Uporaba programskega jezika Julia za namene podatkovne znanosti

Tadej Lahovnik, Grega Vrbančič, Iztok Fister ml., Vili Podgorelec

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
tadej.lahovnik1@um.si, grega.vrbancic@um.si,
iztok.fister1@um.si, vili.podporelec@um.si

Programski jezik Julia je v zadnjih letih pridobil veliko pozornosti zaradi svoje edinstvene kombinacije zmožnosti programiranja na visoki ravni, zmogljivosti in enostavne uporabe. Dandanes programski jezik spada med tri najpomembnejše jezike za namene podatkovne znanosti poleg jezika Python in R. Namen tega članka je zagotoviti pregled ključnih lastnosti jezika Julia in prikazati njegov potencial kot zmogljivega orodja za analizo podatkov, vizualizacijo in strojno učenje. V članku predstavimo temeljne koncepte programskega jezika Julia, vključno z njegovo sintakso, podatkovnimi strukturami in ekosistemom paketov. Dodatno se v članku posvetimo opisu bogatega nabora funkcij za manipulacijo in analizo podatkov, ki jih ponuja Julia. Poleg tega članek obravnava vizualno predstavitev podatkov z uporabo zmogljivih vizualizacijskih knjižnic programa Julia. Prikažemo, kako Julia omogoča ustvarjanje vizualno privlačnih in interaktivnih grafikonov, diagramov in nadzornih plošč, ki nam omogočajo učinkovito pridobivanje novih spoznanj, ki temeljijo na podatkih. V zaključku predstavimo še primerjavo med jeziki Julia, Python in R ter navedemo prednosti jezika Julia za podatkovne znanstvenike.

Ključne besede:
podatkovna znanost
strojno učenje
Julia
velepodatki
globoko učenje

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.24

1 Uvod

V zadnjih letih smo priča veliki rasti podatkov, ki nastajajo na različnih področjih, vse od različnih industrijskih okolij, sekvenciranja genetskih podatkov, zbiranja podatkov za analitiko in podobno [1]. Posledično se je zaradi rasti podatkov kot tudi njihove dostopnosti ter izzivov, ki jih prinašajo, močno razvilo področje podatkovne znanosti (angl. data science), kjer poskušajo podatkovni znanstveniki (angl. data scientists) in ostali deležniki izkoristiti moč odločanja na podlagi podatkov in s tem dobiti globlji vpogled v zakonitosti, ki so skrite v podatkih [2]. Zaradi tega se je v zadnjih letih razvilo tudi veliko orodij, ki olajšajo delo podatkovnim znanstvenikom ter ljudem, ki rokujejo z veliko količino podatkov. Poleg različnih komercialnih rešitev za namene spopadanja s problemi podatkovne znanosti in ostalih programskih paketov v že obstoječih programskih jezikih, so se pojavili tudi nekateri novi programski jeziki, ki so namenjeni delu s podatki, izvajanju numeričnih analiz, podatkovnemu rudarjenju, strojnemu učenju in vizualizaciji podatkov. Eden izmed njih je programski jezik Julia, ki je visokozmogljiv programski jezik posebej zasnovan za reševanje izzivov povezanih z umetno inteligenco, analitiko velikih količin podatkov in podatkovno znanost [3]. Programski jezik Julia, ki je nastal s skupnimi prizadevanji raznolike skupnosti raziskovalcev in razvijalcev, ponuja edinstveno kombinacijo preprostosti, hitrosti in prilagodljivosti, zato je idealna izbira za podatkovno-usmerjene aplikacije. Ena od ključnih gonilnih sil priljubljenosti programskega jezika Julia je njegova zmožnost učinkovitega izvajanja numeričnih in znanstvenih izračunov. S pomočjo prevajalnika JIT (just-in-time) Julia dosega zmogljivost, primerljivo s statično tipiziranimi jeziki, kot sta jezika C ali Fortran, hkrati pa zagotavlja enostavnost uporabe in dinamičnost, ki sta povezani z jeziki, ki uporabljajo tolmač. Ta prednost izhaja iz Juliinega inovativnega sistema tipov in večkratnega pošiljanja, ki omogočata učinkovito generiranje in optimizacijo kode. Na področju umetne inteligence in strojnega učenja se zmožnosti programskega jezika Julia kažejo z obsežnim ekosistemom paketov in knjižnic, ki so dostopne preko Juliinega repozitorija paketov[1]. Julia ponuja veliko orodij, ki raziskovalcem in praktikom omogočajo hitro izdelavo prototipov in uvajanje modelov umetne inteligence, ne glede na to, ali gre za gradnjo globokih nevronskih mrež, izvajanje algoritmov za obdelavo naravnega jezika ali raziskovanje najsodobnejših tehnik okrepitvenega učenja. Poleg tega se Julia brez težav povezuje s priljubljenimi ogrodji, kot sta TensorFlow in PyTorch, kar omogoča interoperabilnost in izkorišča prednosti različnih ekosistemov.

V tem strokovnem članku bomo predstavili osnovne značilnosti in zmogljivosti programskega jezika Julia, poudarili njegove prednosti in razpravljali o tem, kako se umešča v širši kontekst sodobnih metod umetne inteligence, analitike velikih količin podatkov, podatkovne znanosti in naraščajočih zahtev po razložljivi umetni inteligenci (angl. Explainable Artificial Intelligence, krajše XAI) [4].

Struktura članka je naslednja: v drugem poglavju so predstavljene glavne značilnosti programskega jezika Julia. V tretjem poglavju je prikazana uporaba programskega jezika Julia za namene podatkovne znanosti. V zadnjem četrtem poglavju je narejen povzetek članka.

[1] https://juliapackages.com

2 Programski jezik Julia

Julia je odprtokoden, visoko zmogljiv, ekspresiven in sodoben programski jezik za znanstveno računalništvo in obdelavo podatkov. Začetek razvoja programskega jezika Julia sega v leta 2009. Prva stabilna verzija (v1.0) je bila izdana leta 2018, najnovejša verzija (v1.9.2) pa julija 2023 [3], [5], [6]. Glavne značilnosti programskega jezika Julia so:

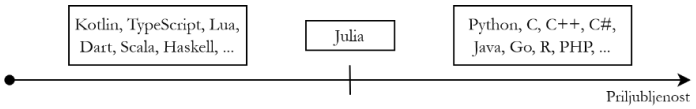
- večkratno pošiljanje (angl. multiple dispatch), kar pomeni posplošitev polimorfnega mehanizma, ki se uporablja v objektno-orientiranih programskih jezikih (OOP) [7],
- dinamično tipiziran jezik, kar pomeni, da tipov spremenljivk ni treba deklarirati v kodi. Poudariti je treba, da Julia ne omogoča statičnega preverjanja tipov, zaradi česar se lahko med izvajanjem pojavi t. i. »type error« [6],
- JIT prevajalnik z uporabo ogrodja LLVM[2], ki skupaj z zasnovo jezika Julii omogoča doseganje maksimalne zmogljivosti za numerično, tehnično in znanstveno računanje [6],
- vzporedno računanje (angl. parallel computing), ki uporabnikom omogoča prekinitev in nadaljevanje izračunov s popolnim nadzorom nad komunikacijo, brez potrebe po ročni interakciji z razporejevalnikom (angl. scheduler) operacijskega sistema [3].

V tabeli 2 je predstavljena osebna izkaznica programskega jezika Julia [3], [7], [8].

Tabela 2: Osebna izkaznica programskega jezika Julia.

Nastanek jezika	2009
Prevajalnik	JIT z uporabo ogrodja LLVM
Nastal po vzoru sledečih programskih jezikov	Python, R, C, Lisp, Lua, …
Operacijski sistem	Deluje na vseh platformah
Končnice datotek	.jl
Spletna stran	https://julialang.org
GitHub repozitorij	https://github.com/JuliaLang/julia

Priljubljenost programskega jezika dokazuje lestvica »Tiobe Index«, ki Julio uvršča na 24. mesto med vsemi programskimi jeziki. Na sliki 1 je prikazan odsek Tiobe lestvice in umestitev programskega jezika Julia.



Slika 1: Delna Tiobe lestvica
Vir: [9].

Programski jeziki Python, Java, PHP, Go in R so glede na priljubljenost prepričljivo pred Julio, a se ta uvršča pred Kotlin, TypeScript in Luo. Glavni razlog za zaostanek v priljubljenosti leži v nestabilnosti programskega jezika. Razvijalci neprenehoma pripravljajo izdaje, ki vključujejo manjše spremembe, le-te pa pogosto ne ponujajo združljivosti za nazaj (angl. backwards compatibility). Nekaj dodatnih razlogov za manjšo priljubljenost navaja spletni vir [10].

V nadaljevanju poglavja je predstavljenih nekaj preprostih in kratkih primerov uporabe programskega jezika Julia.

[2] low-level virtual machine

2.1 Uvodni program "Hello world"

Klasični uvodni program (t. i. »Hello World«) v programskem jeziku Julia zapišemo, kot prikazuje levi stolpec tabele 3. Če želimo, da se po besedilu izpiše še prazna vrstica, lahko namesto »print« uporabimo funkcijo »println«.

Tabela 3: Hello World.

'hello.jl'	>> julia hello.jl
print("Hello World")	>> Hello World

Programsko kodo po uspešni namestitvi Julie poženemo z ukazom »julia hello.jl«. V novi vrstici terminalnega okna se nato izpiše »Hello World«. S tem preprostim zgledom smo uspešno ustvarili svoj prvi program v programskem jeziku Julia.

2.2 Funkcije

V naslednjem primeru ustvarimo preprosto funkcijo (poimenovano »kvadriranje«). Funkcija prejme parameter »st« in ga pomnoži s seboj. Celotna implementacija funkcije »kvadriranje« je prikazana v levem stolpcu tabele 4. Po definiranju funkcije je treba blok kode zaključiti s ključno besedo »end«. Desni stolpec prikazuje poganjanje datoteke »kvadriranje.jl« in izpis rezultata funkcije v ukazni vrstici.

Tabela 4: Funkcija kvadriranja.

'kvadriranje.jl' function kvadriranje(st) return st^2 end print(kvadriranje(parse(Int64, ARGS[1])))	>> julia kvadriranje.jl 5 >> 25

V tem primeru je parameter funkcije podan kot vhodni parameter ob pogonu programa. Funkciji bi lahko parameter podali tudi neposredno v programski kodi. To dosežemo s spremembo zadnje vrstice kode, kjer zamenjamo parameter funkcije »kvadriranje« (npr. »print(kvadriranje(5))«). Kodo bi nato pognali z ukazom »julia kvadriranje.jl«.

2.3 Pogojni stavki

Pogojni stavki v Julii uporabljajo ključne besede »if«, »elseif« in »else«. Podobno kot funkcije je pogojne stavke treba zaključiti s ključno besedo »end«. Primer pogojnega stavka, ki preverja sodost in lihost števila, je prikazan v levem stolpcu tabele 5.

Tabela 5: Pogojni stavki.

'pogoj.jl'
function sodoLiho(st)
  if st%2 == 0
  print("Sodo")
  else
  print("Liho")
  end
end
sodoLiho(3)
	>> julia pogoj.jl
	>> Liho

Za preproste logične izjave lahko namesto bloka kode uporabimo ternarni operator (»?«). Del kode, ki sledi ternarnemu operatorju, se izvede, če je pogoj pred ternarnim operatorjem resničen, sicer se izvede del kode, ki sledi operatorju »:«. Tabela 6 prikazuje preverjanje sodosti in lihosti števila z uporabo ternarnega operatorja.

Tabela 6: Ternarni operator.

'pogoj.jl' function sodoLiho(st) st%2 == 0 ? print("Sodo") : print("Liho") end sodoLiho(4)	>> julia pogoj.jl >> Sodo

2.4 Zanke in funkcija »map«

Znotraj zanke lahko z uporabo »continue« preskočimo iteracijo. Z uporabo »break« se izvajanje zanke prekine. Podobno kot funkcije in pogojne stavke je zanke treba zaključiti s ključno besedo »end«. S pomočjo funkcije »map« lahko vrednostim v obstoječem seznamu priredimo nove vrednosti. V tem primeru vsa števila v seznamu kvadriramo in jih shranimo v nov seznam. V levem stolpcu tebele 7 sta prikazana primera uporabe zank in funkcije »map«.

Tabela 7: Zanke in funkcija »map«.

'zanka.jl'
for i in 1:10
  if i % 2 != 1
  continue
  end
  if i >= 8
  break
  end
  print(i, " ")
end
s = map(i -> i^2, [1, 2, 3, 4, 5])
print(s)
	>> julia zanka.jl
	>> 1 3 5 7
	>> [1, 4, 9, 16, 25]

2.5 Seznami

Naslednji primer prikazuje uporabo seznamov (angl. array) v programskem jeziku Julia. Treba je poudariti, da se indeksiranje elementov v seznamu v Julii začne z 1 [11]. Tabela 8 prikazuje različne načine ustvarjanja seznamov. Seznam »s1« se ustvari skupaj s svojo vsebino, tip seznama pa se določi samodejno. Seznam »s2« je prazen in nima tipa, zaradi česar nekatere operacije nad seznami ne delujejo (npr. funkcija »push!« za dodajanje elementov). Seznam »s3« je prav tako prazen, vendar ima definiran tip. Sezname lahko ustvarimo tudi s pomočjo t. i. »list comprehensions« (npr. seznam »s4«), ki delujejo na podoben način kot funkcija map v prejšnjem primeru.

Tabela 8: Ustvarjanje seznamov.

```
julia> s1 = [1, 2, 3]
3-element Vector{Int64}: [1, 2, 3]
julia> s2 = []
0-element Vector{Any}: Any[]
julia> s3 = Int64[]
0-element Vector{Int64}
julia> s4 = [i^2 for i = 1:3]
3-element Vector{Int64}: [1, 4, 9]
```

Primer v tabeli 9 prikazuje uporabo t. i. »dequeue« funkcij nad seznami, ki jih ponuja programski jezik Julia. Med dequeue funkcije sodijo »push!«, »pop!«, »insert!«, »deleteat!«, »splice!«, »resize!«, »append!« in »prepend!«. Te omogočajo dodajanje in odstranjevanje elementov ter združevanje več seznamov v enega. Klicaj, ki sledi imenu funkcije, omogoča posodobitev prvega parametra, ki ga funkcija prejme.

Tabela 9: »Dequeue« funkcije.

```
julia> s = Any[1, 2, 3]
3-element Vector{Any}: Any[1, 2, 3]
julia> push!(s, "štiri")
4-element Vector{Any}: Any[1, 2, 3, "štiri"]
julia> pop!(s)
3-element Vector{Any}: Any[1, 2, 3]
julia> append!(s, ["štiri", "pet"])
5-element Vector{Any}: Any[1, 2, 3, "štiri", "pet"]
julia> deleteat!(s, 4)
4-element Vector{Any}: Any[1, 2, 3, "pet"]
```

2.6 Ekosistem paketov

Julia ponuja lastni upravitelj paketov, imenovan Pkg. Ta je za razliko od običajnih upraviteljev paketov, ki namestijo in upravljajo en globalni nabor paketov, zasnovan na podlagi "okolij"; neodvisnih naborov paketov, ki so lahko lokalni za posamezen projekt ali skupni in izbrani po imenu. Natančen nabor paketov in različic v okolju je zabeležen v manifestni datoteki, ki jo je mogoče preveriti v repozitoriju projekta in spremljati v sistemu za upravljanje z izvorno kodo (angl. version control), kar znatno izboljša ponovljivost projektov [12].

Upravitelj paketov Pkg omogoča enostavno nameščanje in posodabljanje paketov. Za namestitev novega paketa se uporablja ukaz »Pkg.add(»[ime paketa]«), za posodobitev paketov pa »Pkg.update()«. Nameščene pakete nato uporabimo v kodi z ukazoma »using [ime paketa]« in »import [ime paketa]«. Prvi način uvozi vse funkcije, ki jih paket ponuja, v trenutni imenski prostor, kar omogoči klicanje funkcije brez eksplicitne navedbe paketa. Pri drugem načinu je treba eksplicitno določiti, iz katerega paketa se funkcija kliče (»[ime paketa].funkcija()«). Ta način je predvsem uporaben, če se ime funkcije pojavi v več paketih [11].

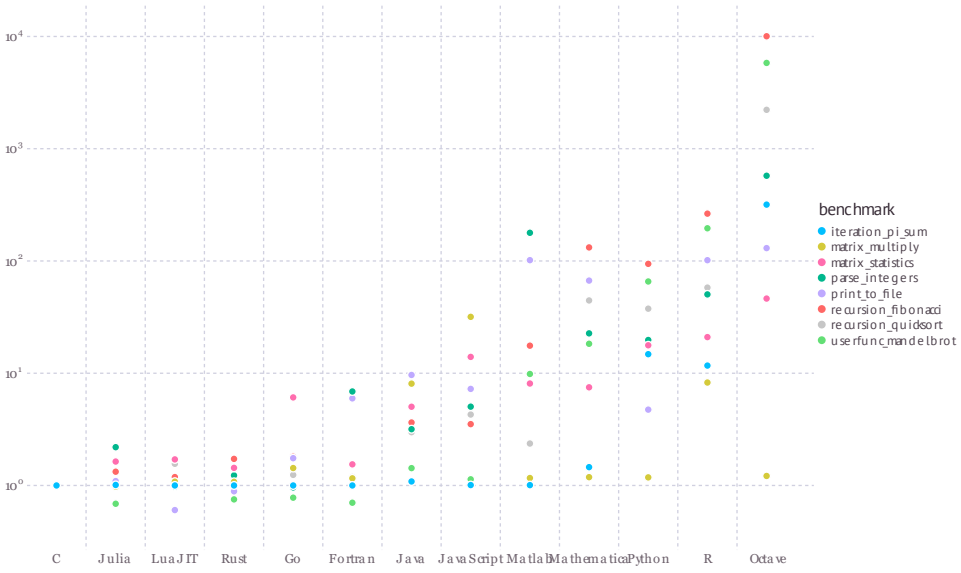
Bogat ekosistem paketov vključuje pakete za delo s podatki (npr. DataFrames.jl, CSV.jl), vizualizacijo podatkov (npr. Plots.jl, PlotlyJS.jl), strojno učenje (npr. TimeSeries.jl, Clustering.jl) in obdelavo slik (npr. Images.jl, ImageSegmentation.jl, ImageTransformations.jl) [13].

3 Programski jezik Julia in podatkovna znanost

O Pythonu se zadnja leta veliko govori. Glede na podatke indeksa Tiobe [9] je Python v letih 2018, 2020-2021 in v tem letu bil razglašen za programski jezik leta. Izjemno priljubljen je zlasti med podatkovnimi znanstveniki in strokovnjaki za strojno učenje ter se v veliki meri uporablja za umetno inteligenco.

Python je odprtokodni programski jezik; njegova preprostost in kratka krivulja učenja sta med ključnimi razlogi za njegovo priljubljenost. Rast Pythona je v zadnjem času skokovito narasla; vendar je Julia, razmeroma novejši jezik, v zadnjem času požel veliko zanimanja in iskanja še posebej v skupnosti podatkovnih znanstvenikov in ekspertov strojnega učenja. Ključen vzrok za vedno večje zanimanje v prej omenjeni skupnosti je hitrost izvajanja, ki je še kako pomembna pri časovno zahtevnih algoritmih strojnega učenja in obdelavi velikih količin podatkov.

Obstajajo različne primerjalne študije za primerjavo hitrosti Julie z drugimi programskimi jeziki. Prvi primer, ki ga lahko omenimo, je z uradne spletne strani jezika Julia [14]. V tem poskusu je osem algoritmov izvedenih v 12 različnih jezikih. Julia je bila med najuspešnejšimi v primerjavi s C, Lua in Rust. Presenetljivo je, da so najslabše rezultate v raziskavi dosegli jeziki, ki so jih najpogosteje uporabljali strokovnjaki s področja podatkovne znanosti in strojnega učenja: Matlab, Mathematica, Python, R in Octave v padajočem vrstnem redu (slika 2).



Slika 2: Primerjava hitrosti izvajanja različnih algoritmov v različnih programskih jezikov.
Vir: [14].

Avik Sengupta v knjigi Julia High Performance [15] primerja zmogljivost 10 programskih jezikov s C za izračun Mandelbrotove množice. Tudi v tem primeru je Julia med najuspešnejšimi z jeziki Fortran, Javascript in Lua. Po drugi strani pa so najmanj uspešni jeziki numeričnega računanja in podatkovne znanosti: Mathematica, Matlab, Python in R v padajočem vrstnem redu.

V nadaljevanju bomo predstavili ključne podobnosti oz. razlike iz vidika podatkovne znanosti med programskima jezikoma Julia in Python. Čeprav bo primerjava v splošnem med omenjenima programskima jezikoma, lahko te apliciramo tudi na programski jezik R, saj ta iz stališča izvajanja deluje podobno kot Python.

3.1 Branje podatkov

Prva naloga v vsakem delovnem postopku analize podatkov je preprosto branje podatkov, kar je treba nujno opraviti hitro in učinkovito, da se lahko začne bolj zanimivo delo. V številnih panogah in področjih je datotečni format CSV kralj za shranjevanje in izmenjavo tabelaričnih podatkov. Hitro in zanesljivo nalaganje datotek CSV je ključnega pomena, poleg tega pa mora biti dobro skalirano za različne velikosti datotek, vrste podatkov in oblike. V tem prispevku primerjamo zmogljivost branja 8 različnih naborov podatkov iz resničnega sveta s tremi različnimi razčlenjevalniki CSV: fread iz programa R, read_csv iz programa Pandas in CSV.jl iz programa Julia[3]. Vsak od njih je bil izbran kot »najboljši v svojem razredu« razčlenjevalnik (angl. parser) datotek CSV v programskih jezikih R, Python in Julia. Večnitnost je bistvenega pomena za doseganje največje zmogljivosti današnjih računalnikov, vendar je le en jezik (Julia) lahko dosledno in učinkovito uporabljal več jeder.

Vsa tri orodja imajo zanesljivo podporo za nalaganje najrazličnejših podatkovnih vrst s potencialno manjkajočimi vrednostmi, vendar samo knjižnici fread (R) in CSV.jl (Julia) podpirata večnitno nalaganje medtem ko Pandas podpira samo enonitno nalaganje datotek CSV. Knjižnica CSV.jl je posebna tudi zato, ker je edina, ki je v celoti implementirana v enakem programskem jeziku in ni implementirana v katerem izmed nižjenivojskih programskih jezikov (npr. C) ter zgolj ovita v programskem jeziku Python ali R. Pandas ima sicer nekoliko zmogljivejši razčlenjevalnik, ki temelji na jeziku Python, vendar je bistveno počasnejši in skoraj vse uporabe funkcije read_csv privzeto uporabljajo C implementacijo. Zato takšna primerjava ne predstavlja zgolj prikaza hitrosti nalaganja podatkov v programskem jeziku Julia, temveč nakazuje tudi na splošno zmogljivost programskega jezika.

[3] Opis eksperimenta skupaj z rezultati je na voljo na https://www.queryverse.org/benchmarks

Primerjava s stališča hitrosti izvajanja nalaganja podatkov kažejo, da je CSV.jl v programskem jeziku Julia 1,5- do 5-krat hitrejši od Pandas, tudi če je omejen na eno jedro; z omogočeno večnitnostjo je lahko več kot 20-krat hitrejši. R-jev fread je bolj konkurenčen, saj podpira večnitnost, vendar v številnih podatkovnih nizih še vedno zaostaja za Julio za 10-krat ali več. Za primerjalno analizo so bila uporabljena naslednja orodja: BenchmarkTools.jl za Julio, microbenchmark za R in timeit za Python.

3.2 Obdelava podatkov

Ko govorimo o obdelavi podatkov, imamo v mislih obdelavo strukturiranih podatkov, ki so navadno podani v obliki datotek CSV. V programskem jeziku Python je prva izbira za ta namen uporaba ene izmed najpopularnejših knjižnic; Pandas. Alternativa omenjeni knjižnici v programskem jeziku Julia je DataFrames.jl, ki je široko uporabljena knjižnica za obdelavo in analizo podatkov. Zagotavlja podobne funkcije kot Pandas v Pythonu in omogoča delo s strukturiranimi podatki v tabelarični obliki.

3.2.1 Ustvarjanje podatkovnih okvirjev

Za namen primerjave uporabe vzemimo ustvarjanje osnovnega podatkovnega okvirja (angl. DataFrame) v obeh programskih jezikih. Najprej zgradimo enostaven podatkovni okvir s tremi značilnicami: ID, ime in starost v programskem jeziku Julia (slika 3). V ta namen ustvarimo primerek razreda DataFrame, ki mu podamo vhodne parametre.

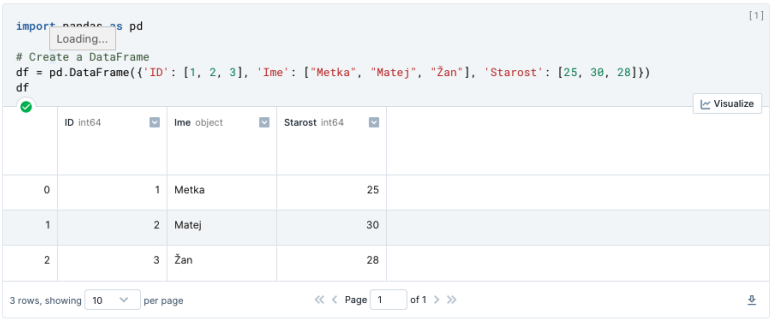
```
using DataFrames

df = DataFrame(ID=[1, 2, 3], Ime=["Metka", "Matej", "Žan"], Starost=[25, 30, 28])
```

3×3 DataFrame

Row	ID	Ime	Starost
	Int64	String	Int64
1	1	Metka	25
2	2	Matej	30
3	3	Žan	28

Slika 3: Primer ustvarjanja podatkovnega okvirja v programskem jeziku Julia.

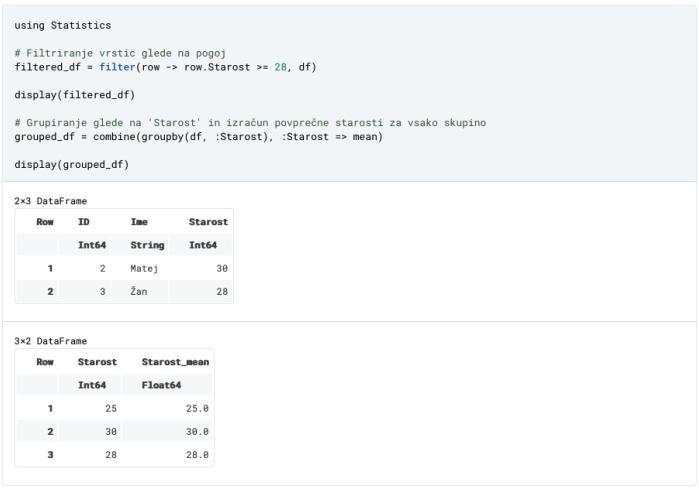
Enako nato ponovimo tudi v programskem jeziku Python (slika 4). Kot je razvidno, je implementacija izgradnje enostavnega podatkovnega okvirja praktično enaka. Razlika je zgolj v načinu podajanja parametrov konstruktorju razreda DataFrame, končen rezultat pa je enak.



Slika 4: Primer ustvarjanja podatkovnega okvirja v programske jeziku Python.

3.2.2 Manipulacija nad podatki

V procesu obdelave podatkov je vsekakor ključna možnost manipulacije nad podatki, ki v splošnem vključuje operacije zapolnjevanja manjkajočih podatkov, transformiranje, preoblikovanje, filtriranje, grupiranje, agregiranje, itd. podatkov. Za namen prikaza uporabe bomo prikazali uporabo filtriranja in grupiranja ter izračuna povprečnih vrednosti. Slika 5 prikazuje uporabo prej omenjenih funkcij nad v sekciji 3.2.1 ustvarjenim podatkovnim okvirjem v programskem jeziku Julia.



Slika 5: Uporaba osnovnih operacij filtriranja, grupiranja in izračuna povprečnih vrednosti nad podatkovnim okvirjem v programskem jeziku Julia.

Implementacija enakega primera v programskem jeziku Python je razvidna na sliki 6, ki še enkrat več prikazuje, da je doseganje enakega cilja pri uporabi omenjenih knjižnic precej podobno oz. se razlikuje zgolj v osnovni sintaksi primerjanih programskih jezikov.

Slika 6: Uporaba osnovnih operacij filtriranja, grupiranja in izračuna povprečnih vrednosti nad podatkovnim okvirjem v programskem jeziku Python.

3.2.3 Hitrost izvajanja

Za namen preverjanja oz. primerjave hitrosti izvajanja najbolj pogostih operacij nad podatkovnim okvirjem smo si zastavili enostaven eksperiment (slika 7). V vsakem izmed okolij smo ustvarili podatkovni okvir z milijon naključnimi vnosi števil med 20 in vključno 50. Nad ustvarjenim podatkovnim okvirjem smo nato izvedli testiranje hitrosti izvedbe za filtriranje in grupiranje ter izračun povprečne vrednosti starosti v vsaki izmed skupin. Dekorator @benchmark v programskem jeziku Julia omogoča samodejno beleženje časa izvajanja podanega ukaza oz. klica funkcije ter po koncu izvedbe tudi izpisa izmerjenih vrednosti.

Slika 7: Prikaz hitrosti izvajanja operacij nad podatkovnim okvirjem v programskem jeziku Julia.

Slika 8 prikazuje implementacijo enakega eksperimenta v programskem jeziku Python. Za merjenje časa izvajanja posamezne operacije nad podatkovnim okvirjem smo v tem primeru uporabili knjižnico timeit. Glede na izpisane rezultate izvedb obeh implementacij lahko ugotovimo, da je filtriranje v programskem jeziku Julia v povprečju trajalo 66 milisekund, medtem ko je enaka operacija v Pythonu trajala slabo sekundo. Še večja razlika, ki govori v

korist programskemu jeziku Julia, se je pokazala pri grupiranju in izračunu povprečne vrednosti. Ta operacija se je v programskem jeziku Julia izvedla v manj kot 3 milisekundah, na drugi strani pa je Python za to potreboval 2 sekundi.

Glede na to, da že pri popolnoma preprostem primeru, kot je naš, lahko opazimo ogromno prednost programskega jezika Julia, je mogoče pričakovati, da se hitrost izvedbe še bolj izrazito opazi pri manipulaciji nad kompleksnejšimi podatkovnimi strukturami ter operacijami.

[7]

```python
import pandas as pd
import numpy as np
import timeit

# Create a large DataFrame
df = pd.DataFrame({'ID': range(1, 10**6 + 1), 'Age': np.random.randint(20, 51, 10**6)})

# Filtering rows based on a condition
def filter_condition():
    return df[df['Age'] >= 40]

# Grouping and calculating the mean age
def group_and_mean():
    return df.groupby('Age').mean().reset_index()

# Benchmark filtering
filter_time = timeit.timeit(filter_condition, number=100)

# Benchmark grouping and mean calculation
group_time = timeit.timeit(group_and_mean, number=100)

print(f"Filtering time (Python): {filter_time:.5f} seconds")
print(f"Grouping and Mean calculation time (Python): {group_time:.5f} seconds")
```

```
Filtering time (Python): 0.98302 seconds
Grouping and Mean calculation time (Python): 2.02604 seconds
```

Slika 8: Prikaz hitrosti izvajanja nad podatkovnim okvirjem v programskem jeziku Julia.

3.3 Vizualiziranje podatkov

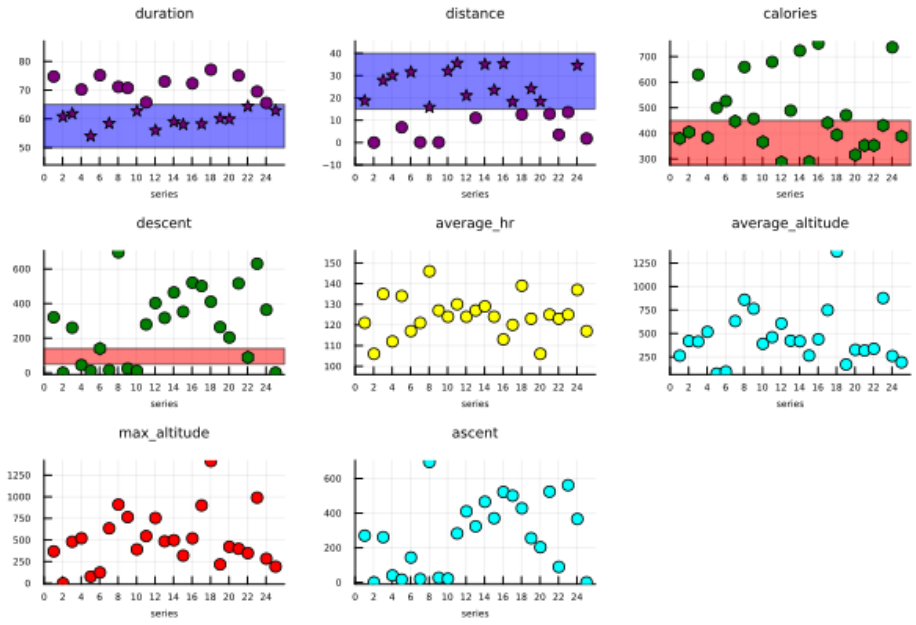
Vizualizacija podatkov je po navadi zadnji korak pri odkrivanju znanja iz podatkov oziroma analize podatkov. Namen vizualiziranja oziroma vizualizacije podatkov je grafična predstavitev informacij ter podatkov. Je učinkovita tehnika za posredovanje zapletenih podatkovnih vzorcev in vpogledov z vizualnimi elementi, kot so preglednice, grafi, diagrami in pismenke (angl. glyphs). Programski jezik Julia se je na področju vizualizacije podatkov izkazal kot odlično orodje za ustvarjanje enostavnih kot tudi kompleksnih ter interaktivnih vizualizacij.

Znotraj ekosistema programskega jezika Julia lahko najdemo različne pakete in knjižnice, posebej zasnovane za različne naloge vizualizacije podatkov. Najpomembnejša knjižnica je verjetno knjižnica Plots.jl, ki zagotavlja poenoten vmesnik za ustvarjanje številnih grafikonov z minimalno kodo. Plots.jl podpira več zaledij, kot so GR, Plotly in PyPlot, kar uporabnikom omogoča, da izberejo najprimernejši način vizualizacije za svoje potrebe. Primer uporabe knjižnice Plots.jl za vizualizacijo odkritega znanja iz asociacijskih pravil je prikazan na sliki 9.

Drugo zelo priljubljeno ogrodje za vizualizacijo podatkov v Julii je Makie.jl. Makie ponuja visoko zmogljive interaktivne grafe, ki so idealni za obdelavo velikih zbirk podatkov ter ustvarjanje vizualizacij v realnem času. Njegove interaktivne zmožnosti omogočajo nemoteno raziskovanje in manipuliranje s podatkovnimi točkami neposredno znotraj ploskve.

Poleg tega Julia podpira integracijo s priljubljenimi knjižnicami za izrisovanje iz drugih programskih jezikov, kot sta Matplotlib preko PyCall.jl [17] in Plotly preko PlotlyJS.jl knjižnic. To uporabnikom omogoča, da uporabijo obstoječe knjižnice za vizualizacijo in jih nemoteno združijo z zmogljivostmi analize podatkov, ki jih ponuja Julia.

Z izkoriščanjem zmogljivosti vizualizacije podatkov programskega jezika Julia in uporabo njegovega bogatega ekosistema ogrodij in knjižnic lahko uporabniki odkrijejo globlje vpoglede in svoje ugotovitve učinkovito sporočijo širšemu občinstvu.



Slika 9: Uporaba knjižnice Plots.jl za vizualizacijo odkritega znanja iz asociacijskih pravil
Vir: [16].

3.4 Strojno učenje

Ekosistem knjižnic za strojno učenje v programskem jeziku Julia iz leta v leto narašča. Ena ključnih knjižnic je Flux.jl, ki zagotavlja podporo za globoko učenje in nevronske mreže. Flux.jl [18] ponuja samodejno diferenciacijo, podporo za grafične procesorje in jedrnato sintakso, ki spominja na matematični zapis. Flux.jl nudi nizkonivojski način implementacije algoritmov strojnega učenja, ki sicer uporabniku nudi možnost neposrednega nadzora nad učenjem, po drugi strani pa je za začetnika morebiti prezahteven. Za enostavnejšo splošno uporabo je bila razvita knjižnica FastAI.jl [19], ki omogoča uporabo algoritmov strojnega učenja v le nekaj vrsticah programske kode (Slika 9: Primer izgradnje napovednega modela za klasifikacijo slik z uporabo XresNet arhitekture globoke nevronske mreže.

V splošnem sta tako Julia kot tudi Python povsem legitimni izbiri za uporabo pri strojnem učenju, vsaka s svojimi prednostmi. Julia se odlikuje pri visokozmogljivem računalništvu in numeričnih nalogah, zato je primerna za aplikacije, kjer je hitrost ključnega pomena. Po drugi strani pa je Python, prav zaradi njegove enostavnosti in nizke krivulje učenja, dobra izbira za različne profile uporabnikov, saj je kos različnim nalogam strojnega učenja.

Poleg omenjene knjižnice Flux.jl velja omeniti še knjižnico MLJ.jl [20], ki je celovita knjižnica za strojno učenje, katere namen je zagotoviti dosleden vmesnik za različne naloge strojnega učenja, vključno s predobdelavo, modeliranjem, vrednotenjem in cevovodi. MLJ.jl se dobro povezuje z drugimi paketi Julia in zagotavlja enoten način dela z različnimi modeli in vrstami podatkov. Prav tako velja omeniti še knjižnico Turing.jl [21] za verjetnostno programiranje, ki omogoča definiranje in ocenjevanje verjetnostnih modelov z uporabo visokonivojske sintakse. Podpira različne inferenčne algoritme, vključno z algoritmom MCMC (Markov Chain Monte Carlo), zato je primerna za bayesovsko analizo podatkov in verjetnostno modeliranje.

Nasproti rastočemu ekosistemu knjižnic za strojno učenje stoji Python kot uveljavljen programski jezik skupaj z naborom široko uporabljanih in uveljavljenih knjižnic. Slabost slednjega je seveda interpretativna narava Pythona, vendar knjižnice, kot so NumPy, TensorFlow in PyTorch, s svojo nizkonivojsko implementacijo to kar se da omilijo. Ne nazadnje, v kolikor je Tensorflow naša prva izbira, ko je govora o knjižnici za globoko učenje, je za slednjo na voljo tudi vmesnik za programski jezik Julia, ki omogoča domorodno uporabo knjižnice.

```
using FastAI
data, blocks = loaddataset("imagenette2-160", (Image, Label))
method = ImageClassificationSingle(blocks)
learner = methodlearner(method, data, Models.xresnet18(), ToGPU(), Metrics(accuracy))
fitonecycle!(learner, 10, 0.002)
```

Slika 9: Primer izgradnje napovednega modela za klasifikacijo slik z uporabo XresNet arhitekture globoke nevronske mreže.

V splošnem sta tako Julia kot tudi Python povsem legitimni izbiri za uporabo pri strojnem učenju, vsaka s svojimi prednostmi. Julia se odlikuje pri visokozmogljivem računalništvu in numeričnih nalogah, zato je primerna za aplikacije, kjer je hitrost ključnega pomena. Po drugi strani pa je Python, prav zaradi njegove enostavnosti in nizke krivulje učenja, dobra izbira za različne profile uporabnikov, saj je kos različnim nalogam strojnega učenja.

Poleg omenjene knjižnice Flux.jl velja omeniti še knjižnico MLJ.jl [20], ki je celovita knjižnica za strojno učenje, katere namen je zagotoviti dosleden vmesnik za različne naloge strojnega učenja, vključno s predobdelavo, modeliranjem, vrednotenjem in cevovodi. MLJ.jl se dobro povezuje z drugimi paketi Julia in zagotavlja enoten način dela z različnimi modeli in vrstami podatkov. Prav tako velja omeniti še knjižnico Turing.jl [21] za verjetnostno programiranje, ki omogoča definiranje in ocenjevanje verjetnostnih modelov z uporabo visokonivojske sintakse. Podpira različne inferenčne algoritme, vključno z algoritmom MCMC (Markov Chain Monte Carlo), zato je primerna za bayesovsko analizo podatkov in verjetnostno modeliranje.

Če ne želimo sprejemati kompromisov s stališča hitrosti izvajanja, nam je na voljo tudi možnost, da znotraj Julia ekosistema uporabljamo Pythonove knjižnice strojnega učenja. To je mogoče doseči z interoperabilnostjo med obema jezikoma. V ta namen je na voljo nekaj knjižnic, ki nam uporabo Python knjižnic v programskem jeziku Julia povsem poenostavijo. Prva izmed knjižnic, ki smo jo omenili že v poglavju o vizualizaciji, je PyCall, ki omogoča klicanje funkcij Pythona in uporabo knjižnic neposredno iz Julie. Preprost primer uporabe scikit-learn, priljubljene knjižnice strojnega učenja Python, iz programskega okolja Julie je prikazan na sliki 10.

.

```
using PyCall

# Vključitev knjižnice
@pyimport sklearn.linear_model as lm

# Kreiranje modela linearne regresije
model = lm.LinearRegression()

# Učenje modela
X = [[1.0, 2.0], [2.0, 3.0]]
y = [3.0, 5.0]
model.fit(X,y)

# Napovedovanje
model.predict(X)
```

```
2-element Vector{Float64}:
 3.0
 5.0
```

Slika 10: Primer uporabe Python knjižnice scikit-learn znotraj programskega okolja Julia.

Še en primer je knjižnica Conda.jl. Conda je sicer poznana kot upravitelj paketov in razvojnih okolij za Python. Knjižnica Conda.jl nam omogoča, da znotraj ekosistema Julie ustvarimo okolje Python in namestimo Python knjižnice. V povezavi s prej predstavljeno knjižnico PyCall.jl lahko enostavno upravljamo množico Python knjižnic in jih brez težav uporabljamo znotraj programskega okolja Julia.

Čeprav se ekosistem strojnega učenja v Julii še razvija, ima več zmogljivih paketov, kot so Flux.jl za globoko učenje, MLJ.jl za splošno strojno učenje in Turing.jl za verjetnostno programiranje. Te knjižnice lahko v Julii pokrivajo široko paleto nalog strojnega učenja.

Neobstoj ali nezrelost knjižnic tako nista več razloga za neuporabo programskega jezika Julia za namen strojnega učenja. Na koncu je izbira med uporabo knjižnic za strojno učenje v jeziku Python iz Julie ali uporabo izvornih knjižnic Julie odvisna od dejavnikov, kot so zapletenost projekta, zahteve glede zmogljivosti, razpoložljivost posebnih funkcij in strokovno znanje ekipe v obeh jezikih.

4 Zaključek

V tem prispevku smo na kratko predstavili programski jezik Julia in njegove značilnosti. Julia je jezik, namenjen znanstvenemu računalništvu in obdelavi podatkov. Odlikujeta ga JIT prevajalnik z uporabo ogrodja LLVM, ki omogoča doseganje maksimalne zmogljivosti za numerično, tehnično in znanstveno računanje, ter vzporedno računanje, zaradi česar ocenjujemo, da je jezik zelo primeren za uporabo na področju podatkovne znanosti in obdelave podatkov. Čeprav se ekosistem programskega jezika Julia še razvija, ima več zmogljivih knjižnic, kot so Flux.jl za globoko učenje, MLJ.jl za splošno strojno učenje in Turing.jl za verjetnostno programiranje. S temi knjižnicami lahko v Julii pokrijemo širok razpon nalog strojnega učenja. Morebitne težave glede zmanjšanega nabora knjižnic pa lahko naslovimo z uporabo knjižnic, kot sta Conda.jl in PyCall.jl, ki omogočata uporabo širokega nabora Pythonovih knjižnic ob minimalnem žrtvovanju zmogljivosti.

Zaključimo lahko, da je programski jezik Julia legitimna izbira za namen izvajanja podatkovne znanosti z zrelimi knjižnicami, ki pokrivajo ključen nabor nalog podatkovnih znanstvenikov. V primerjavi s programskim jezikom Python mu lahko očitamo malenkost strmejšo krivuljo učenja in manjši nabor virov za učenje. Omeniti še velja, da je Python poleg enostavnosti, še posebej mikaven z razširjenostjo izvajalnega okolja Jupyter, ki uporabniku omogoča uporabo brez namestitve na lasten sistem. Julia ima sicer na voljo enako izvajalno okolje, vendar večina največjih ponudnikov Julie še nima v naboru izvajalnih okolij, kar je lahko dodaten minus. Če potrebujemo izvajanje na ravni nizkonivojskih programskih jezikov, pa Julia, ne glede na omenjeno, predstavlja zelo primerno okolje s poznano sintakso široko uporabljanih interpreterskih jezikov.

Literatura

[1] S. Sagiroglu in D. Sinanc, „Big data: A review“, v 2013 international conference on collaboration technologies and systems (CTS), 2013, str. 42–47.

[2] V. Dhar, „Data science and prediction“, Commun ACM, let. 56, št. 12, str. 64–73, 2013.

[3] K. Gao, G. Mei, F. Piccialli, S. Cuomo, J. Tu, in Z. Huo, „Julia language in machine learning: Algorithms, applications, and open issues“, Comput Sci Rev, let. 37, str. 100254, 2020.

[4] A. B. Arrieta idr., „Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI“, Information fusion, let. 58, str. 82–115, 2020.

[5] „The Julia Programming Language“. https://julialang.org (pridobljeno 25. julij 2023).

[6] I. Balbaert, A. Sengupta, in M. Sherrington, Julia: High Performance Programming. Packt Publishing Ltd, 2016.

[7] „Julia (programming language)“. https://en.wikipedia.org/wiki/Julia_(programming_language) (pridobljeno 25. julij 2023).

[8] J. Bezanson, S. Karpinski, V. B. Shah, in A. Edelman, „Julia: A fast dynamic language for technical computing“, arXiv preprint arXiv:1209.5145, 2012.

[9] „TIOBE Index for July 2023“. https://www.tiobe.com/tiobe-index (pridobljeno 24. julij 2023).

[10] „Why is Julia not more popular?“ https://craftofcoding.wordpress.com/2020/09/24/why-is-julia-not-more-popular (pridobljeno 25. julij 2023).

[11] „Julia by Example“. https://juliabyexample.helpmanual.io (pridobljeno 27. julij 2023).

[12] „Pkg.jl“. https://pkgdocs.julialang.org/v1 (pridobljeno 27. julij 2023).

[13] E. Roesch idr., „Julia for biologists“, Nat Methods, let. 20, št. 5, str. 655–664, 2023.

[14] „Julia Micro-Benchmarks“. https://julialang.org/benchmarks (pridobljeno 27. julij 2023).

[15] A. Sengupta in A. Edelman, Julia High Performance: Optimizations, distributed computing, multithreading, and GPU programming with Julia 1.0 and beyond, 2nd Edition. Packt Publishing, 2019. [Na spletu]. Dostopno na: https://books.google.si/books?id=etacDwAAQBAJ

[16] I. Fister Jr, „NarmViz.jl“. https://raw.githubusercontent.com/firefly-cpp/NarmViz.jl/main/.github/figures/Fig2.png (pridobljeno 2. avgust 2023).

[17] „PyCall.jl“. https://github.com/JuliaPy/PyCall.jl (pridobljeno 3. avgust 2023).

[18] „Flux.jl“. https://fluxml.ai/Flux.jl/stable (pridobljeno 3. avgust 2023).

[19] „FastAI.jl“. https://github.com/FluxML/FastAI.jl (pridobljeno 3. avgust 2023).

[20] „MLJ.jl“. https://github.com/alan-turing-institute/MLJ.jl (pridobljeno 3. avgust 2023).

[21] „Turing.jl“. https://turing.ml (pridobljeno 3. avgust 2023).