUSKUT WAY	SACRAMENTO	95823	CA	3	2	1477	Residential	Thu May 15 00:00:00 EDT 2008	234000	38.4999	-121.459
812	7933 DAFFODIL WAY	CITRUS HEIGHTS	95610	CA	3	2	1216	Residential	Thu May 15 00:00:00 EDT 2008	235000	38.7088	-121.257
813	8304 RED FOX WAY	ELK GROVE	95758	CA	4	2	1685	Residential	Thu May 15 00:00:00 EDT 2008	235301	38.417	-121.397
814	3882 YELLOWSTONE LN	EL DORADO HILLS	95762	CA	3	2	1362	Residential	Thu May 15 00:00:00 EDT 2008	235738	38.6552	-121.076

Рис. 2.27: Кластеризация данных. Метод k-средних (6)

```
[57]: # Построение графика:
x = filter_houses[!,:sq__ft]
y = filter_houses[!,:price]
scatter(x,y)
```

[57]:



Используя для фильтрации значений функцию by пакета DataFrames и для вычисления среднего значения функцию mean пакета Statistics, можно посмотреть среднюю цену домов определённого типа:

Рис. 2.28: Кластеризация данных. Метод k-средних (7)

Используя для фильтрации значений функцию by пакета DataFrames и для вычисления среднего значения функцию mean пакета Statistics, можно посмотреть среднюю цену домов определённого типа:

```
[58]: # Подключение пакета Statistics:
using Statistics
```

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

[64]: 3×2 DataFrame

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

Отфильтровав таким образом данные, можно приступить к формированию кластеров.

Сначала подключаем необходимые пакеты и формируем данные в нужном виде:

```
[65]: # Подключение пакета CLustering:
import Pkg
Pkg.add("Clustering")
```

```
  Resolving package versions...
 No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
 No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
```

```
[66]: using Clustering
```

```
[ Info: Precompiling Clustering [aaaa29a8-35af-508c-8bc3-b662a17a0fe5]
```

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

Рис. 2.29: Кластеризация данных. Метод k-средних (8)

[80]: 814×2 DataFrame

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
11	38.6377	-121.452
12	38.4707	-121.459
13	38.6187	-121.436
⋮	⋮	⋮
803	38.7035	-121.375
804	38.7031	-121.235
805	38.3898	-121.446
806	38.8978	-121.325
807	38.4679	-121.445
808	38.4453	-121.442
809	38.4174	-121.484
810	38.4577	-121.36
811	38.4999	-121.459
812	38.7088	-121.257
813	38.417	-121.397
814	38.6552	-121.076

Рис. 2.30: Кластеризация данных. Метод k-средних (9)

[77]:
```
# Конвертация данных в матричный вид:
X = convert(Matrix{Float64}, X)
```

```
MethodError: Cannot `convert` an object of type DataFrame to an object of type Matrix{Float64}
Closest candidates are:
  convert(::Type{T}, ::LinearAlgebra.Factorization) where T<:AbstractArray at C:\Users\User\AppData\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\LinearAlgebra\src\factorization.jl:58
  convert(::Type{Array{T, N}}, ::StaticArraysCore.SizedArray{S, T, N, M, Array{T, N}}) where {S, T, N} at C:\Users\User\.julia\packages\StaticArrays\yXGNL\src\SizedArray.jl:88
  convert(::Type{Array{T, N}}, ::StaticArraysCore.SizedArray{S, T, N, M, TData} where {M, TData<:AbstractArray{T, M}}) where {T, S, N} at C:\Users\User\.julia\packages\StaticArrays\yXGNL\src\SizedArray.jl:82
  ...

Stacktrace:
 [1] top-level scope
   @ In[77]:2
```

[87]:
```
X = hcat(X[!, :latitude], X[!, :longitude])
```

[87]:
```
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  -121.401
 38.6212  -121.271
 38.7009  -121.443
 38.6377  -121.452
 38.4787  -121.459
 38.6187  -121.436
  ⋮
 38.7035  -121.375
 38.7031  -121.235
 38.3890  -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
```

Рис. 2.31: Кластеризация данных. Метод k-средних (10)

Каждая функция хранится в виде строки X, но можно транспонировать получившуюся матрицу, чтобы иметь возможность работать с столбцами данных X:

[88]:
```
# Транспонирование матрицы с данными:
X = X'
```

[88]:
```
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
```

В качестве критерия для формирования кластеров данных и определения количества кластеров попробуем использовать количество почтовых индексов:

[89]:
```
# Задание количества кластеров:
k = length(unique(filter_houses[!,:zip]))
```

[89]:
```
66
```

Для определения k-среднего можно воспользоваться соответствующей функцией пакета Statistics:

[90]:
```
# Определение k-среднего:
C = kmeans(X,k)
```

[90]:
```
KmeansResult{Matrix{Float64}, Float64, Int64}([38.58511508333333 38.6931872307692 … 38.592264875 38.621407818181815; -121.40477416666668 -121.45854476923073 … -121.38094325000001 -121.44570477272728], [66, 52, 66, 66, 37, 27, 39, 10, 23, 2  …  4, 10, 52, 15, 60, 33, 10, 8, 16, 59], [0.0002275562001159414 6, 0.0002292002310426966 7, 1.3108590792398 9e-5, 6.392816430880907e-5, 0.0001518218232376966 6, 0.0001730482617 8587973, 0.0002487462297722316, 0.0006515444306889549, 1.1448978326455 31e-5, 0.0001168665839941240 8  …  0.0001813077271636575 5, 5.241325197857 0046e-5, 1.372360203674175e-5, 0.0002582297194410581, 0.0001457850 3396470482, 1.5192672435659915e-6, 0.0004283446687622927, 0.0005581095097490 84, 0.0001547381398268044, 0.0014500179277092684], [12, 13, 10, 33, 2, 21, 1, 15, 25, 17  …  11, 25, 12, 8, 14, 10, 4, 10, 8, 22], 0.2027918416169996, 13, true)
```

Далее сформируем новый фрейм, включающий исходные данные о недвижимости и столбец с данными о назначенном каждому дому кластере:

[91]:
```
# Формирование фрейма данных:
df = DataFrame(cluster = C.assignments, city = filter_houses[!,:city],
    latitude = filter_houses[!,:latitude],
    longitude = filter_houses[!,:longitude],
    zip = filter_houses[!,:zip])
```

Рис. 2.32: Кластеризация данных. Метод k-средних (11)

[91]: 814×5 DataFrame

Row	cluster	city	latitude	longitude	zip
	Int64	String15	Float64	Float64	Int64
1	66	SACRAMENTO	38.6319	-121.435	95838
2	52	SACRAMENTO	38.4789	-121.431	95823
3	66	SACRAMENTO	38.6183	-121.444	95815
4	66	SACRAMENTO	38.6168	-121.439	95815
5	37	SACRAMENTO	38.5195	-121.436	95824
6	27	SACRAMENTO	38.6626	-121.328	95841
7	39	SACRAMENTO	38.6817	-121.352	95842
8	10	SACRAMENTO	38.5351	-121.481	95820
9	23	RANCHO CORDOVA	38.6212	-121.271	95670
10	2	RIO LINDA	38.7009	-121.443	95673
11	32	SACRAMENTO	38.6377	-121.452	95838
12	52	SACRAMENTO	38.4707	-121.459	95823
13	66	SACRAMENTO	38.6187	-121.436	95815
⋮	⋮	⋮	⋮	⋮	⋮
803	6	NORTH HIGHLANDS	38.7035	-121.375	95660
804	8	ORANGEVALE	38.7031	-121.235	95662
805	4	ELK GROVE	38.3898	-121.446	95757
806	18	LINCOLN	38.8978	-121.325	95648
807	52	SACRAMENTO	38.4679	-121.445	95823
808	15	SACRAMENTO	38.4453	-121.442	95823
809	60	ELK GROVE	38.4174	-121.484	95758
810	33	SACRAMENTO	38.4577	-121.36	95829
811	10	SACRAMENTO	38.4999	-121.459	95823
812	8	CITRUS HEIGHTS	38.7088	-121.257	95610
813	16	ELK GROVE	38.417	-121.397	95758
814	59	EL DORADO HILLS	38.6552	-121.076	95762

Рис. 2.33: Кластеризация данных. Метод k-средних (12)

Построим график, обозначив каждый кластер отдельным цветом:

[92]:
```
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)
```

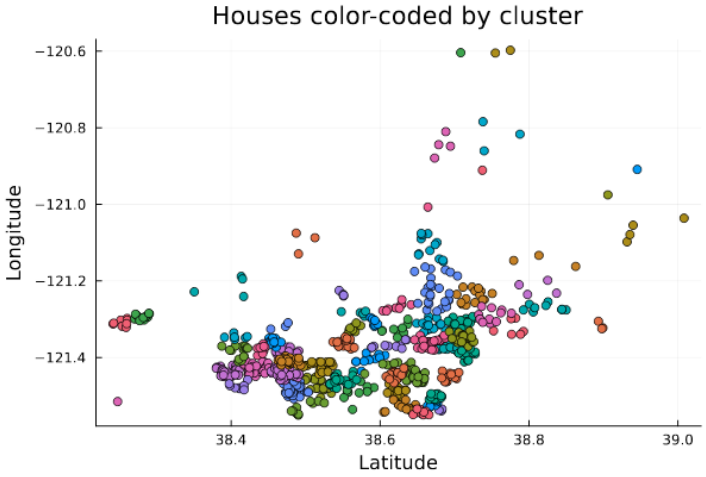


Рис. 2.34: Кластеризация данных. Метод k-средних (13)

Построим график, раскрасив кластеры по почтовому индексу:

```
[93]: unique_zips = unique(filter_houses[!,:zip])
```

```
[93]: 66-element Vector{Int64}:
 95838
 95823
 95815
 95824
 95841
 95842
 95820
 95670
 95673
 95822
 95621
 95833
 95660
    ⋮
 95650
 95821
 95603
 95762
 95677
 95623
 95663
 95746
 95619
 95614
 95690
 95691
```

Рис. 2.35: Кластеризация данных. Метод k-средних (14)

```
[94]: 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)
```

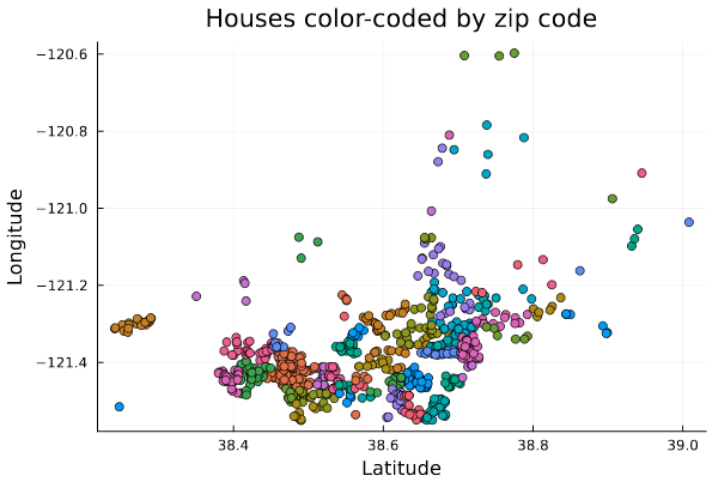


Рис. 2.36: Кластеризация данных. Метод k-средних (15)

2.1.2.2 Кластеризация данных. Метод k ближайших соседей

Данный метод заключается в отнесении объекта к тому из известных классов, который является наиболее распространённым среди k соседей данного элемента. В случае использования метода для регрессии, объекту присваивается среднее значение по k ближайшим к нему объектам.

Реализуем кластеризация данных методом k ближайших соседей ([2.37-2.40]).

7.2.2.2. Кластеризация данных. Метод k ближайших соседей

Данный метод заключается в отнесении объекта к тому из известных классов, который является наиболее распространённым среди k соседей данного элемента. В случае использования метода для регрессии, объекту присваивается среднее значение по k ближайшим к нему объектам.

Рассмотрим использование метода k ближайших соседей на примере того же файла с данными об объектах недвижимости в Сакраменто.

Подключим необходимый пакет:

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

```
  Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
```

```
[96]: using NearestNeighbors
```

Найдём k-среднее одного из объектов недвижимости:

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

```
[118]: 2-element Vector{Float64}:
          38.44004
        -121.421012
```

Определим ближайших соседей:

```
[119]: # Поиск ближайших соседей:
       kdtree = KDTree(X)
```

```
[119]: KDTree{StaticArraysCore.SVector{2, Float64}, Euclidean, Float64}
         Number of points: 814
         Dimensions: 2
         Metric: Euclidean(0.0)
         Reordered: true
```

Рис. 2.37: Кластеризация данных. Метод k ближайших соседей (1)

```
[120]: idxs, dists = knn(kdtree, point, knearest, true)
```

```
[120]: ([70, 764, 196, 125, 557, 368, 415, 92, 112, 683], [0.0, 0.0062648915393364138, 0.008253202590504462, 0.008473585132630057, 0.009164073548370188, 0.009405065124697706, 0.009921759722950759, 0.009941028618812013, 0.010332637707777167, 0.011168993911721985])
```

Отобразим на графике соседей выбранного объекта недвижимости:

```
[121]: # Все объекты недвижимости:
       x = filter_houses[:,:latitude];
       y = filter_houses[:,:longitude];
       scatter(x,y)
```

```
[121]:
```

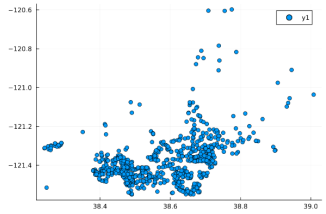


Рис. 2.38: Кластеризация данных. Метод k ближайших соседей (2)

```
[122]: # Соседи:
x = filter_houses[idxs,:latitude];
y = filter_houses[idxs,:longitude];
scatter!(x,y)
```

[122]:

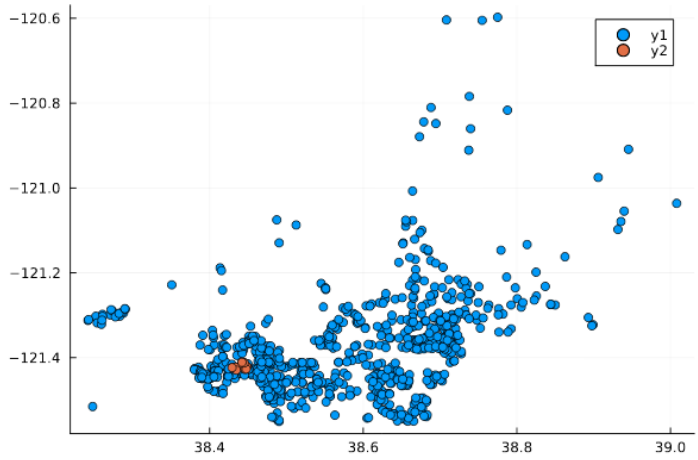


Рис. 2.39: Кластеризация данных. Метод k ближайших соседей (3)

Используя индексы idxs и функцию `:city` для индексации в DataFrame filter_houses, можно определить районы соседних домов:

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

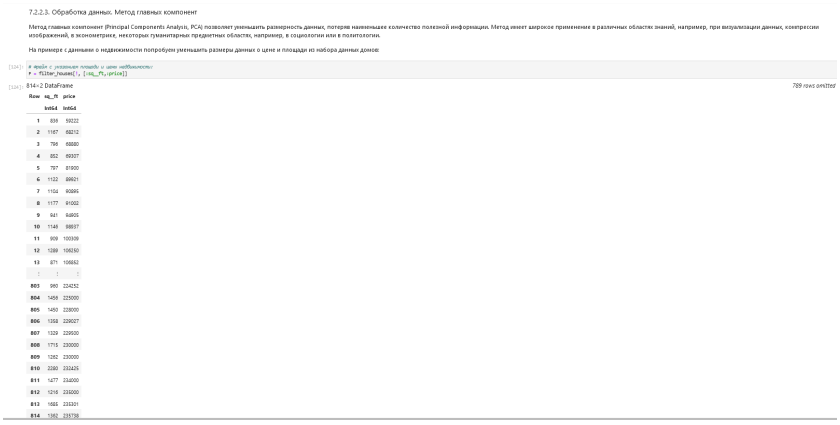
```
[123]: 10-element PooledArrays.PooledVector{String15, UInt32, Vector{UInt32}}:
 "SACRAMENTO"
 "ELK GROVE"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "SACRAMENTO"
 "ELK GROVE"
 "ELK GROVE"
 "ELK GROVE"
 "ELK GROVE"
```

Рис. 2.40: Кластеризация данных. Метод k ближайших соседей (4)

2.1.2.3 Обработка данных. Метод главных компонент

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

Реализуем метод главных компонент ([2.41-2.44]).



7.2.2.3. Обработка данных. Метод главных компонент

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

На примере с данными о недвижимости попробуем уменьшить размеры данных о цене и площади из набора данных donoe:

```
# Фичи с указанием площадь и цена недвижимости:
F = filter_houses[:, [:sq__ft, :price]]
```

814×2 DataFrame — 789 rows omitted

Row	sq__ft	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
811	1477	234000
812	1216	235000
813	1685	235301
814	1362	235738

Рис. 2.41: Обработка данных. Метод главных компонент (1)

```
[125]: # Конвертация данных в массив:
       F = hcat(F[!, :sq__ft], F[!, :price])
```

```
[125]: 814×2 Matrix{Int64}:
         836   59222
        1167   68212
         796   68880
         852   69307
         797   81900
        1122   89921
        1104   90895
        1177   91002
         941   94905
        1146   98937
         909  100309
        1289  106250
         871  106852
           ⋮
         960  224252
        1456  225000
        1450  228000
        1358  229027
        1329  229500
        1715  230000
        1262  230000
        2280  232425
        1477  234000
        1216  235000
        1685  235301
        1362  235738
```

```
[126]: F = F'
```

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

Рис. 2.42: Обработка данных. Метод главных компонент (2)

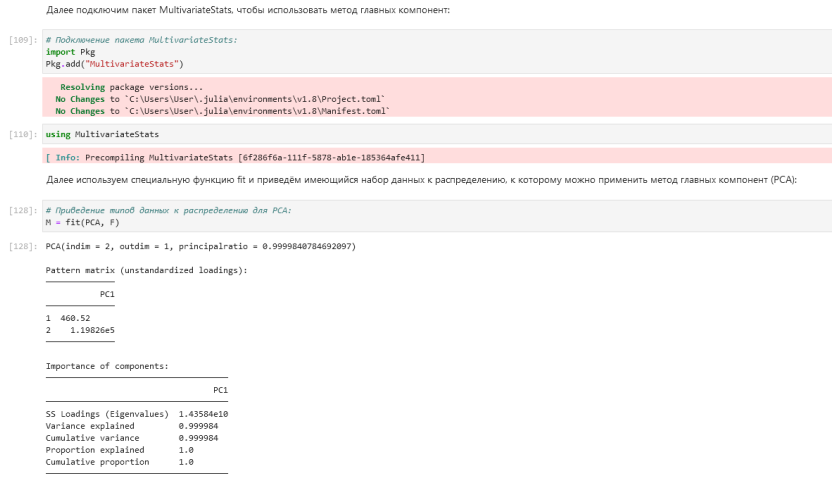


Рис. 2.43: Обработка данных. Метод главных компонент (3)

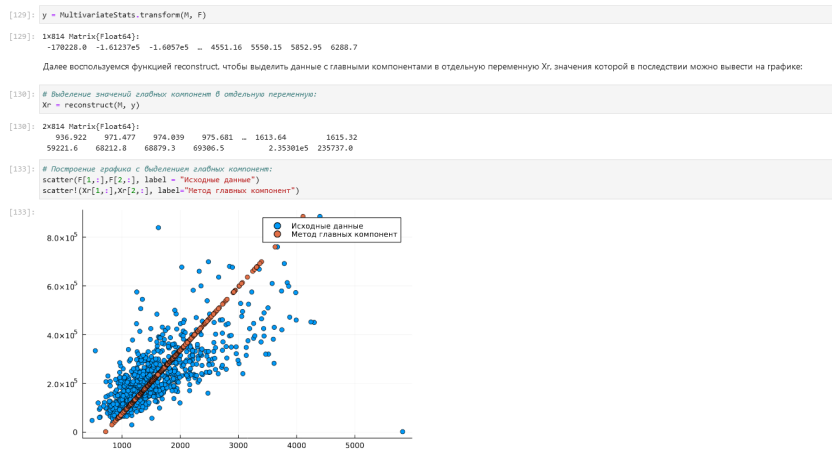


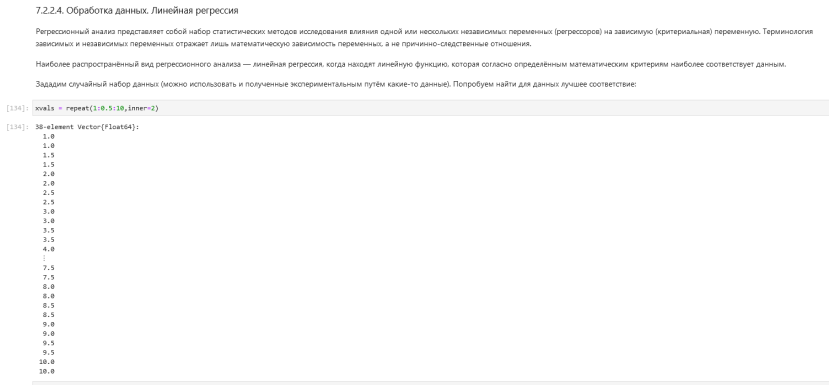
Рис. 2.44: Обработка данных. Метод главных компонент (4)

2.1.2.4 Обработка данных. Линейная регрессия

Регрессионный анализ представляет собой набор статистических методов исследования влияния одной или нескольких независимых переменных (регрессоров) на зависимую (критериальная) переменную. Терминология зависимых и независимых переменных отражает лишь математическую зависимость переменных, а не причинно-следственные отношения.

Наиболее распространённый вид регрессионного анализа — линейная регрессия, когда находят линейную функцию, которая согласно определённым математическим критериям наиболее соответствует данным.

Реализуем линейную регрессию ([2.45-2.51]).


7.2.2.4. Обработка данных. Линейная регрессия

Регрессионный анализ представляет собой набор статистических методов исследования влияния одной или нескольких независимых переменных (регрессоров) на зависимую (критериальная) переменную. Терминология зависимых и независимых переменных отражает лишь математическую зависимость переменных, а не причинно-следственные отношения.

Наиболее распространённый вид регрессионного анализа — линейная регрессия, когда находят линейную функцию, которая согласно определённым математическим критериям наиболее соответствует данным.

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

[134]: xvals = repeat(1:0.5:10,inner=2)

[134]: 38-element Vector{Float64}:
1.0
1.0
1.5
1.5
2.0
2.0
2.5
2.5
3.0
3.0
3.5
3.5
4.0
⋮
7.5
7.5
8.0
8.0
8.5
8.5
9.0
9.0
9.5
9.5
10.0
10.0

Рис. 2.45: Обработка данных. Линейная регрессия (1)

```
[135]: yvals = 3 .+ xvals + 2*rand(length(xvals)) .- 1
```

```
[135]: 38-element Vector{Float64}:
         3.700872114852647
         3.149794490768901
         3.6616850125147256
         4.773823884753347
         4.77283451395138
         4.862024150019121
         5.477049117487369
         5.971653162186213
         6.138003404988811
         6.532678002895267
         5.625310110589791
         5.685028418445752
         6.564074011243896
         ⋮
        10.379815889541872
        10.174086159559883
        11.37882681999818
        10.740023220718566
        11.509567817826987
        11.447265978387565
        12.031868383308332
        11.842423579535676
        11.608377578499919
        11.980200116572918
        13.269764610996738
        12.711339885201506
```

Рис. 2.46: Обработка данных. Линейная регрессия (2)

```
[136]: scatter(xvals,yvals,color=:black,leg=false)
```

[136]:

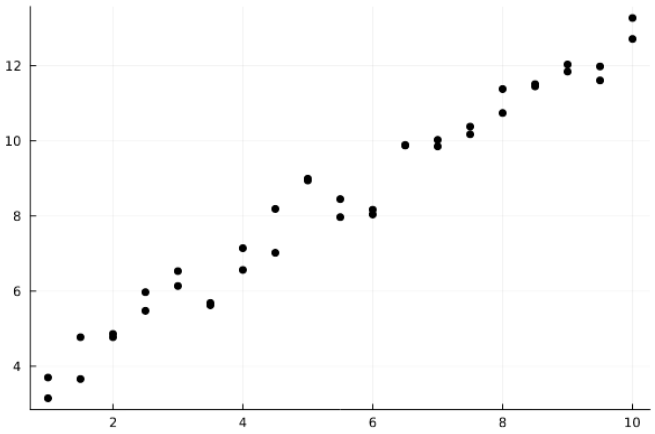


Рис. 2.47: Обработка данных. Линейная регрессия (3)

Определим функцию линейной регрессии:

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

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

Рис. 2.48: Обработка данных. Линейная регрессия (4)

Применим функцию линейной регрессии для построения соответствующего графика значений:

[138]:
```
a,b = find_best_fit(xvals,yvals)
ynew = a * xvals .+ b
plot!(xvals,ynew)
```

[138]:

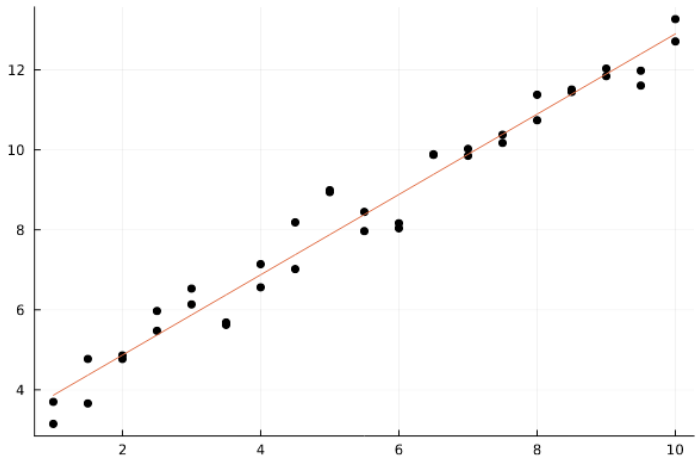


Рис. 2.49: Обработка данных. Линейная регрессия (5)

Сгенерируем больший набор данных:

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

```
size(xvals) = (300000,)
size(yvals) = (300000,)
[139]: (300000,)
```

Определим, сколько времени потребуется, чтобы найти соответствие этим данным:

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

```
  0.002049 seconds (5 allocations: 128 bytes)
[143]: (0.9999999862636407, 3.000436615635408)
```

Для сравнения реализуем подобный код на языке Python:

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

```
   Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
   Resolving package versions...
  No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
  No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
```

```
[145]: using PyCall
       using Conda
```

Рис. 2.50: Обработка данных. Линейная регрессия (6)

```
[163]: 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
       """

[164]: find_best_fit_python = py"find_best_fit_python"

[164]: PyObject <function find_best_fit_python at 0x000001B62C05A3B0>

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

         0.009822 seconds (19 allocations: 448 bytes)
[167]: (0.9999999862636422, 3.0004366155699245)
```

Используем пакет для анализа производительности, чтобы провести сравнение:

```
[168]: import Pkg
       Pkg.add("BenchmarkTools")

          Resolving package versions...
        No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
        No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`

[170]: using BenchmarkTools

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

         7.234 ms (27 allocations: 864 bytes)
[171]: (0.9999999862636422, 3.0004366155699245)

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

         1.007 ms (1 allocation: 32 bytes)
[172]: (0.9999999862636407, 3.000436615635408)
```

Рис. 2.51: Обработка данных. Линейная регрессия (7)

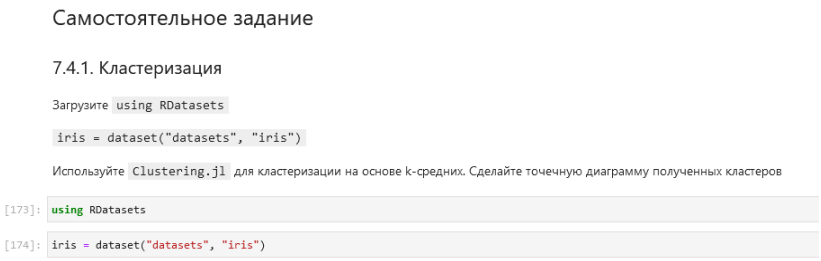
2.2 Самостоятельная работа

2.2.1 Кластеризация

Используя `Clustering.jl` для кластеризации на основе k-средних, сделаем точечную диаграмму полученных кластеров.

Для таблицы данных об ирисах реализуем кластеризацию по количеству ви-

дов ирисов в зависимости от длин стебля и лепестков, а также ширины стебля и лепестков ([2.52-2.64]). Итоговая кластеризация, где справа изображены кластеры по исходным видам ирисов (из данных таблицы), а слева результат кластеризации на основе k-средних ([2.65])


Самостоятельное задание

7.4.1. Кластеризация

Загрузите `using RDatasets`

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

Используйте `Clustering.jl` для кластеризации на основе k-средних. Сделайте точечную диаграмму полученных кластеров

[173]:
```
using RDatasets
```

[174]:
```
iris = dataset("datasets", "iris")
```

Рис. 2.52: Задание 7.4.1. Кластеризация (1)

[174]: 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
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. 2.53: Задание 7.4.1. Кластеризация (2)

```
[183]: F1 = iris[!, [:SepalLength, :PetalLength]]
```

[183]: 150×2 DataFrame

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
146	6.7	5.2
147	6.3	5.0
148	6.5	5.2
149	6.2	5.4
150	5.9	5.1

Рис. 2.54: Задание 7.4.1. Кластеризация (3)

```
[184]: F1 = hcat(F1[!, :SepalLength], F1[!, :PetalLength])'
```

```
[184]: 2×150 adjoint(::Matrix{Float64}) with eltype Float64:
 5.1  4.9  4.7  4.6  5.0  5.4  4.6  5.0  …  6.8  6.7  6.7  6.3  6.5  6.2  5.9
 1.4  1.4  1.3  1.5  1.4  1.7  1.4  1.5     5.9  5.7  5.2  5.0  5.2  5.4  5.1
```

```
[200]: scatter(F1[1,:],F1[2,:], label = "Длина лепестков от длины стебля")
```

[200]:

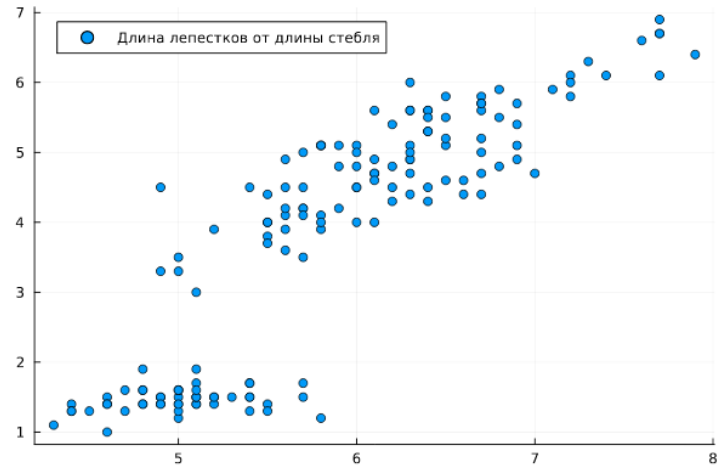


Рис. 2.55: Задание 7.4.1. Кластеризация (4)

```
[197]: F2 = iris[!, [:SepalWidth, :PetalWidth]]
```

[197]: 150×2 DataFrame

Row	SepalWidth	PetalWidth
	Float64	Float64
1	3.5	0.2
2	3.0	0.2
3	3.2	0.2
4	3.1	0.2
5	3.6	0.2
6	3.9	0.4
7	3.4	0.3
8	3.4	0.2
9	2.9	0.2
10	3.1	0.1
11	3.7	0.2
12	3.4	0.2
13	3.0	0.1
⋮	⋮	⋮
139	3.0	1.8
140	3.1	2.1
141	3.1	2.4
142	3.1	2.3
143	2.7	1.9
144	3.2	2.3
145	3.3	2.5
146	3.0	2.3
147	2.5	1.9
148	3.0	2.0
149	3.4	2.3
150	3.0	1.8

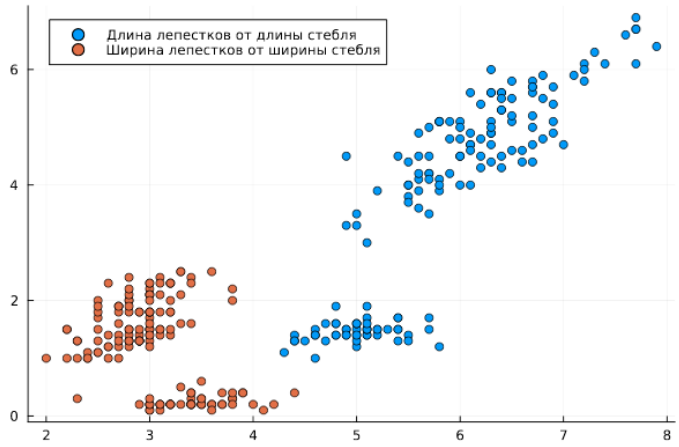
Рис. 2.56: Задание 7.4.1. Кластеризация (5)

```
[198]: F2 = hcat(F2[!, :SepalWidth], F2[!, :PetalWidth])'

[198]: 2×150 adjoint(::Matrix{Float64}) with eltype Float64:
        3.5  3.0  3.2  3.1  3.6  3.9  3.4  3.4  …  3.2  3.3  3.0  2.5  3.0  3.4  3.0
        0.2  0.2  0.2  0.2  0.2  0.4  0.3  0.2     2.3  2.5  2.3  1.9  2.0  2.3  1.8

[201]: scatter!(F2[1,:],F2[2,:], label = "Ширина лепестков от ширины стебля")

[201]:
```




```
[175]: # Задание количества кластеров:
       k = length(unique(iris[!,:Species]))

[175]: 3
```

Рис. 2.57: Задание 7.4.1. Кластеризация (6)

```
[202]: # Определение k-среднего:
       C1 = kmeans(F1, k)

[202]: KmeansResult{Matrix{Float64}, Float64, Int64}([5.007843137254902 6.8390243902439025 5.874137931034483; 1.4921568627450983 5.6780487804878064 4.393103448275863], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1  …  2, 2, 3, 2, 2, 2, 3, 2, 2, 3], [0.01605577470203583, 0.020123029604000092, 0.13169165705497932, 0.16639753940791735, 0.008554402153016830, 0.13690577470204427, 0.17482891195603564, 0.0001230296039977929, 0.37796616685889717, 0.011691657054981874  …  0.025419393221183, 0.33785841760854396, 0.5051991676575369, 0.05078524687684194, 0.01980963712071570, 0.24785841760854055, 0.5496819262782395, 0.34346817370612825, 0.48566329565733213, 0.5003715814506586], [51, 41, 58], [51, 41, 58], 53.80997864418625, 3, true)

[203]: C2 = kmeans(F2, k)

[203]: KmeansResult{Matrix{Float64}, Float64, Int64}([3.0416666666666666 2.707547169811321 3.4510204081632656; 2.052083333333333 1.3094339622641509 0.2448979591836735], [3, 3, 3, 3, 3, 3, 3, 3, 3, 3  …  1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [0.004414827155351102, 0.20543523531861752, 0.06502707285331149, 0.12523115368590665, 0.024210745522697863, 0.22563931695127357, 0.005639316951270956, 0.004618908788003750 5, 0.30563931695126 81, 0.14421074552270 24  …  0.12444878472222243, 0.004865451388888644, 0.13986545138888573, 0.006532118055554 72, 0.26736545138889056, 0.06319878472222129, 0.31653211805555515, 0.004448784722221433, 0.18986545138 888044, 0.06522211805555628], [48, 53, 49], [48, 53, 49], 20.602395953664466, 3, true)

[205]: # Формирование фрейма данных:
       df1 = DataFrame(cluster = C1.assignments, Species = iris[!, :Species],
           SepalLength = iris[!,:SepalLength],
           SepalWidth = iris[!, :SepalWidth],
           PetalLength = iris[!,:PetalLength],
           PetalWidth = iris[!,:PetalWidth])
```

Рис. 2.58: Задание 7.4.1. Кластеризация (7)

[205]: 150×6 DataFrame

Row	cluster	Species	SepalLength	SepalWidth	PetalLength	PetalWidth
	Int64	Cat...	Float64	Float64	Float64	Float64
1	1	setosa	5.1	3.5	1.4	0.2
2	1	setosa	4.9	3.0	1.4	0.2
3	1	setosa	4.7	3.2	1.3	0.2
4	1	setosa	4.6	3.1	1.5	0.2
5	1	setosa	5.0	3.6	1.4	0.2
6	1	setosa	5.4	3.9	1.7	0.4
7	1	setosa	4.6	3.4	1.4	0.3
8	1	setosa	5.0	3.4	1.5	0.2
9	1	setosa	4.4	2.9	1.4	0.2
10	1	setosa	4.9	3.1	1.5	0.1
11	1	setosa	5.4	3.7	1.5	0.2
12	1	setosa	4.8	3.4	1.6	0.2
13	1	setosa	4.8	3.0	1.4	0.1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
139	3	virginica	6.0	3.0	4.8	1.8
140	2	virginica	6.9	3.1	5.4	2.1
141	2	virginica	6.7	3.1	5.6	2.4
142	2	virginica	6.9	3.1	5.1	2.3
143	3	virginica	5.8	2.7	5.1	1.9
144	2	virginica	6.8	3.2	5.9	2.3
145	2	virginica	6.7	3.3	5.7	2.5
146	2	virginica	6.7	3.0	5.2	2.3
147	3	virginica	6.3	2.5	5.0	1.9
148	2	virginica	6.5	3.0	5.2	2.0
149	2	virginica	6.2	3.4	5.4	2.3
150	3	virginica	5.9	3.0	5.1	1.8

Рис. 2.59: Задание 7.4.1. Кластеризация (8)

```
[213]: clusters_figure1 = plot(legend = false)
for i = 1:k
    clustered_irises = df1[df1[!,:cluster].== i,:]
    xvals = clustered_irises[!,:SepalLength]
    yvals = clustered_irises[!,:PetalLength]
    scatter!(clusters_figure1, xvals, yvals, markersize=4)
end
xlabel!("Длина стебля")
ylabel!("Длина лепестков")
title!("Irises color-coded by clusters (Length)")
display(clusters_figure1)
```

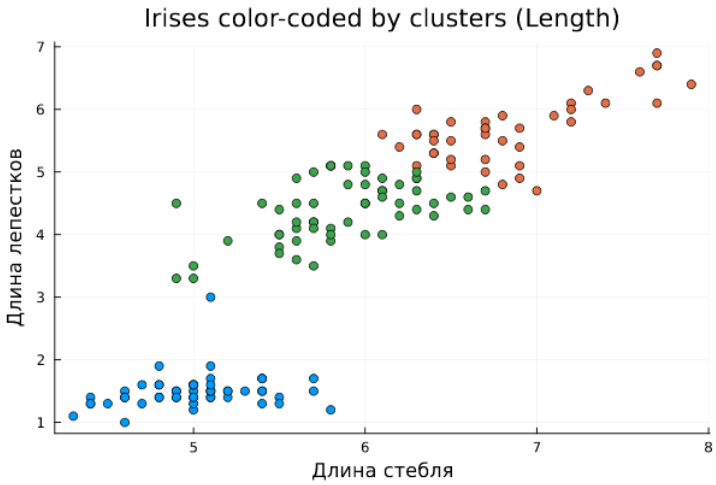


Рис. 2.60: Задание 7.4.1. Кластеризация (9)

```
[214]: unique_species = unique(iris[!, :Species])
```

```
[214]: 3-element Vector{String}:
 "setosa"
 "versicolor"
 "virginica"
```

```
[222]: species_figure1 = plot(legend = true)
for spec in unique_species
    subs = iris[iris[!,:Species].==spec,:]
    x = subs[!,:SepalLength]
    y = subs[!,:PetalLength]
    scatter!(species_figure1, x, y, label = "$(spec)")
end
xlabel!("Длина стебля")
ylabel!("Длина лепестков")
title!("Irises color-coded by species (Length)")
display(species_figure1)
```

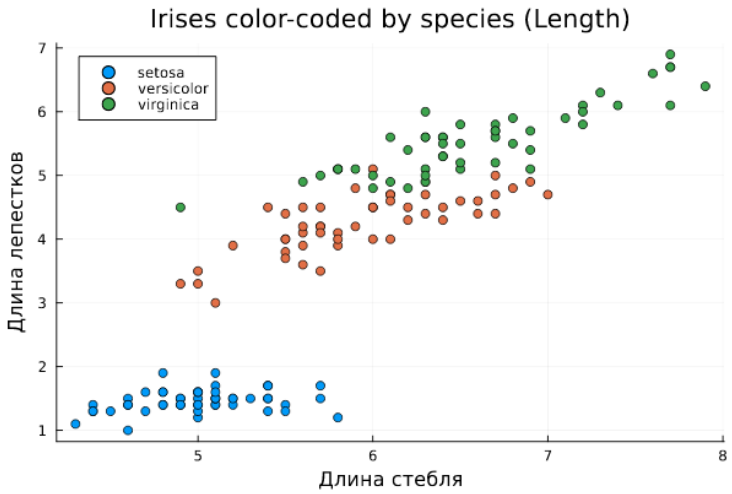


Рис. 2.61: Задание 7.4.1. Кластеризация (10)

[216]:
```
# Формирование фрейма данных:
df2 = DataFrame(cluster = C2.assignments, Species = iris[!, :Species],
    SepalLength = iris[!,:SepalLength],
    SepalWidth = iris[!, :SepalWidth],
    PetalLength = iris[!,:PetalLength],
    PetalWidth = iris[!,:PetalWidth])
```

[216]: 150×6 DataFrame

Row	cluster	Species	SepalLength	SepalWidth	PetalLength	PetalWidth
	Int64	Cat...	Float64	Float64	Float64	Float64
1	3	setosa	5.1	3.5	1.4	0.2
2	3	setosa	4.9	3.0	1.4	0.2
3	3	setosa	4.7	3.2	1.3	0.2
4	3	setosa	4.6	3.1	1.5	0.2
5	3	setosa	5.0	3.6	1.4	0.2
6	3	setosa	5.4	3.9	1.7	0.4
7	3	setosa	4.6	3.4	1.4	0.3
8	3	setosa	5.0	3.4	1.5	0.2
9	3	setosa	4.4	2.9	1.4	0.2
10	3	setosa	4.9	3.1	1.5	0.1
11	3	setosa	5.4	3.7	1.5	0.2
12	3	setosa	4.8	3.4	1.6	0.2
13	3	setosa	4.8	3.0	1.4	0.1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
139	1	virginica	6.0	3.0	4.8	1.8
140	1	virginica	6.9	3.1	5.4	2.1
141	1	virginica	6.7	3.1	5.6	2.4
142	1	virginica	6.9	3.1	5.1	2.3
143	1	virginica	5.8	2.7	5.1	1.9
144	1	virginica	6.8	3.2	5.9	2.3
145	1	virginica	6.7	3.3	5.7	2.5
146	1	virginica	6.7	3.0	5.2	2.3
147	1	virginica	6.3	2.5	5.0	1.9
148	1	virginica	6.5	3.0	5.2	2.0
149	1	virginica	6.2	3.4	5.4	2.3
150	1	virginica	5.9	3.0	5.1	1.8

Рис. 2.62: Задание 7.4.1. Кластеризация (11)

[223]:
```
clusters_figure2 = plot(legend = false)
for i = 1:k
    clustered_irises = df2[df2[!,:cluster].== i,:]
    xvals = clustered_irises[!,:SepalWidth]
    yvals = clustered_irises[!,:PetalWidth]
    scatter!(clusters_figure2, xvals, yvals, markersize=4)
end
xlabel!("Ширина стебля")
ylabel!("Ширина лепестков")
title!("Irises color-coded by clusters (Width)")
display(clusters_figure2)
```

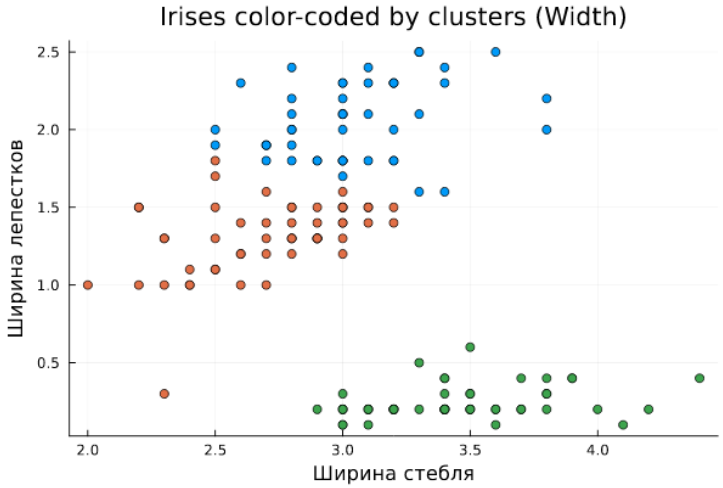


Рис. 2.63: Задание 7.4.1. Кластеризация (12)

[224]:
```
species_figure2 = plot(legend = true)
for spec in unique_species
    subs = iris[iris[!,:Species].==spec,:]
    x = subs[!,:SepalWidth]
    y = subs[!,:PetalWidth]
    scatter!(species_figure2, x, y, label = "$(spec)")
end
xlabel!("Ширина стебля")
ylabel!("Ширина лепестков")
title!("Irises color-coded by species (Width)")
display(species_figure2)
```

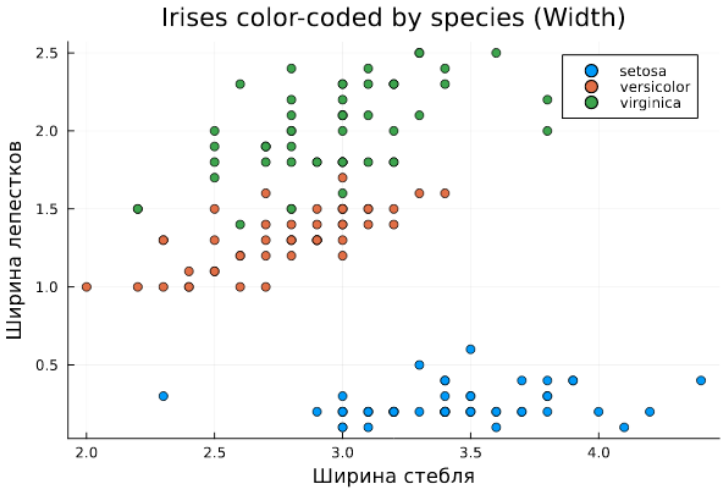


Рис. 2.64: Задание 7.4.1. Кластеризация (13)

```
[226]: plot(
           clusters_figure1,species_figure1,clusters_figure2,species_figure2,
           layout=(2,2),
           legend=true,
           size=(800,600),
       )
```

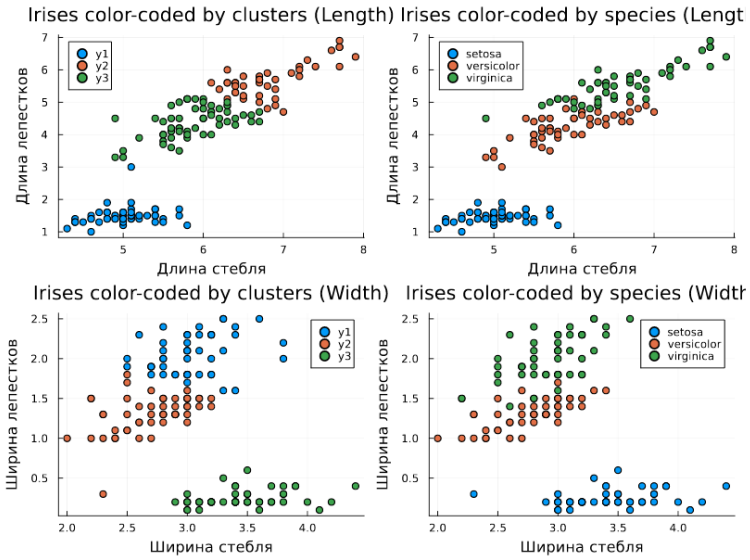
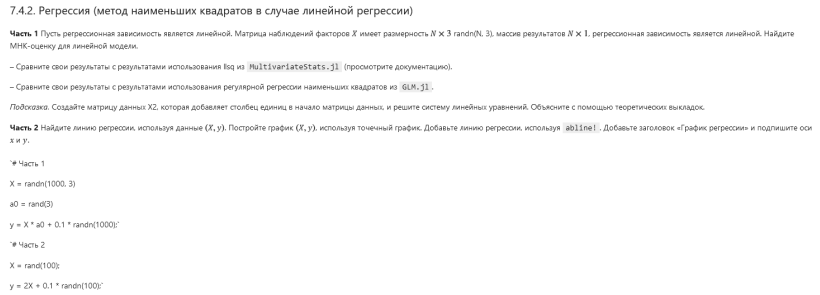


Рис. 2.65: Задание 7.4.1. Кластеризация (14)

2.2.2 Регрессия (метод наименьших квадратов в случае линейной регрессии)

Часть 1:

В условиях задания ([2.66]) выполним линейную регрессию своим методом и методом llsq из пакета `MultivariateStats.jl` ([2.67]), загрузим пакет `GLM.jl` ([2.68]) и решим ту же задачу методом регулярной регрессии из данного пакета ([2.69,2.70]).



7.4.2. 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.

– Сравните свои результаты с результатами использования llsq из `MultivariateStats.jl` (просмотрите документацию).

– Сравните свои результаты с результатами использования регулярной регрессии наименьших квадратов из `GLM.jl`.

Подсказка. Создайте матрицу данных X2, которая добавляет столбец единиц в начало матрицы данных, и решите систему линейных уравнений. Объясните с помощью теоретических выкладок.

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.

```
# Часть 1

X = randn(1000, 3)

a0 = rand(3)

y = X * a0 + 0.1 * randn(1000);

# Часть 2

X = rand(100);

y = 2X + 0.1 * randn(100);
```

Рис. 2.66: Задание 7.4.2. Часть 1. Регрессия (метод наименьших квадратов в случае линейной регрессии) (1)

Часть 1

```
[163]: X = randn(1000, 3)
       a0 = rand(3)
       y = X * a0 + 0.1 * randn(1000);
```

```
[202]: function find_best_fit(xvals,yvals)
           meanx = mean(xvals, dims = 1)
           meany = mean(yvals)
           stdx = std(xvals, dims = 1)
           stdy = std(yvals)
           r = cor(xvals, yvals, dims = 1)
           a = [0., 0., 0.]
           a[1] = r[1]*stdy/stdx[1]
           a[2] = r[2]*stdy/stdx[2]
           a[3] = r[3]*stdy/stdx[3]
           b = meany - (meanx[1]*a[1] + meanx[2]*a[2] + meanx[3]*a[3])
           return a[1], a[2], a[3], b
       end
```

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

```
[203]: a1, a2, a3, b = find_best_fit(X, y)
```

```
[203]: (0.8056844409228956, 0.3946642724238367, 0.6549320859265566, -0.0018294958707968725)
```

```
[204]: a = llsq(X, y, bias = true)
```

```
[204]: 4-element Vector{Float64}:
         0.7896160677142112
         0.4627605056385784
         0.6233463280473506
        -0.0008202220992807055
```

Рис. 2.67: Задание 7.4.2. Часть 1. Регрессия (метод наименьших квадратов в случае линейной регрессии) (2)

```
[167]: import Pkg
       Pkg.add("GLM")
```

```
  Resolving package versions...
  Installed GLM ──────────── v1.9.0
  Installed ShiftedArrays ─ v2.0.0
  Installed StatsModels ──── v0.7.3
   Updating `C:\Users\User\.julia\environments\v1.8\Project.toml`
 [38e38edf] + GLM v1.9.0
   Updating `C:\Users\User\.julia\environments\v1.8\Manifest.toml`
 [38e38edf] + GLM v1.9.0
 [1277b4bf] + ShiftedArrays v2.0.0
 [3eaba693] + StatsModels v0.7.3
Precompiling project...
  ✓ ShiftedArrays
  ✓ StatsModels
  ✓ GLM
  3 dependencies successfully precompiled in 36 seconds. 358 already precompiled. 83 skipped during auto due to previous errors.
```

```
[168]: using GLM
```

Рис. 2.68: Задание 7.4.2. Часть 1. Регрессия (метод наименьших квадратов в случае линейной регрессии) (3)

```
[169]: data = DataFrame(y = y, x1 = X[:, 1], x2 = X[:, 2], x3 = X[:, 3])
```

[169]: 1000×4 DataFrame

Row	y	x1	x2	x3
	Float64	Float64	Float64	Float64
1	0.112826	-0.792696	0.684447	0.628825
2	1.60916	0.144226	0.635432	1.73522
3	-0.104212	1.09997	0.426474	-1.61229
4	1.58855	1.52638	-0.660282	1.16349
5	1.08055	0.916813	0.205755	0.658162
6	-1.43068	-1.6775	0.620572	-0.405418
7	-1.66325	-1.01755	0.236616	-1.34662
8	0.963618	-0.568169	0.682365	1.78743
9	-0.850657	-0.238586	-0.601545	-0.49603
10	-1.51768	-0.399302	-2.13995	-0.175696
11	-1.77173	-1.25413	0.299587	-1.23053
12	1.02045	0.377952	1.50278	0.0781439
13	0.0706521	1.05147	-0.451033	-1.24829
⋮	⋮	⋮	⋮	⋮
989	1.07589	1.89066	-0.612778	-0.447511
990	0.326728	0.558816	-1.62669	1.22148
991	-0.548044	-0.142651	-0.540728	-0.375917
992	-0.580139	-0.0521992	0.148554	-1.01152
993	-0.875436	-2.32577	0.920174	0.569319
994	-0.548573	0.331037	-0.384293	-1.16478
995	-0.279327	-0.521084	-0.757634	0.922247
996	-0.78037	0.264434	-1.61353	-0.348277
997	-0.0601804	0.49836	0.089103	-0.781576
998	-0.735713	1.27568	-1.46213	-1.73693
999	0.293166	1.02036	-0.431648	-0.42969
1000	-0.208143	0.974969	-1.23073	-0.684202

Рис. 2.69: Задание 7.4.2. Часть 1. Регрессия (метод наименьших квадратов в случае линейной регрессии) (4)

```
[171]: lm(@formula(y ~ 1 + x1 + x2 + x3), data)

[171]: 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)	-0.000820222	0.0030891	-0.27	0.7907	-0.00688211	0.00524166
x1	0.789616	0.00317794	248.47	<1e-99	0.78338	0.795852
x2	0.462761	0.00301554	153.46	<1e-99	0.456843	0.468678
x3	0.623346	0.00306642	203.28	<1e-99	0.617329	0.629364

Рис. 2.70: Задание 7.4.2. Часть 1. Регрессия (метод наименьших квадратов в случае линейной регрессии) (5)

Часть 2:

Построим точечный график и осуществим линейную регрессию, построив регресионную прямую с помощью команды `abline` ([2.71,2.72]).

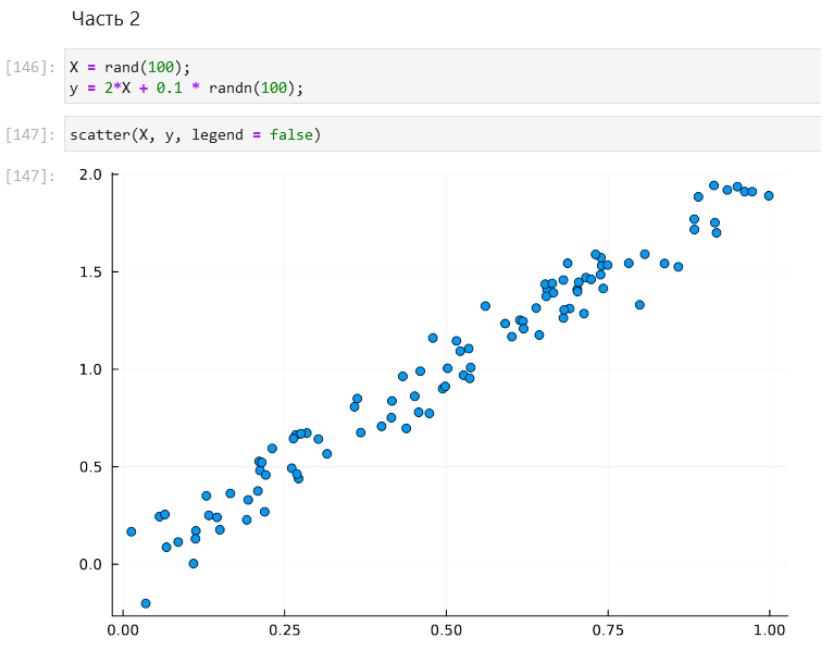


Рис. 2.71: Задание 7.4.2. Часть 2. Регрессия (метод наименьших квадратов в случае линейной регрессии) (1)

```
[148]: a,b = find_best_fit(X, y)
       #ynew = a * X .+ b
       Plots.abline!(a, b)
       xlabel!("x")
       ylabel!("y")
       title!("График регрессии")
```

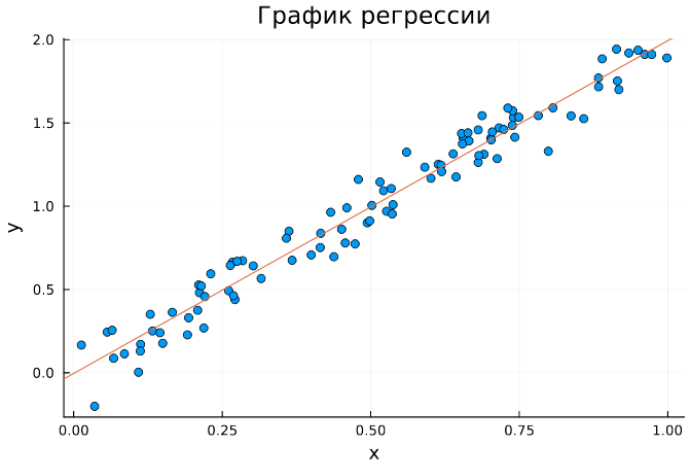


Рис. 2.72: Задание 7.4.2. Часть 2. Регрессия (метод наименьших квадратов в случае линейной регрессии) (2)

2.2.3 Модель ценообразования биномиальных опционов

В условиях задачи ([2.73]), после построения случайной кривой ([2.74]), создадим функцию, которая высчитывает вероятность подъёма цены акции и множители при увеличении и сокращении её цены, после чего, разыгрывая вероятность подъема цены акции, составим список значений цен акции за заданный промежуток ([2.75]). Далее построим траектории цены акции в зависимости от числа периодов ([2.76,2.77]).

7.4.3. Модель ценообразования биномиальных опционов

Постройте траекторию возможных цен на акции:

– S — начальная цена акции;

– T — длина биномиального дерева в годах;

– n — количество периодов;

– $h = \frac{T}{n}$ — длина одного периода;

– σ — волатильность акции;

– r — годовая процентная ставка;

– $u = e^{rh+\sigma\sqrt{h}}$;

– $d = e^{rh-\sigma\sqrt{h}}$;

– $p^* = \frac{e^{rh}-d}{u-d}$.

a) 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. Вы можете использовать функцию построения графика из библиотеки графиков.

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.

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

d) 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. Вы можете использовать функцию построения графика из библиотеки графиков.

Рис. 2.73: Задание 7.4.3. Модель ценообразования биномиальных опционов (1)

Пункт а

[127]:
```
S = 100.;
T = 1;
n = 10000;
σ = 0.3;
r = 0.08;
```

[128]:
```
m = 100
x = range(0, m, length = m+1)
y = rand(m+1)
plot(x, y)
```

[128]:

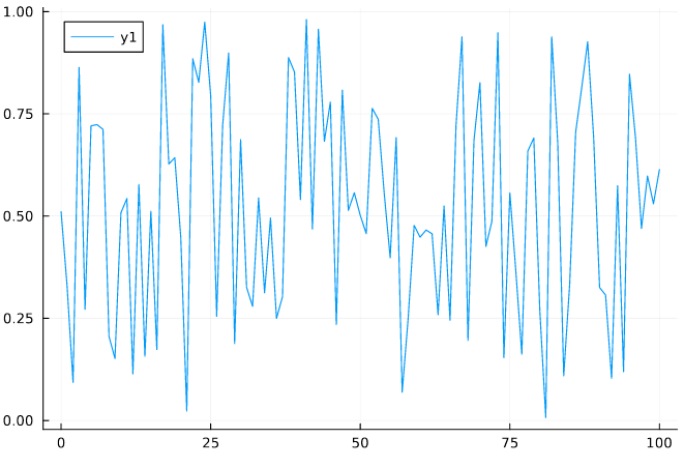


Рис. 2.74: Задание 7.4.3. Модель ценообразования биномиальных опционов (2)

Пункт b

```
[129]: function createPath(S::Float64, r::Float64, sigma::Float64, T::Int64, n::Int64)
           h = T / n
           u = exp(r*h + sigma*√(h))
           d = exp(r*h - sigma*√(h))
           # Вероятность того, что цена акции поднимется
           p = (exp(r*h) - d) / (u - d)
           Price = [S]
           s = S
           for i ∈ 1:n
               q = rand()
               if q < p
                   s = S*u
                   push!(Price, s)
               else
                   s = S*d
                   push!(Price, s)
               end
           end
           return Price
       end

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

[130]: for i ∈ 1:10
           n = 1000*i
           println("Вероятность увеличения цены акции при n = $n равна ",(exp(r*T/n) - exp(r*T/n - σ*√(T/n))) / (exp(r*T/n + σ*√(T/n)) - exp(r*T/n - σ*√(T/n))))
       end

Вероятность увеличения цены акции при n = 1000 равна 0.4976283095425302
Вероятность увеличения цены акции при n = 2000 равна 0.4983229553057931
Вероятность увеличения цены акции при n = 3000 равна 0.4986386970294895
Вероятность увеличения цены акции при n = 4000 равна 0.49881414810090235
Вероятность увеличения цены акции при n = 5000 равна 0.49893934141922
Вероятность увеличения цены акции при n = 6000 равна 0.49903175537374855
Вероятность увеличения цены акции при n = 7000 равна 0.4991035795034541
Вероятность увеличения цены акции при n = 8000 равна 0.4991614752945346
Вероятность увеличения цены акции при n = 9000 равна 0.4992094312437527
Вероятность увеличения цены акции при n = 10000 равна 0.4992500005625153
```

Рис. 2.75: Задание 7.4.3. Модель ценообразования биномиальных опционов (3)

```
[131]: n = 50
       x = range(0, n, length = n+1)
       y = createPath(S, r, σ, T, n)
       println("Вероятность увеличения цены акции при n = $n равна ",(exp(r*T/n) - exp(r*T/n - σ*√(T/n))) / (exp(r*T/n + σ*√(T/n)) - exp(r*T/n - σ*√(T/n))))
       plot(x, y)

Вероятность увеличения цены акции при n = 50 равна 0.48939498898613293
```

[131]:

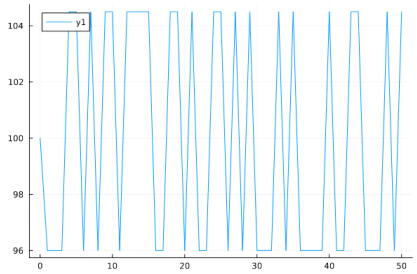


Рис. 2.76: Задание 7.4.3. Модель ценообразования биномиальных опционов (4)

```
[132]: n = 10
       x = range(0, n, length = n+1)
       y = [createPath(S, r, σ, T, n) for i ∈ 1:10]
       plt = plot(x, y)
```

[132]:

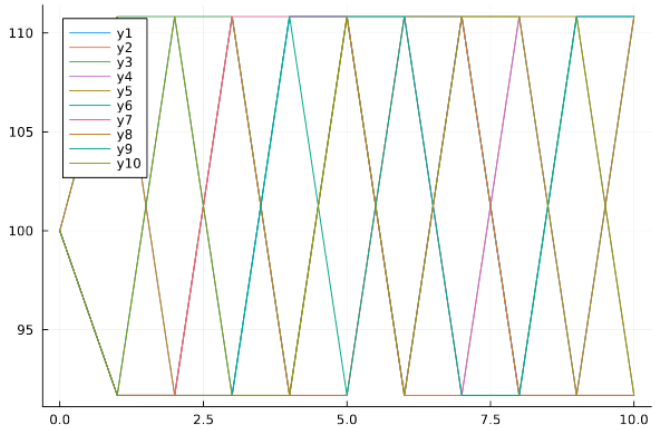


Рис. 2.77: Задание 7.4.3. Модель ценообразования биномиальных опционов (5)

Так как параллелизация невозможна внутри Jupyter ([2.78]), то сделаем данный пункт в отдельном файле ([2.79,2.80]).

Код для пункта c:

```
using Base.Threads
using BenchmarkTools

S = 100.;
T = 1;
n = 10000;
sigma = 0.3;
r = 0.08;

function createPath(S::Float64, r::Float64, sigma::Float64, T::Int64, n::Int64)
    h = T / n
    u = exp(r*h + sigma*sqrt(h))
    d = exp(r*h - sigma*sqrt(h))
```

```
    # Вероятность того, что цена акции поднимется
    p = (exp(r*h) - d) / (u - d)
    Price = [S]
    s = S
    for i in 1:n
        q = rand()
        if q < p
            s = S*u
            push!(Price, s)
        else
            s = S*d
            push!(Price, s)
        end
    end
    return Price
end

println("Число потоков равно ",Threads.nthreads())

x = range(0, n, length = n+1)
y = []
@btime begin
    @sync for i in 1:10
        Threads.@spawn begin
            push!(y, createPath(S, r, sigma, T, n))
        end
    end
end
```

Пункт c

```
[149]: ;julia -t auto

[150]: Threads.nthreads()

[150]: 1

[135]: import Pkg
       Pkg.add("Distributed")

         Resolving package versions...
        No Changes to `C:\Users\User\.julia\environments\v1.8\Project.toml`
        No Changes to `C:\Users\User\.julia\environments\v1.8\Manifest.toml`

[136]: using Distributed

[137]: nprocs()

[137]: 1
```

Дополнительные потоки через notebook файл не вызываются, поэтому распараллеливание сделать не удастся

Сделаем это в отдельном файле ex2_c.jl

Пункт d

Аналогичен пункту a

Рис. 2.78: Задание 7.4.3. Модель ценообразования биномиальных опционов (6)

```
S = 100.;
T = 1;
n = 10000;
g = 0.3;
r = 0.08;

function createPath(S::Float64, r::Float64, sigma::Float64, T::Int64, n::Int64)
    h = T / n
    u = exp(r*h + sigma*√(h))
    d = exp(r*h - sigma*√(h))
    # Вероятность того, что цена акции поднимется
    p = (exp(r*h) - d) / (u - d)
    Price = [S]
    s = S
    for i ∈ 1:n
        q = rand()
        if q < p
            s = S*u
            push!(Price, s)
        else
            s = S*d
            push!(Price, s)
        end
    end
    return Price
end

println("Число потоков равно ",Threads.nthreads())

x = range(0, n, length = n+1)
y = []
@btime begin
    @sync for i ∈ 1:10
        Threads.@spawn begin
            push!(y, createPath(S, r, g, T, n))
        end
    end
end
```

Рис. 2.79: Задание 7.4.3. Модель ценообразования биномиальных опционов (7)

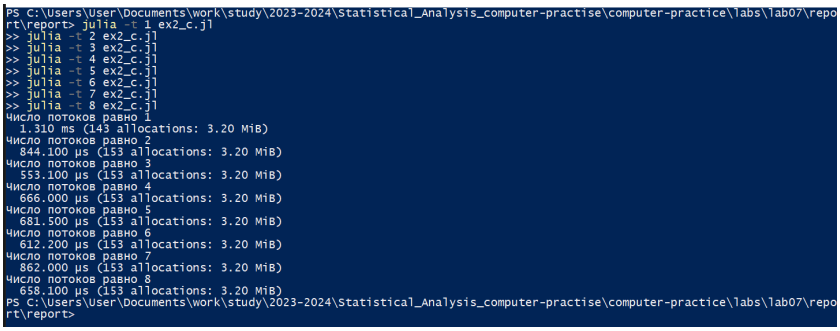


Рис. 2.80: Задание 7.4.3. Модель ценообразования биномиальных опционов (8)

3 Выводы

В ходе выполнения лабораторной работы я освоил специализированные пакеты в Julia для обработки данных.

Список литературы

1. Королькова А. В., Кулябов Д. С. Лабораторная работа № 7. Введение в работу с данными [Электронный ресурс]. RUDN, 2023. URL: https://esystem.rudn.ru/pluginfile.php/2231361/mod_resource/content/2/007-lab_data-science.pdf.