

Лабораторная работа № 7. Введение в работу с данными

7.1. Цель работы

Основной целью работы является специализированных пакетов Julia для обработки данных.

7.2. Предварительные сведения

Обработка и анализ данных, полученных в результате проведения исследований, — важная и неотъемлемая часть исследовательской деятельности. Большое значение имеет выявление определённых связей и закономерностей в имеющихся неструктурированных данных, особенно в данных больших размерностей. Выявленные в данных связи и закономерности позволяет строить прогнозные модели с предполагаемым результатом. Для решения таких задач применяют методы из таких областей знаний как математическая статистика, программирование, искусственный интеллект, машинное обучение.

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

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:

```
# 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
```

Предположим, что у вас в рабочем каталоге с проектом есть файл с данными programminglanguages.csv, содержащий перечень языков программирования и год их создания. Тогда для заполнения массива данными для последующей обработки требуется считать данные из исходного файла и записать их в соответствующую структуру:

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

Далее приведём пример функции, в которой на входе указывается название языка программирования, а на выходе — год его создания:

```
# Функция определения по названию языка программирования года его
# создания:
function language_created_year(P,language::String)
    loc = findfirst(P[:,2].==language)
    return P[loc,1]
end

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

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

В следующем примере при вызове функции, в качестве аргумента которой указано слово `julia`, написанное со строчной буквы:

```
language_created_year(P, "julia")
```

будет выдана ошибка, так как список не содержит таких данных.

Для того, чтобы убрать в функции зависимость данных от регистра, необходимо изменить исходную функцию следующим образом:

```
# 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")
```

Можно считывать данные построчно, с элементами, разделенными заданным разделителем:

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

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:

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

Можно задать тип файла и разделитель данных:

```
# 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, '-')
```

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

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

7.2.1.3. 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.

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

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

а можно инициировать пустой словарь, не задавая строго структуру:

```
# Инициализация словаря:
dict2 = Dict()
```

В последнем случае словарь принимает ключи и значения любого типа.

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

```
# 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:

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

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:

```
# Подгружаем пакет DataFrames:
using DataFrames
```

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

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

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

В Julia это означает, что имена заголовков обрабатываются как символы. Также следует иметь в виду, что вызов `df[1]` эквивалентен вызову `df[:, :year]`.

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

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

7.2.1.5. RDatasets

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

```
# Подгружаем пакет RDatasets:
using RDatasets
```

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

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

```
# Определения типа переменной:
typeof(iris)
```

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

```
describe(iris)
```

7.2.1.6. Работа с переменными отсутствующего типа (Missing Values)

Пакет `DataFrames` позволяет использовать так называемый «отсутствующий» тип:

```
# Отсутствующий тип:
a = missing
typeof(a)
```

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

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

Приведём пример работы с данными, среди которых есть данные с отсутствующим типом.

Предположим есть перечень продуктов, для которых заданы калории:

```
# Определение перечня продуктов:
foods = ["apple", "cucumber", "tomato", "banana"]
# Определение калорий:
calories = [missing, 47, 22, 105]
```

В массиве значений калорий есть значение с отсутствующим типом:

```
# Определение типа переменной:
typeof(calories)
```

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

```
# Подключаем пакет Statistics:
using Statistics

# Определение среднего значения:
mean(calories)
```

Для решения этой проблемы необходимо игнорировать отсутствующий тип:

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

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

```
# 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 = join(dataframe_calories, dataframe_prices, on=:item)
```

7.2.1.7. FileIO

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

```
# Подключаем пакет FileIO:
using FileIO
```

Попробуем посмотреть, как Julia работает с изображениями.

Подключим соответствующий пакет:

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

Загрузим изображение (в данном случае логотип Julia):

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

Julia хранит изображение в виде множества цветов:

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

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 Сакramento, о которых было сообщено в течение определённого числа дней.

Сначала подключим необходимые для работы пакеты:

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

Затем загрузим данные:

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

Построим график цен на недвижимость в зависимости от площади (рис. 7.1):

```
# Построение графика:
using Plots
plot(size=(500,500),leg=false)
```

```
x = houses[!, :sq_ft]
y = houses[!, :price]
scatter(x,y,markersize=3)
```

Как видно из графика на рис. 7.1, имеются так называемые «артефакты», т.е. проявляются отсутствующие или невозможные сведения в исходных данных, например, цены на недвижимость нулевой площади.

Для того чтобы избавиться от такого эффекта, можно отфильтровать и исключить такие значения, получить более корректный график цен (рис. 7.2):

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

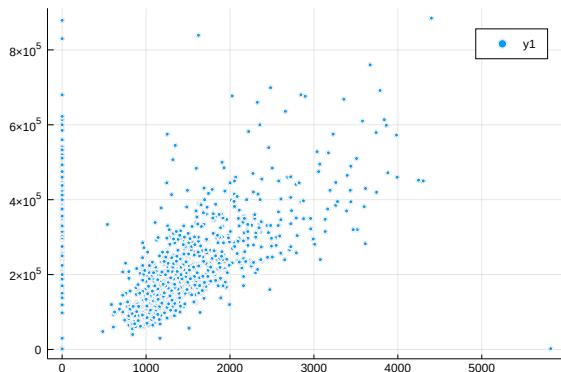


Рис. 7.1. Цены на недвижимость в зависимости от площади

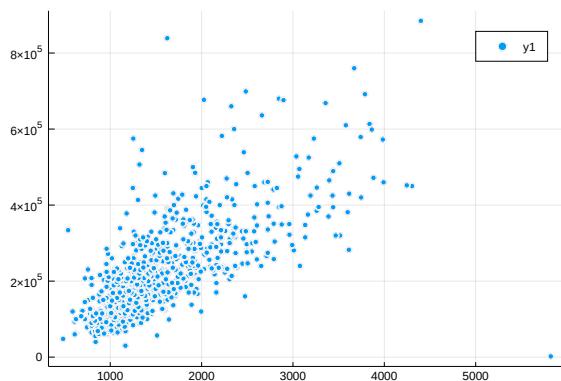


Рис. 7.2. Цены на недвижимость в зависимости от площади (исключены артефакты данных)

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

```
# Подключение пакета Statistics:  
using Statistics
```

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

Отфильтровав таким образом данные, можно приступить к формированию кластеров. Сначала подключаем необходимые пакеты и формируем данные в нужном виде:

```
# 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 = convert(Matrix{Float64}, X)

```

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

```

# Транспонирование матрицы с данными:
X = X'

```

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

```

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

```

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

```

# Определение k-среднего:
C = kmeans(X,k)

```

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

```

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

```

Построим график (рис. 7.3), обозначив каждый кластер отдельным цветом:

```

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)

```

Построим график (рис. 7.4), раскрасив кластеры по почтовому индексу:

```

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)

```

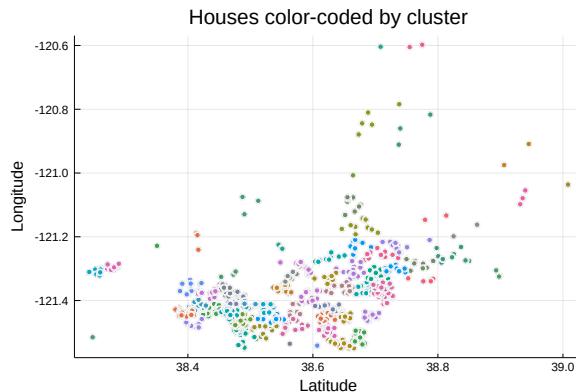


Рис. 7.3. Пример кластеризации объектов недвижимости по географическому расположению

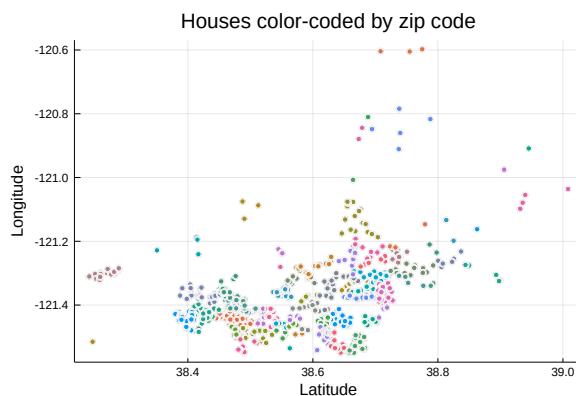


Рис. 7.4. Пример кластеризации объектов недвижимости по почтовому индексу

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:

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

```
using NearestNeighbors
```

Найдём k-среднее одного из объектов недвижимости:

```
knearest = 10
```

```
id = 70
```

```
point = X[:,id]
```

Определим ближайших соседей:

```
# Поиск ближайших соседей:
```

```
kdtree = KDTree(X)
```

```
idxs, dists = knn(kdtree, point, knearest, true)
```

Отобразим на графике соседей выбранного объекта недвижимости (рис. 7.5):

```
# 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)
```

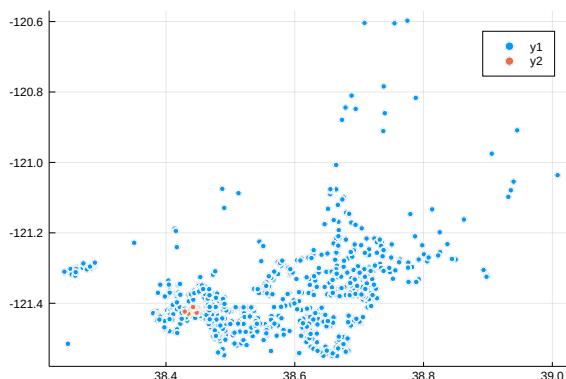


Рис. 7.5. Определение соседей объекта недвижимости

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

```
# Фильтрация по районам соседних домов:
```

```
cities = filter_houses[idxs, :city]
```

7.2.2.3. 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.

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

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

# Конвертация данных в массив:
F = convert(Array{Float64,2},F)'
```

Далее подключим пакет MultivariateStats, чтобы использовать метод главных компонент:

```
# Подключение пакета MultivariateStats:
import Pkg
Pkg.add("MultivariateStats")
using MultivariateStats
```

Далее используем специальную функцию `fit` и приведём имеющийся набор данных к распределению, к которому можно применить метод главных компонент (PCA):

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

Далее воспользуемся функцией `reconstruct`, чтобы выделить данные с главными компонентами в отдельную переменную X_r , значения которой в последствии можно вывести на графике (рис. 7.6):

```
# Выделение значений главных компонент в отдельную переменную:
Xr = reconstruct(M, y)

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

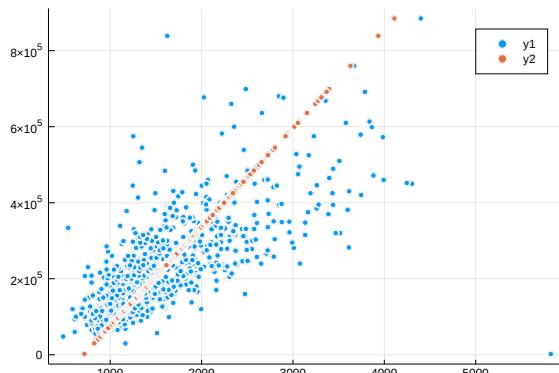


Рис. 7.6. Определение главных компонент для данных по объектам недвижимости

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 (рис. 7.7):

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

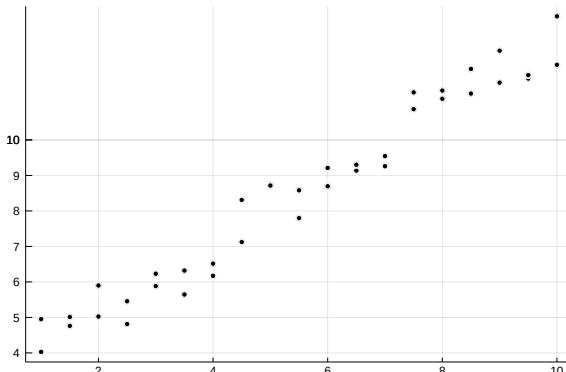


Рис. 7.7. Исходные данные

Определим функцию линейной регрессии:

```
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
```

Применим функцию линейной регрессии для построения соответствующего графика значений (рис. 7.8):

```
a,b = find_best_fit(xvals,yvals)
ynew = a * xvals .+ b
```

```
plot!(xvals,ynew)
```

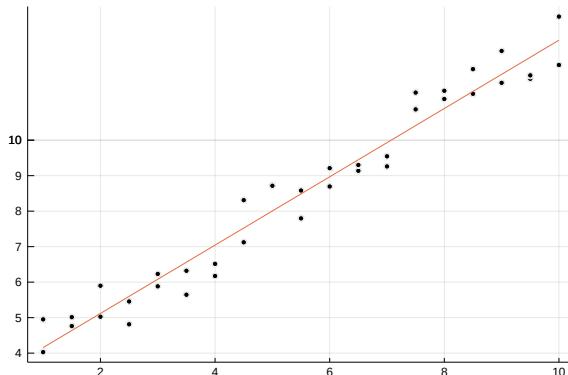


Рис. 7.8. Линейная регрессия

Сгенерируем больший набор данных:

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

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

```
@time a,b = find_best_fit(xvals,yvals)
```

Для сравнения реализуем подобный код на языке Python:

```
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.correlcoef(xvals,yvals)[0][1]
    a = r*stdy/stdx
    b = meany - a*meanx
    return a,b
"""

xpy = PyObject(xvals)
ypy = PyObject(yvals)
@time a,b = find_best_fit_python(xpy,ypy)
```

Используем пакет для анализа производительности, чтобы провести сравнение:

```
import Pkg
Pkg.add("BenchmarkTools")
using BenchmarkTools
@btime a,b = find_best_fit_python(xvals,yvals)
@btime a,b = find_best_fit(xvals,yvals)
```

7.3. Задание

1. Используя Jupyter Lab, повторите примеры из раздела 7.2.
2. Выполните задания для самостоятельной работы (раздел 7.4).

7.4. Задания для самостоятельного выполнения

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.

Подсказка: вам нужно будет проиндексировать фрейм данных, преобразовать его в массив и транспонировать.

7.4.2. Регрессия (метод наименьших квадратов в случае линейной регрессии)

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);
```

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 `l1sq` из `MultivariateStats.jl` (просмотрите документацию).
- Сравните свои результаты с результатами использования регулярной регрессии наименьших квадратов из `GLM.jl`.

Подсказка. Создайте матрицу данных X_2 , 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 .

7.4.3. Модель ценообразования биномиальных опционов

Описание модели ценообразования биномиальных опционов можно найти на стр. https://en.wikipedia.org/wiki/Binomial_options_pricing_model.

Постройте траекторию возможных цен на акции:

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

- 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`, `rmpr` 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.

7.5. Содержание отчёта

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка задания работы.
3. Описание выполнения задания:
 - подробное пояснение выполняемых в соответствии с заданием действий;
 - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
 - листинги (исходный код) программ и результаты его выполнения;
4. Выводы, согласованные с заданием работы.