Uvod

Ova skripta napravljena je za potrebe kolegija Statistika koji se predaje na drugoj godini preddiplomskih studija Informatike, Računarstva i Proizvodnog strojarstva na Sveučilištu Jurja Dobrile u Puli. Kolegij u osnovi obuhvaća gradivo koje se često predaje u formi kolegija Uvod u vjerojatnost i statistiku. Ideja kolegija je da se uvedu osnovne statističke metode kroz jezik vjerojatnosti. Takav pristup uvođenju statistike je danas uobičajen i ne treba posebno obrazlaganje. S obzirom da se kolegij predaje na drugoj godini preddiplomskog studija te se ne predaje na studiju matematike,teorija vjerojatnosti je predstavljena bez teorije mjere. Takav odabir materijala donekle utječe na sadržaj kolegija,kao i na literaturu na koju se skripta oslanja. Naime,na hrvatskom jeziku je(i dalje) najpotpunija knjiga o vjerojatnosti ona Nikole Sarape[7],ali je njen obuhvat daleko iznad razine koja je potrebna za uvođenje osnovnih pojmova vjerojatnosti i statistike.Stoga je odabir referentne literature prilagođen razini općenitosti na kojoj se kolegij predaje.Neke od knjiga koje je autor češće koristio u pripremi ove skripte su Blitzstein , Hwang:Introduction to Probability [3] i Ross:Intro duction to Probability and Statistics for Engineers and Scientists[8] na engleskom te knjiga Benšić, Šuvak: Uvod u vjerojatnost i statistiku[2] i skripta Basrak:Statistika[1] na hrvatskomjeziku. Kao reference povremeno koristimo i knjige Feller:An Introduction to Probability Theory and Its Applications [5] te Wasserman:All of statistics[10], kao i neke druge koje su navedene unutar teksta skripte.

Poglavlje 1.

Alati

Originalna je skripta pisana koristeći R programski jezik, a ovdje ćemo dati ekvivalentne primjere u Pythonu, koristeći njegove biblioteke poput Pandasa, Matplotliba ili Matha.

1.1 - osnovne matematičke operacije u Pythonu

Iako Python i po defaultu pruža mogućnost odrađivanja najosnovnijih matematičkih operacija, podržava i posebnu biblioteku Math čijim "uvozom" dobivamo pristup znatno bogatijem asortimanu matematičkih funkcionalnosti.

Tako, primjerice, možemo obavljati osnovne operacije:

- zbrajanja, oduzimanja, množenja i dijeljenja(cjelobrojnog i decimalnog)

```
print(2+3)
```

Daje rezultat: 5

```
print(3-2)
```

Daje rezultat: 1

```
print(2*5)
```

Daje rezultat: 10

```
print(10/3)
```

Daje rezultat: 3

```
print(10//3)
```

Daje rezultat: 3.3333

Također, moguće je koristiti i naprednije matematičke funkcije poput:

- sinusa i kosinusa

```
import math
rezultat = math.sin(0.5)**2 + math.cos(0.5)**2
print(rezultat)
```

Daje rezultat: 1

- eksponencijalne funkcije

```
import math
rez = math.exp(-float('inf'))
print(rez)
```

Daje rezultat:

0.0

Dakle, ovdje smo pokušali izračunati eksponencijalnu funkciju za - beskonačno što se sve može postići pomoću Pythonove math biblioteke

- rada s brojem PI

```
import math
```

pi = math.pi

```
Možemo i definirati broj decimala u ispisu

~~~Python
pi_precision_ten = format(pi, '.10f') # odredili smo da želimo 10 decimalnih mjesta
print(pi_precision_ten)
Daje rezultat:
3.141592654
```

Nadalje, moguće je dodijeliti vrijednost broja PI nekoj varijabli:

```
import math

x = math.pi
```

S tom varijablom možemo kasnije raditi što god želimo, primjerice, pribrojiti joj neki drugi broj

```
x = math.pi +1
print(x)
```

```
Daje rezultat:
4.141592653589793
```

- rad s kompleksnim brojevima

Python podržava i dodjeljivanje kompleksnih brojeva varijablama, pa tako možemo definirati i kompleksnu varijablu komp1.

```
num1 = 2
num2 = 2.5
komp1 = num1 + 2i (definirali smo kompleksni broj 2+2i)
```

Za svaku varijablu možemo lako provjeriti kojeg je tipa pomoću ključne riječi type

```
print(type(num1)) #int
print(type(num2) # float
print(type(komp1) # complex
```

- rad sa riječima

Python za rad s riječima koristi stringove

```
rijec = "abc"
print(rijec) - ispisat će se abc
print(type(rijec)) - ispisat će se <class 'str'>
```

- rad s logičkim vrijednostima

```
num1 = 2
num2 = 2.5

rezultat = num1 > num2

print(type(rezultat)) - ispisat će se <class 'bool'>
print(rezultat) - ispisat će se false
```

- varijablama možemo dodijeliti logičke vrijednosti (VAŽNA NAPOMENA: u Pythonu se bool vrijednosti pišu VELIKIM početnim slovom)

```
log1 = True
log2 = True
log3 = False

print(type(log1)) - ispisat će se <class 'bool'>
print(type(log2)) - ispisat će se <class 'bool'>
print(type(log3)) - ispisat će se <class 'bool'>
```

Također, možemo obavljati osnovne operacije s logičkim vrijednostima (OR, AND, NOT)

```
print(log1 + log2) - ispisat će se 2     (True = 1, False = 0)
print(log1 or log3) - ispisat će se True
print(log1 and log3) - ispisat će se False
print(not log1) - ispisat će se False
```

- liste

Ekvivalent vektora u R-u su liste u Pythonu. Pomoću njih se kreiraju varijable čija je duljina veća od 2. Definiraju se pomoću uglatih zagrada ([])

```
num1 = 2
num2 = 2.5

li = [num1, num2]
print(li) - ispisat će [2, 2.5]
```

Sljedeća važna stavka koju treba obraditi su Python tipovi i strukture podataka.

Python podržava različite tipove podataka i strukture podataka koje omogućavaju organizaciju i pohranu podataka. Svaki tip podataka u Pythonu odgovara određenom načinu interpretacije i manipulacije vrijednostima.

1. **Brojevi (Numbers):** Python podržava različite vrste brojeva, uključujući cjelobrojne brojeve (`int`) i brojeve s pomičnim zarezom (`float`). Na primjer:

```
integer_num = 42
float_num = 3.14
```

2. **Liste (Lists):** Liste su osnovna struktura podataka koja može sadržavati elemente različitih tipova. Ovo je ekvivalent vektorima u R-u. Na primjer:

```
lista_svega = [1, R, 2, 3, 'Python', 4.5, 'miksvegapomalo']
```

3. **Nizovi (Arrays):** NumPy, popularna Python biblioteka, omogućava rad s višedimenzionalnim nizovima koji sadrže elemente istog tipa, poput brojeva. Nizovi su učinkoviti za numeričke operacije.

4. **Stringovi**: Stringovi su nizovi znakova koje koristimo za pohranu teksta

```
my_string = "Ovo sadrži tekst!"
```

5. **Rječnici (Dictionaries):** Rječnici su strukture podataka koje omogućavaju pohranu ključeva i njima pridruženih vrijednosti. Svaka vrijednost se može dohvatiti pomoću odgovarajućeg ključa.

```
dictionary_rj = {'ime': 'Mirko', 'prezime': 'Marić', 'dob': 66}
```

6. **Skupovi (Sets):** Skupovi su kolekcije jedinstvenih elemenata i koriste se za matematičke operacije nad skupovima

```
skup = {1, 2, 3, 7, 9}
```

7. **N-torke (N-tuples):** N-torka je nepromjenjiva (immutable) kolekcija elemenata sa točno određenim brojem elemenata. Nakon što se torka definira, njezini elementi ne mogu se mijenjati.

```
new_tuple = (1, 'Python', 7, 'R')
```

8. **Data Frame (Pandas):** Pandas je biblioteka koja omogućava rad s tabličnim podacima u obliku DataFrame-ova, slično radu s podacima u R-ovom Data Frame-u.

```
import pandas as pd

data = {'grad': ['Pazin', 'Zagreb', 'NN'], 'poštanski_broj': [52000, 10000, 0000]}
df = pd.DataFrame(data)
```

9. **Mješoviti Tipovi Podataka:** Slično R-u, u Pythonu možete koristiti liste za pohranu različitih tipova podataka, što omogućava kreiranje mješovitih struktura podataka.

```
popis_svega = [5, 'Python', 'R', 3.14, {'ime': 'John'}]
```

U R-u je važna funkcija c za rad s vektorima. U Pythonu možemo iste operacije obaviti na sličan način, koristeći liste U Pythonu koristimo listu za stvaranje vektora. Osnovni podaci o listi i njezinim metodama mogu se pronaći pomoću

```
help(list)
```

Prva lista: (eksplicitno definirani članovi)

```
prviv = [2, 2, 2, 2]
```

Druga lista: (range funkcija koja će napuniti listu sa brojevima u navedenom rasponu)

```
drugiv = list(range(1, 9))
```

Treća lista: (range funkcija s navedenim rasponom i korakom)

```
treciv = list(range(8, 4, -1)) -> spremamo u listu brojeve od 8 do 4, pri čemu ih umanjujemo za 1 u svakom koraku iteracije
```

Operacije s vektorima

```
zbrojv = [a + b for a, b in zip(prviv, treciv)]
incrv = [x + 1 for x in drugiv]
reciprv = [1 / x for x in prviv]
```

Postoji niz korisnih operacija za rad s listama(vektorima) u Pythonu

Funkcija seq() i vektori u Pythonu

U Pythonu možemo koristiti funkciju `numpy.linspace` za postizanje istog efekta kao funkcija `seq()` u R-u.

```
import numpy as np

# Kreiranje vektora od 0 do 9 s 6 brojeva, uključujući nulu i devetku.
niz09 = np.linspace(0, 9, num=6)

# Ispisuje vektor niz09: [0.  1.8 3.6 5.4 7.2 9. ]
```

Kreiranje vektora tipa string

```
ime = ["Iva", "Ana", "Marko"]
```

Ispisuje vektor ime: ['Iva', 'Ana', 'Marko']

Elementima vektora ime pristupa se isto kao u R-u

```
prvi_element = ime[0]  # Prvi element vektora
```

Ispisuje prvi element: 'Iva'

Pristup uzastopnim elementima vektora

```
uzastopni_elementi = ime[1:2]  # Prvi i drugi element vektora
```

Ispisuje uzastopne elemente: ['Ana'], ['Marko']

Pristup određenim elementima vektora koristeći listu indeksa

```
odabrani_elementi = [ime[2], ime[0]]  # Treći i prvi element vektora
```

Ispisuje odabrane elemente: ['Marko', 'Iva']

Uklanjanje elementa iz vektora koristeći negativni indeks

```
vektor_bez_elementa = ime[-2]  # Bez drugog elementa vektora
```

Ispisuje vektor bez elementa: ['Iva', 'Marko']

Filtriranje elemenata vektora koji ispunjavaju određeni uvjet

```
veci_od_sest = [x for x in treciv if x >= 6]  # Svi elementi vektora treciv veći ili jednaki 6
```

Ispisuje elemente veće ili jednake 6: [8, 7, 6] Pristup elementima vektora

```
treci_element = drugiv[2]  # Python indeksiranje počinje od 0, pa je treći element indeksiran s 2
```

Radi s vektorima koji nisu jednake duljine tako da se reciklira kraći vektor

```
zbrojv2 = [a + b for a, b in zip(drugiv, treciv * (len(drugiv) // len(treciv)) + treciv[:len(drugiv) % len(treciv)])]
```

Funkcija rep() u Pythonu nije potrebna jer možemo jednostavno množiti listu s brojem ponavljanja.

```
prviv2 = [2] * 4
```

Funkcija range() u Pythonu je ekvivalent funkciji seq() u R-u

```
par5 = list(range(2, 11, 2))
```

Generiranje brojeva između 0 i 1 s određenim korakom

```
niz01 = [i / 10 for i in range(11)]
```

Ispis rezultata

```
print("Prvi vektor:", prviv)
print("Drugi vektor:", drugiv)
print("Treći vektor:", treciv)
print("Zbroj vektora prviv i treciv:", zbrojv)
print("Inkrementirani drugi vektor:", incrv)
print("Recipročni prviv:", reciprv)
print("Treći element drugog vektora:", treci_element)
print("Zbroj vektora drugiv i treciv:", zbrojv2)
print("Prvi vektor pomnožen sa 4:", prviv2)
print("Paran brojevi od 2 do 10:", par5)
print("Niz brojeva od 0 do 1 s korakom 0.1:", niz01)
```

To nije naročito korisno kod malih vektora, ali je jasno da je kod velikih vektora, ili drugih struktura podataka, korisno na brz način izdvojiti elemente koji zadovoljavaju neki uvjet. Naravno, moguće je odrediti na kojim se mjestima unutar vektora nalaze elementi koji zadovoljavaju određeni uvjet i to korištenjem funkcije "numpy.where()".

```
import numpy as np

treciv = np.array([8, 7, 6, 5])
ime = ["Iva", "Ana", "Marko"]

# Elementi vektora treciv koji su veći ili jednaki od 6:
rezultat_treciv = np.where(treciv >= 6)
print(rezultat_treciv)
# Output: (array([0, 1, 2]),)

# Najmanji element vektora treciv:
min_element_treciv = np.min(treciv)
print(min_element_treciv)
# Output: 5

# Indeks najmanjeg elementa vektora treciv:
indeks_min_elementa = np.argmin(treciv)
print(indeks_min_elementa)
# Output: 3

# Elementi vektora ime koji su različiti od "Iva":
rezultat_ime = [x for x in ime if x != "Iva"]
print(rezultat_ime)
# Output: ['Ana', 'Marko']

# Indeks (na kojem se mjestu nalazi) elementa vektora ime koji sadrži "Marko":
indeks_marko = ime.index("Marko")
print(indeks_marko)
# Output: 2
```

Mali zadatak: u Pythonu se operator koji kao rezultat daje ostatak pri dijeljenju dva cijela broja označava s %. Kako bismo našli sve elemente vektora/liste koji su parni? **Parni brojevi imaju ostatak pri dijeljenju s 2 jednak nuli. Možemo provjeriti koji od elemenata iz drugiv imaju ostatak pri dijeljenju s 2 jednak nuli.**

```
drugiv = [1, 2, 3, 4, 5, 6, 7, 8]

**Pronalaženje parnih brojeva u vektoru drugiv:**
parni_brojevi = [x for x in drugiv if x % 2 == 0]
print(parni_brojevi)
Output: [2, 4, 6, 8]
```

U Pythonu, tipovi varijabli se automatski određuju, ali možemo koristiti funkcije za provjeru tipova.

Definiranje varijabli

```
cjel = 42
real = 12.65
```

Provjera tipova varijabli

```
tip_cjel = type(cjel)
tip_real = type(real)
```

Ispis tipova varijabli

```
print(tip_cjel)  # Output: <class 'int'>
print(tip_real)  # Output: <class 'float'>
```

Drugi način da se broj pohrani kao cjelobrojni u Pythonu je da se koristi funkcija int().

```
cjel2 = int(12)
```

Provjera tipa cjelobrojne varijable

```
tip_cjel2 = type(cjel2)
print(tip_cjel2)  # Output: <class 'int'>
```

U Pythonu također možete koristiti funkciju type() za provjeru tipa objekta.

Bez želje za zbunjivanjem, napomenut ćemo da Python također podržava razne metode za manipulaciju tipovima podataka, ali o tome se može govoriti detaljnije u drugom kontekstu.

Matrice u Pythonu

Koristeći arrayeve (nizove), moguće je definirati i matrice. Potrebno je uvesti biblioteku NumPy

```
import numpy as np
```

Kreiranje matrice - možemo i prije samog ispisa matrice, približno vidjeti kako će naša matrica izgledati zahvaljujući prilično intuitivnoj sintaksi np.array-a

```
A = np.array([[1, 2, 3],
              [4, 5, 6]])
```

Pristup elementima matrice

Elementima pristupamo preko njihove pozicije koju dohvaćamo pomoću indeksa reda i stupca

```
element = A[0, 2]  # Pristup elementu u prvom retku i trećem stupcu (indeksi se broje od 0)
print(element)  # Output: 3
```

Pristup cijelom retku ili stupcu

Također, možemo pristupati cijelim retcima ili stupcima tako što unosimo njihov redni broj

```
prvi_redak = A[0, :]  # Pristup cijelom prvom retku
treci_stupac = A[:, 2]  # Pristup cijelom trećem stupcu
print(prvi_redak)  # Output: [1 2 3]
print(treci_stupac)  # Output: [3 6]
```

Izdvajanje elemenata koji zadovoljavaju uvjet

Ukoliko želimo iz matrice ispisati određene elemente, možemo posatviti uvjet ispisa

```
uvjet = A >= 3
rezultat = A[uvjet]  # Izdvajanje elemenata većih ili jednakih 3
print(rezultat)  # Output: [3 4 5 6]
```

Provjera tipa podataka

```
tip = A.dtype  # Tip podataka u matrici
print(tip)  # Output: int64

import numpy as np

### Kreiranje matrice A
A = np.array([[1, 2, 3],
              [4, 5, 6]])

### Pristup elementima matrice
element = A[0, 2]  # Pristup elementu u prvom retku i trećem stupcu (počinje s 0)
print(element)  # Output: 3

### Pristup cijelom retku ili stupcu
prvi_redak = A[0, :]  # Pristup cijelom prvom retku
treci_stupac = A[:, 2]  # Pristup cijelom trećem stupcu
print(prvi_redak)  # Output: [1 2 3]
print(treci_stupac)  # Output: [3 6]
```

```
### Pristup elementima koji zadovoljavaju uvjet
uvjetni_elementi = A[A >= 3]  # Elementi veći ili jednaki 3
print(uvjetni_elementi)  # Output: [3 4 5 6]

### Operacije s matricama
rezultat_zbrajanja = A + 2  # Zbrajanje svakog elementa s brojem 2
rezultat_mnozenja = A * 2  # Množenje svakog elementa s brojem 2
print(rezultat_zbrajanja) # Output:
 [[3 4 5]
 [6 7 8]]
print(rezultat_mnozenja) # Output:
 [[ 2  4  6]
  [ 8 10 12]]

### Umnožak matrice A i transponirane matrice A
umnozak = np.dot(A, A.T)
print(umnozak) #Output:
 [[14 32]
  [32 77]]
```

Liste

U Pythonu, liste su elementi koji mogu sadržavati više podataka koji mogu biti istovrsnog, ali i različitog tipa.

```
logi = [True, True, False, False, True] # primjer liste sa Bool vrijednostima
```

Možemo stvoriti listu koja sadrži različite tipove podataka.

```
lista = [treciv, ime, logi]
```

(konkretno, u ovu smo listu ubavili razne nizove podataka koje smo već kreirali ranije u skripti)

Da bismo uklonili nazive iz vektora ime, možemo koristiti " ".

DEFINIRALI SMO DA JE LISTA IME KOJA SE NALAZI U NAŠOJ VEĆOJ LISTI S NAZIVOM "LISTA", OD SADA PRAZNA I PRAKTIČKI JU UKLONILI

```
ime = []

### Prikaz liste
print(lista)
```

Data frame-ovi u Pythonu obično se stvaraju pomoću pandas biblioteke i služe za pohranu raznih tabličnih podataka

```
import pandas as pd

# Stvaranje data frame-a
dob = [21, 25, 22]
```

```
df = pd.DataFrame({'ime': ime, 'dob': dob})

### Prikaz data frame-a
print(df)

### Dodavanje novog stupca u data frame
vis = [165, 172, 180]
df['vis'] = vis

### Prikaz proširenog data frame-a
print(df)
```

Učitavanje vanjskih datoteka

Za učitavanje tabličnih podataka, možemo koristiti Excel datoteke. Najprije moramo uključiti biblioteku os koja nam omogućava do dohvatimo podatke iz našeg operativnog sustava

```
import os
import pandas as pd
```

Promjena radnog direktorija

Pomoću naredbe os.chdir možemo promijeniti direktorij u kojem se nalazimo, kako bismo mogli dohvatiti našu točnu datoteku. (opaska: chdir dolazi od "Change directory")

```
os.chdir("C:/Sveuciliste-Pula/Statistika/Markdown/Predavanja")
```

Prikaz trenutnog radnog direktorija

```
print(os.getcwd()) (opaska: getcwd dolazi od "Get current working directory")
```

Učitavanje Excel datoteke

Kako su Excel datoteke čest način pohrane raznih podataka, praktično je nekada učitati Excel worksheet kako bismo ga obradili. Koristimo pandas library za rad s Excel datotekama

```
podaci = pd.read_excel("BMI.xlsx", sheet_name="Prva", header=0, nrows=13)
```

Prikaz strukture podataka

```
print(podaci.info())
```

Prikaz podataka

```
print(podaci)
```

Pristupanje podacima u DataFrame-u

Na primjer, podatke o visini dobijemo na sljedeći način:

```
visina = podaci["Visina"]
```

Filtriranje podataka

Primjer filtriranja osoba manje ili jednako 180 cm po visini:

```
niska_visina = podaci[podaci["Visina"] <= 180]
```

Izračun BMI

RAČUNANJE BMI-JA I DODAVANJE REZULTATA U DATAFRAME

```
podaci["BMI"] = podaci["Tezina"] / ((podaci["Visina"] / 100) ** 2)
```

Snimanje podataka u Excel datoteku "BMI2.xlsx"

```
podaci.to_excel("BMI2.xlsx", sheet_name="Druga", index=False)
```

Prije djela 2.1.2 nema programskih kodova

NAPOMENA:U PYTHONU UNUTAR LISTI kvant UNOSIMO SVOJE VLASTITE PODATKE

Pogledajmo kako bismo u Pythonu napravili iste tablice i prikazali podatke grafički. Prvo ćemo prevesti tablicu s originalno dobivenim podacima u Python dataframe, kojeg ćemo nazvati "krgrupa".

Možemo koristiti Pandas biblioteku (Python and DataScience). Najprije importamo cijelu biblioteku i zatim kreiramo dataset data.

```
import pandas as pd

data = {
    'Spol': ["Ž", "M", "M", "Ž", "Ž", "M", "M", "Ž", "M", "Ž"],
    'Krvna_grupa': ["A", "0", "B", "0", "0", "A", "AB", "0", "A", "A"]
}

krgrupa = pd.DataFrame(data)
print(krgrupa)

Pohranjujemo ga u dataframe krgrupa i ispisujemo
```

U Pythonu možemo koristiti Pandas za ostvarivanje funkcionalnosti prebrojavanja broja pojavljivanja svake vrijednosti u pojedinim varijablama.

```
import pandas as pd

data = {
    'Spol': ["Ž", "M", "M", "Ž", "Ž", "M", "M", "Ž", "M", "Ž"],
    'Krvna_grupa': ["A", "0", "B", "0", "0", "A", "AB", "0", "A", "A"]
}

krgrupa = pd.DataFrame(data)
```

Koristimo value_counts() za izračun frekvencije pojavljivanja svake vrijednosti u stupcu 'Spol'

```
spol_counts = krgrupa['Spol'].value_counts()

print(spol_counts)
```

Ovaj Python kod koristi funkciju value_counts() za izračun frekvencije pojavljivanja svake vrijednosti u stupcu 'Spol' u DataFrame-u "krgrupa". Nakon izvođenja koda, dobit ćete rezultat koji prikazuje frekvenciju pojave "M" i "Ž".

U Pythonu možemo dobiti frekvenciju pojavljivanja pojedinih krvnih grupa među danim osobama

```
import pandas as pd

data = {
    'Spol': ["Ž", "M", "M", "Ž", "Ž", "M", "M", "Ž", "M", "Ž"],
    'Krvna_grupa': ["A", "0", "B", "0", "0", "A", "AB", "0", "A", "A"]
}

krgrupa = pd.DataFrame(data)
```

Tablični pregled po krvnoj grupi

```
krvna_grupa_table = pd.crosstab(krgrupa['Krvna_grupa'], krgrupa['Spol'])

print(krvna_grupa_table)
```

```
Spol       M  Ž
Krvna_grupa
0          1  3
A          2  2
AB         1  0
B          1  0
```

Tablice relativnih frekvencija za krvne grupe dobivamo tako da frekvenciju pojedinih krvnih grupa dijelimo s ukupnim brojem dostupnih podataka. Koristit ćemo biblioteku Pandas

```
import pandas as pd

data = {
    'Krvna_grupa': ["0", "A", "AB", "B"],
}

krgrupa = pd.DataFrame(data)

# Izračun relativnih frekvencija za krvne grupe
relk = krgrupa['Krvna_grupa'].value_counts() / len(krgrupa)

print(relk)
```

Ovaj Python kod koristi funkciju value_counts() i dijeli rezultate s ukupnim brojem dostupnih podataka kako bi dobio relativne frekvencije krvnih grupa.

```
0      0.4
A      0.4
AB     0.1
B      0.1
Name: Krvna_grupa, dtype: float64
```

Operacije nad jedinstvenim elementima objekata

U Pythonu se ponovno koristi Pandas i za operacije nad jedinstvenim elementima objekta, što je posebno korisno kada radimo s tablicama; primjerice možemo ispisati tablicu vertikalno

```
import pandas as pd

data = {
    'Krvna_grupa': ["0", "A", "AB", "B"],
}

krgrupa = pd.DataFrame(data)
```

Koristimo funkciju value_counts() za dobivanje frekvencije svake krvne grupe

```
frekvencije = krgrupa['Krvna_grupa'].value_counts().reset_index()
frekvencije.columns = ['Krvna_grupa', 'freq']

print(frekvencije)
```

Ovaj Python kod koristi value_counts() da dobije frekvenciju svake krvne grupe i zatim koristi reset_index() i columns kako bi formatirao tablicu.

```
  Krvna_grupa  freq
0           0     4
1           A     4
2          AB     1
3           B     1
```

Python

Možemo vizualizirati tablicu frekvencija krvnih grupa koristeći biblioteku matplotlib koja služi za grafički prikaz podataka.

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
    'Krvna_grupa': ["0", "A", "AB", "B"],
}

krgrupa = pd.DataFrame(data)
```

Kreiranje grafikona stupčaste frekvencije

```
plt.figure(figsize=(6, 4))  # Postavljanje veličine grafikona
plt.bar(krgrupa['Krvna_grupa'].value_counts().index,
krgrupa['Krvna_grupa'].value_counts(), color='skyblue')
plt.title("Stupčasti prikaz frekvencije krvnih grupa")  # Naslov grafikona
plt.xlabel("Krvna grupa")  # X-osa
plt.ylabel("Broj osoba")  # Y-osa
plt.xticks(rotation=0)  # Rotacija oznaka na X-osi
plt.tight_layout()

plt.show()  # Prikaz grafikona
```

S vremena na vrijeme, osim tablice relativnih frekvencija, prikazivanje tablice kumulativnih relativnih frekvencija može biti relevantno. Python koristi pandas za izračun kumulativnih relativnih frekvencija i stvara DataFrame s kumulativnim frekvencijama. Nakon izvođenja koda, dobit ćemo ekvivalentnu tablicu kumulativnih relativnih frekvencija.

```
import pandas as pd

data = {
    'Krvna_grupa': ["0", "A", "AB", "B"],
}

krgrupa = pd.DataFrame(data)

## Izračun kumulativnih relativnih frekvencija
relative_frequencies =
krgrupa['Krvna_grupa'].value_counts(normalize=True).sort_index()
cumulative_frequencies = relative_frequencies.cumsum()

# Dodajemo kumulativne relativne frekvencije u novi stupac 'Cumulative_Freq'
cumulative_frequencies_df = pd.DataFrame(cumulative_frequencies, columns=
['Cumulative_Freq'])

print(cumulative_frequencies_df)
```

```
  Cumulative_Freq
0             0.4
A             0.8
```

```
AB             0.9
B              1.0
```

```
NA|   Ime  |  Težina |  Visina |   BMI   |
  |--------|---------|---------|---------|
  |   A    |    60   |   165   | 22.038572|
  |   B    |    80   |   180   | 24.69136 |
  |   C    |    85   |   175   | 27.755104|
  |   D    |    84   |   184   | 24.810965|
  |   E    |    62   |   168   | 21.967126|
  |   F    |    65   |   175   | 21.224497|
  |   G    |    86   |   184   | 25.401708|
  |   H    |    86   |   192   | 23.328999|
  |   I    |    95   |   182   | 28.68011 |
  |   J    |    78   |   178   | 24.61810 |
  |   K    |    70   |   175   | 22.85714 |
  |   L    |    68   |   170   | 23.52941 |

1   1   1    3   1   1   1   2   1
```

Naravno, jasno je da se visina 175 javlja najčešće, ali u ovom slučaju to nije toliko zanimljivo. Kada se radi s kvantitativnim podacima, imamo mnogo više informacija koje možemo koristiti prilikom analize podataka. Prije svega, to uključuje mjeru središnje tendencije i mjeru raspršenosti podataka.

Vratit ćemo se BMI podacima kako bismo definirali navedene pojmove.

U Pythonu možemo provesti analizu kvantitativnih podataka koji se nalaze u Excel datoteci nazvanoj "BMI2.xlsx" na listi "Druga". Glavna svrha je izračunati i prikazati različite informacije o tim podacima. Prvo, koristi se biblioteka Pandas za učitavanje podataka iz navedene Excel datoteke. Nakon toga, prvi stupac (Ime) se izbacuje jer nije potreban za analizu.

Nakon obrade podataka, koristi se funkcija print(kvant.dtypes) kako bi se ispisali tipovi podataka za preostale stupce, što pomaže u razumijevanju strukture podataka.

Zatim se analizira stupac "Visina". Funkcija value_counts() koristi se kako bi se izračunala i ispisala frekvencija (broj pojavljivanja) svake jedinstvene vrijednosti u tom stupcu. Ovaj korak pomaže u dobivanju informacija o tome koliko često se pojedine visine pojavljuju u analiziranim podacima.

```
import pandas as pd

# Učitavanje kvantitativnih podataka iz Excel datoteke
kvant = pd.read_excel("BMI2.xlsx", sheet_name="Druga", header=0, nrows=13)

# Izbacivanje prvog stupca (Ime) jer nije potreban
kvant = kvant.iloc[:, 1:]

# Ispis tipova preostalih podataka
print(kvant.dtypes)

# Prikaz tablice frekvencija visine
```

```
visina_counts = kvant['Visina'].value_counts()
print(visina_counts)
```

DIO 2.2 NE SADRŽI NIKAKVE PROGRAMSKE DJELOVE

2.2.1

Aritmetička sredina, mod i medijan

U Pythonu za izračun aritmetičke sredine koristimo ugrađenu funkciju mean iz NumPy-a

```
import numpy as np

x = np.array([24, 5, 9, 12, 22, 2, 89, 4, 13])
result = np.mean(x)
print(result)  # Output: 20.0

# Uklanjanje vrijednosti 89 i ponovno računanje aritmetičke sredine
x = x[x != 89]

result = np.mean(x)
print(result)  # Output: 11.375
```

Medijan

Medijan je statistička mjera središnje tendencije koja predstavlja srednju vrijednost unutar poretka podataka.

U Pythonu koristimo funkciju median() iz biblioteke NumPy za izračun medijana. Prvo, izračunavamo medijan za originalni niz podataka, a zatim izračunavamo medijan za niz x nakon što smo iz njega izbacili vrijednost 89.

```
import numpy as np

# Primjer izračunavanja medijana
# Prvo, koristimo funkciju median() iz biblioteke NumPy za izračun medijana za cijeli niz podataka
data = [24, 5, 9, 12, 22, 2, 89, 4, 13]
median_value = np.median(data)
print(median_value)  # Output: 12.0

# Ako uklonimo vrijednost 89 iz niza 'x' i ponovno izračunamo medijan, dobivamo:
x = [value for value in x if value != 89]
median_x = np.median(x)
print(median_x)  # Output: 10.5
```

Medijan je koristan za mjerenje središnje tendencije, a posebno je koristan kada imate outliers (ekstremne vrijednosti) u vašim podacima, jer neće biti toliko osjetljiv na te ekstremne vrijednosti kao aritmetička sredina. Medijan je vrijednost koja dijeli

vaše podatke na dvije jednake polovice, tako da je polovina vaših podataka manja od medijana, a polovina veća.

Mod je statistička mjera koja predstavlja vrijednost koja se najčešće pojavljuje u nizu brojeva. U Pythonu mod možemo izračunati pomoću biblioteke statistics:

```
import statistics

# Stvaranje uzorka brojeva
nuz = [1, 2, 3, 3, 4, 5, 6, 6, 7, 8, 8, 9]

# Izračunavanje moda niza brojeva
mod_values = statistics.multimode(nuz)
print(mod_values)
```

Ovaj kod koristi funkciju multimode() iz statistics biblioteke kako bi pronašao sve vrijednosti koje se pojavljuju najčešće u nizu. Rezultat će sadržavati sve takve mod vrijednosti koje se pojavljuju u nizu nuz

Postoji alternativni način, pomoću numpyja:

```
import numpy as np

# Stvaranje uzorka brojeva
nuz = np.array([1, 2, 3, 3, 4, 5, 6, 6, 7, 8, 8, 9])

# Pronalazak mod vrijednosti
unique_values, counts = np.unique(nuz, return_counts=True)
max_count = np.max(counts)
mod_values = unique_values[counts == max_count]
print(mod_values)
```

U Pythonu, također možemo korisnički definirati funkciju za mod. To možemo napraviti na slijedeći način:

```
import numpy as np

def mod(x):
    unique, counts = np.unique(x, return_counts=True)
    max_count = np.max(counts)
    mode_values = unique[counts == max_count]
    return mode_values

nuz = np.array([1, 2, 3, 3, 4, 5, 6, 6, 7, 8, 8, 9])
mod_result = mod(nuz)
print(mod_result)
```

Razmatramo pojmove središnje tendencije i raspršenosti podataka u statistici. Središnja tendencija je mjera koja opisuje centralnu ili tipičnu vrijednost u skupu podataka, dok je raspršenost mjera koja opisuje varijaciju podataka i koliko su raznolike vrijednosti.

Razmotrimo primjer s nizovima podataka:

Prvi niz podataka: 8, 9, 9, 10, 10, 10, 10, 11, 11, 12 Drugi niz podataka: 1, 2, 5, 8, 10, 10, 12, 15, 18, 19

Prvi niz podataka je prilično koncentriran oko vrijednosti 10, što ukazuje na malu raspršenost podataka. Drugi niz podataka je znatno raznolikiji, što znači da ima veću raspršenost.

Unatoč razlici u raspršenosti, i aritmetička sredina i medijan oba niza podataka su 10. To nas podsjeća da iako te mjere pružaju informacije o središnjoj tendenciji, ne govore nam ništa o raspršenosti podataka. Da bismo bolje razumjeli podatke, trebamo razmotriti i druge statističke mjere koje opisuju raspršenost, kao što su varijanca i standardna devijacija.

```
import statistics

uski = [8, 9, 9, 10, 10, 10, 10, 11, 11, 12]
siroki = [1, 2, 5, 8, 10, 10, 12, 15, 18, 19]

mean_uski = statistics.mean(uski)
mean_siroki = statistics.mean(siroki)

median_uski = statistics.median(uski)
median_siroki = statistics.median(siroki)

# Mod u Pythonu ne postoji u Numpy-ju kao u R-u, pa ćemo koristiti svoju funkciju za računanje moda.
def mod(niz):
    brojac = {}
    for value in niz:
        if value in brojac:
            brojac[value] += 1
        else:
            brojac[value] = 1

    max_frekvencija = max(brojac.values())
    moda = [value for value, frekvencija in brojac.items() if frekvencija == max_frekvencija]

    return mod

mod_uski = mod(uski)
mod_siroki = mod(siroki)

print(f"Srednja vrijednost uskog niza: {mean_uski}")
print(f"Srednja vrijednost širokog niza: {mean_siroki}")

print(f"Medijan uskog niza: {median_uski}")
print(f"Medijan širokog niza: {median_siroki}")

print(f"Mod uskog niza: {mod_uski}")
print(f"Mod širokog niza: {mod_siroki}")
```

Razmatramo različite mjere koje nam pomažu da razumijemo raznolikost i središnje tendencije podataka. Jedna od tih mjera je raspon (range), koji se definira kao razlika između maksimalne i minimalne vrijednosti u nizu podataka. Druga važna mjera je varijanca (variance), koja mjeri koliko su pojedine vrijednosti u nizu razdvojene od srednje vrijednosti. Pomoću varijance možemo bolje razumjeti raspršenost podataka.

Raspon se računa kao:

range = max(x_1, x_2, ..., x_n) - min(x_1, x_2, ..., x_n)

Za primjer, izračunat ćemo raspon i varijancu za dva različita niza podataka: "uski" i "siroki".

```
import numpy as np

uski = np.array([8, 9, 9, 10, 10, 10, 10, 11, 11, 12])
siroki = np.array([1, 2, 5, 8, 10, 10, 12, 15, 18, 19])

# Računanje raspona
range_uski = np.max(uski) - np.min(uski)
range_siroki = np.max(siroki) - np.min(siroki)

# Računanje varijance
variance_uski = np.var(uski, ddof=1)  # ddof=1 za nepristranu varijancu
variance_siroki = np.var(siroki, ddof=1)

print("Za uski niz: Raspon =", range_uski, ", Varijanca =", variance_uski)
print("Za siroki niz: Raspon =", range_siroki, ", Varijanca =", variance_siroki)
```

Standardna devijacija

Iako se standardna devijacija može izračunati i iz varijance (uzimanjem korijena), Numpy ima svoju ugrađenu funkciju `std()` koja izračunava standardnu devijaciju. Nizovi podataka o kojima je riječ u ovom odlomku imaju sljedeće standardne devijacije:

```
- std(uski) = 1.154701
- sd(siroki) = 6.218253

# Varijanca
import numpy as np

# Nizovi podataka
uski = [1, 2, 3, 4, 5]
siroki = [10, 20, 30, 40, 50]

# Izračun standardne devijacije
sd_uski = np.std(uski)
sd_siroki = np.std(siroki)

# Ispis rezultata
print(f"Standardna devijacija za uski niz: {sd_uski}")
print(f"Standardna devijacija za siroki niz: {sd_siroki}")
```

NADALJE NEMA PROGRAMSKIH KODOVA SVE DO NASLOVA 2.5

Vratimo se sada na primjer s indeksom tjelesne mase (BMI). Podaci su spremljeni u dataframe "kvant", a imena stupaca (vektora) u dataframe-u su Ime, Težina, Visina i BMI. Opremljeni novim znanjem, sada ćemo lako izračunati neke osnovne kvantitativne mjere tih podataka.

```
import numpy as np
import pandas as pd

# Podaci
data = {
    'Ime': ['Osoba1', 'Osoba2', 'Osoba3', 'Osoba4', 'Osoba5'],
    'Tezina': [70, 80, 75, 95, 60],
    'Visina': [1.75, 1.80, 1.70, 1.90, 1.65],
    'BMI': [22.86, 24.69, 25.95, 26.32, 22.04]
}

# Pretvaranje podataka u DataFrame
df = pd.DataFrame(data)

# Izračun kvantitativnih mjera
srednja_vrijednost_tezine = np.mean(df['Tezina'])
medijan_tezine = np.median(df['Tezina'])
standardna_devijacija_tezine = np.std(df['Tezina'])
raspon_tezine = np.ptp(df['Tezina'])

# Ispis rezultata
print(f"Srednja vrijednost težine: {srednja_vrijednost_tezine:.5f}")
print(f"Medijan težine: {medijan_tezine:.5f}")
print(f"Standardna devijacija težine: {standardna_devijacija_tezine:.5f}")
print(f"Raspon težine: {raspon_tezine}")
```

U Pythonu koristimo .describe() za prikaz osnovnih statistika za pojedine stupce ili cijele dataframeove.

```
import pandas as pd

# Podaci
data = {
    'Ime': ['Osoba1', 'Osoba2', 'Osoba3', 'Osoba4', 'Osoba5'],
    'Tezina': [60.00, 67.25, 79.00, 76.58, 95.00],
    'Visina': [165.0, 173.8, 176.5, 177.3, 192.0],
    'BMI': [21.22, 22.65, 24.07, 24.24, 28.6]
}

# Pretvaranje podataka u DataFrame
df = pd.DataFrame(data)
```

```
# Prikaz osnovnih statistika za stupac "Tezina"
print(df['Tezina'].describe())

# Prikaz osnovnih statistika za cijeli DataFrame "kvant"
print(df.describe())
```

Vizualizacija

Vizualizaciju podataka u Pythonu moguće je ostvariti pomoću matplotliba.

```
import matplotlib.pyplot as plt

# Podaci o težini i imenima osoba
tezina = [60, 67.25, 79, 76.58, 85.25, 95]
ime = ["Osoba1", "Osoba2", "Osoba3", "Osoba4", "Osoba5", "Osoba6"]

# Postavljanje margina za graf
plt.subplots_adjust(bottom=0.15, left=0.15)

# Priprema za prikaz podataka o težini
plt.bar(ime, tezina)
plt.ylim(0, 100)
plt.title("Podaci o težini")
plt.xlabel("Osobe")
plt.ylabel("Težina (kg)")
plt.xticks(rotation=45, ha="right", fontsize=8)
plt.yticks(fontsize=8)
plt.title("Podaci o težini", fontsize=12)

# Tekstualni opis
plt.text(0.5, -0.25, "Ovdje je prikazan bar grafikon koji prikazuje podatke o težini
različitih osoba.", transform=plt.gca().transAxes)
plt.text(0.5, -0.3, "Imena osoba su prikazana na x-osi, a težina (u kg) na y-osi.",
transform=plt.gca().transAxes)
plt.text(0.5, -0.35, "Grafikon prikazuje raspodjelu težina među osobama.",
transform=plt.gca().transAxes)

# Prikaz grafa
plt.show()
```

```
import matplotlib.pyplot as plt

# Podaci
ime = kvant['Ime']
tezina = kvant['Tezina']

# Postavljanje margina za graf
plt.subplots_adjust(left=0.2, right=0.9, top=0.9, bottom=0.2)

# Priprema za prikaz podataka o težini
plt.barh(ime, tezina, color='darkred')
```

```
# Postavljanje osi i naslova
plt.xlim(0, 100)
plt.xlabel('Težina (kg)')
plt.title('Podaci o težini')

# Prikaz grafikona
plt.show()

# Tekstualni opis
print("Ovdje je prikazan horizontalni bar grafikon koji prikazuje podatke o težini različitih osoba.")
print("Imena osoba su prikazana na y-osi, a težina (u kg) na x-osi.")
print("Grafikon prikazuje raspodjelu težina među osobama.")
```

Histogram je grafički način prikazivanja distribucije podataka. Na x-osi se nalaze različite vrijednosti visine, dok je na y-osi prikazana frekvencija pojavljivanja svake visine. Histogram omogućava vizualno prepoznavanje učestalosti različitih visina u podacima, što je korisno za analizu distribucije podataka.

```
import matplotlib.pyplot as plt

# Postavljanje margina za graf
plt.subplots_adjust(left=0.15)
plt.figure(figsize=(6, 4))

# Prikaz histograma visine
plt.hist(kvant['Visina'], bins=10, edgecolor='k', color='blue')
plt.title('Histogram visine')
plt.xlabel('Visina')
plt.ylabel('Broj pojavljivanja')

plt.show()
```

Posebno je praktično to što možemo ručno podesiti razne parametre našeg grafa, poput veličine, naziva osi, boje i sl.

```
import matplotlib.pyplot as plt
import numpy as np

# Podaci
visina = kvant['Visina']  # Ovdje ubacimo naše podatke

# Postavljanje stila grafikona
plt.style.use('ggplot')

# Postavljanje parametara grafikona
plt.figure(figsize=(6, 6))  # Veličina grafikona
plt.hist(visina, bins=len(visina), color='aquamarine', edgecolor='black')
plt.title('Visina ispitanika', fontsize=14)
plt.xlabel('Visina', fontsize=12)
plt.ylabel('Frekvencija pojavljivanja', fontsize=12)
```

```
# Prilagodba veličine oznaka na osima
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

plt.show()
```

Python koristi numpy.histogram za izračunavanje histograma i pristupa relevantnim podacima poput granica (breaks), brojača (counts), gustoće (density), srednjih točaka (mids), i drugih sličnih podataka.

```
import numpy as np
import matplotlib.pyplot as plt

# Podaci o visini
visina = kvant['Visina']  # Zamijenite ovo sa stvarnim podacima

# Izračunavanje histograma
hist, bin_edges = np.histogram(visina, bins='auto', density=False)

# Prikazivanje histograma
plt.hist(visina, bins=bin_edges, color='aquamarine', edgecolor='black')
plt.title('Visina ispitanika', fontsize=14)
plt.xlabel('Visina', fontsize=12)
plt.ylabel('Frekvencija pojavljivanja', fontsize=12)

# Pristup izračunatim podacima
print("breaks:", bin_edges)
print("counts:", hist)
print("density:", hist / len(visina))
print("mids:", (bin_edges[1:] + bin_edges[:-1]) / 2)
print("xname:", "visina")
print("equidist:", True)
print("class_attr:", "histogram")

plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt

# Podaci o visini
visina = kvant['Visina']  # Zamijenimo ovo sa stvarnim podacima

# Postavljanje stila grafikona
plt.style.use('ggplot')

# Postavljanje parametara grafikona
plt.figure(figsize=(6, 6))  # Veličina grafikona
plt.hist(
    visina,
    bins=len(visina),
    color='darkmagenta',
    edgecolor='black',
```

```
    density=True  # Omogućavanje relativne frekvencije
)
plt.title('Visina ispitanika', fontsize=14)
plt.xlabel('Visina', fontsize=12)
plt.ylabel('Relativna frekvencija', fontsize=12)

# Prilagodba veličine oznaka na osama
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

plt.show()
```

Možemo učiniti i da naši stupci imaju različite širine

```
import numpy as np
import matplotlib.pyplot as plt

# Podaci o visini
visina = kvant['Visina']  # Zamijenimo ovo sa stvarnim podacima

# Postavljanje stila grafikona
plt.style.use('ggplot')

# Postavljanje parametara grafikona
plt.figure(figsize=(6, 6))  # Veličina grafikona
bin_edges = [165, 168, 172, 178, 180, 182, 192]
plt.hist(
    visina,
    bins=bin_edges,
    color='darkmagenta',
    edgecolor='black',
    density=True  # Omogućavanje relativne frekvencije
)
plt.title('Visina ispitanika - različite širine', fontsize=14)
plt.xlabel('Visina', fontsize=12)
plt.ylabel('Relativna frekvencija', fontsize=12)
plt.ylim(0, 0.06)

# Prilagodba veličine oznaka na osima
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

plt.show()
```

Završit ćemo ovaj kratki pregled grafičkog prikaza podataka korištenjem jednog tipa grafa, takozvanog boxplota (funkcija u Pythonu je ggplot`). Boxplot se na engleskom naziva "box and whiskers diagram", a može se prevesti i kao "pravokutni dijagram". On nam je koristan u kontekstu analize grupiranja podataka, njihovog rasporeda i identifikacije ekstremnih vrijednosti (outliera).

```
import matplotlib.pyplot as plt

# Podaci o visini
```

```
visina = kvant['Visina']  # Zamijenimo ovo sa stvarnim podacima

# Postavljanje stila grafikona
plt.style.use('ggplot')

# Postavljanje parametara grafikona
plt.figure(figsize=(6, 6))  # Veličina grafikona
plt.boxplot(
    visina,
    vert=False,  # Horizontalni prikaz
    widths=0.6,  # Širina "pravokutnih" dijelova
    patch_artist=True,  # Omogućava stilizaciju "pravokutnih" dijelova
    flierprops=dict(marker='o', markersize=5, markerfacecolor='red',
markeredgecolor='red')  # Stilizacija outlier-a
)
plt.title('Boxplot visine osoba', fontsize=14)
plt.xlabel('Visina', fontsize=12)

# Prilagodba veličine oznaka na osama
plt.yticks([])  # Uklanjanje oznaka s y-osi

plt.show()
```

Možemo prikazati i horizontalni boxplot

```
import matplotlib.pyplot as plt

# Podaci o indeksu tjelesne mase (BMI)
bmi = kvant['BMI']  # Zamijenite ovo sa stvarnim podacima

# Postavljanje stila grafikona
plt.style.use('ggplot')

# Postavljanje parametara grafikona
plt.figure(figsize=(6, 3))  # Veličina grafikona (prilagodite prema potrebi)
plt.boxplot(
    bmi,
    vert=False,  # Horizontalni prikaz
    widths=0.6,  # Širina "pravokutnih" dijelova
    patch_artist=True,  # Omogućava stilizaciju "pravokutnih" dijelova
)
plt.title('Boxplot indeksa tjelesne mase', fontsize=14)
plt.xlabel('BMI', fontsize=12)

# Prilagodba veličine oznaka na osima
plt.yticks([])  # Uklanjanje oznaka s y-osi

plt.show()
```

Na jednom grafu možemo pokazati podatke za sve 3 liste

```
import matplotlib.pyplot as plt
```

```
# Podaci o BMI
bmi_data = kvant[:, [1, 2, 3]]  # Zamijenite ovo sa stvarnim podacima

# Postavite odgovarajuće margine za grafikon
plt.figure(figsize=(6, 3))  # Veličina grafikona
plt.style.use('ggplot')

# Nacrtajte boxplot za odabrane kolone
plt.boxplot(bmi_data, vert=False, widths=0.6, patch_artist=True)
plt.title('Boxplot svih BMI podataka', fontsize=14)
plt.xlabel('BMI', fontsize=12)
plt.yticks([])  # Uklonite oznake s y-ose

plt.show()
```

Možemo, radi veće preglednosti prikazati i sva 3 boxplota

```
import matplotlib.pyplot as plt

# Podaci za boxplotove
visina_data = kvant['Visina']  # Zamijenite ovo sa stvarnim podacima
tezina_data = kvant['Tezina']  # Zamijenite ovo sa stvarnim podacima
bmi_data = kvant['BMI']  # Zamijenite ovo sa stvarnim podacima

# Postavimo raspored za prikaz boxplotova
plt.figure(figsize=(12, 4))  # Veličina grafikona
plt.style.use('ggplot')

plt.subplot(131)
plt.boxplot(visina_data, vert=False, widths=0.6, patch_artist=True)
plt.title('Visina')

plt.subplot(132)
plt.boxplot(tezina_data, vert=False, widths=0.6, patch_artist=True)
plt.title('Težina')

plt.subplot(133)
plt.boxplot(bmi_data, vert=False, widths=0.6, patch_artist=True)
plt.title('BMI')

plt.tight_layout()  # Organizira panel za prikaz(layout grafova)
plt.show()
```

###Koeficijent korelacije Za izračun koeficijenta korelacije, numpy ima ugrađenu funkciju corrcoef

```
import numpy as np

# Primjer korištenja funkcije corrcoef() za izračunavanje koeficijenta korelacije
xv = np.array([-2, 0, 2, 4, 4])
yv = np.array([2, 2, 3, 4, 4])

correlation_coefficient = np.corrcoef(xv, yv)[0, 1]
```

```
print("Rezultat:", correlation_coefficient)


# Učitavanje podataka za težinu, visinu i indeks tjelesne mase
tezina = np.array([60, 80, 85, 84, 62, 65, 86, 86, 95, 78, 70, 68])
visina = np.array([165, 180, 175, 184, 168, 175, 184, 192, 182, 178, 175, 170])
bmi = np.array([22.03857, 24.69136, 27.75510, 24.81096, 21.96712, 21.22449, 25.40170,
23.32899, 28.68011, 24.61810, 22.85714, 23.52941])

# Izračunajte korelaciju između visine i BMI
correlation_height_bmi = np.corrcoef(visina, bmi)[0, 1]
print("Korelacija između visine i BMI:", correlation_height_bmi)

# Izračunajte korelaciju između visine i težine
correlation_height_weight = np.corrcoef(visina, tezina)[0, 1]
print("Korelacija između visine i težine:", correlation_height_weight)
```

Scatterplot

```
import matplotlib.pyplot as plt

# Prikaz zavisnosti između visine i BMI korištenjem scatterplot-a
plt.figure(figsize=(6, 4))  # Veličina grafikona
plt.style.use('ggplot')

plt.scatter(kvant['Visina'], kvant['BMI'], marker='o', s=40)
plt.title("Visina i BMI")
plt.xlabel("Visina")
plt.ylabel("BMI")

plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Možemo iskoristiti pairplot za prikaz grafova zavisnosti između kvantitativnih
nizova. Njega uvozimo iz posebne seaborn biblioteke.
# Na primjer, za prikaz zavisnosti visine, težine i BMI

data = kvant[['Visina', 'Tezina', 'BMI']]
sns.set(style="ticks")
sns.pairplot(data, markers='o', plot_kws={'s': 40})
plt.suptitle("Zavisnost između visine, težine i BMI", y=1.02)

plt.show()
```

Da bismo u Pythonu učitali Excel datoteku, koristimo Pandas paket

```
import pandas as pd

# Učitavanje podataka iz Excel datoteke "Skok-u-dalj.xlsx" u DataFrame "dalj"
```

```
dalj = pd.read_excel("Skok-u-dalj.xlsx", sheet_name="Skok-u-dalj", header=None)

# Možemo i pridružiti imena stupcima
dalj.columns = ["daljina", "ime", "prezime", "rodjenje", "klub", "dat_skok"]
```

U ovom kontekstu, imamo jedan numerički vektor koji sadrži ukupno 30 podataka. Stupci "rodjenje" i "dat_skok" predstavljaju datume, što znači da se mogu urediti i sortirati. Na primjer, možemo sortirati stupac "rodjenje" kako bismo dobili podatke u određenom redoslijedu, kao što je prikazano u sljedećem Python kodu:

```
# Sortiranje stupca "rodjenje" u DataFrame-u "dalj"
sorted_dalj = dalj.sort_values(by="rodjenje")

import pandas as pd

# Prikaz sažetka podataka za DataFrame "dalj"
summary_dalj = dalj.describe(include="all")

# Ispis tablice srednjim formatiranjem
print(summary_dalj.to_markdown())
```

Ovaj Python kod koristi funkciju describe() za prikaz sažetka podataka u DataFrame dalj, a zatim koristi to_markdown() metodu kako bi ispisao tablicu sa srednjim formatiranjem.

Nadalje, dobivene podatke, možemo prikazati boxplotom.

```
import matplotlib.pyplot as plt

# Postavljanje margina za grafikon
plt.figure(figsize=(5, 5))
plt.margins(0.2)

# Nacrtamo boxplot za stupac "daljina" u DataFrame "dalj"
plt.boxplot(dalj["daljina"], vert=False)
plt.title("Boxplot preskočene daljine")

plt.show()
```

Možemo nacrtati histogram za stupac "daljina u našem DataFrame-u"

```
import matplotlib.pyplot as plt

# Postavljanje margina za grafikon
plt.figure(figsize=(4, 4))
plt.subplots_adjust(left=0.2, right=0.9, top=0.9, bottom=0.2)

# Nacrtajte histogram za stupac "daljina" u DataFrame "dalj"
plt.hist(dalj["daljina"], bins=10, color="darkmagenta")
plt.title("Histogram duljine skoka")
plt.xlabel("Daljina")
plt.ylabel("Frekvencija")
```

```
plt.show()
```

```
import pandas as pd

# Grupiranje duljine skoka prema klubovima i prikaz sažetka za svaku grupu
grouped = dalj.groupby("klub")
summary = grouped["daljina"].describe()
print(summary)
```

Možemo i izdovjiti natjecatelje iz pojedinih gradova i nacrtati boxplot za svaku skupinu

```
import matplotlib.pyplot as plt

# Izdvajamo natjecatelje iz Zagreba
dalj_zg = dalj[dalj["klub"] == "Zagreb"]

# Izdvajamo natjecatelje iz Rijeke
dalj_ri = dalj[dalj["klub"] == "Rijeka"]

# Izdvajamo natjecatelje iz Varaždina
dalj_vz = dalj[dalj["klub"] == "Varaždin"]

# Postavljanje margina za grafikone
plt.subplots(figsize=(12, 4))
plt.subplots_adjust(wspace=0.4)

# Crtamo boxplot za natjecatelje iz Zagreba
plt.subplot(131)
plt.boxplot(dalj_zg["daljina"], vert=False)
plt.title("Zagreb")
plt.xlabel("Daljina")

# Crtamo boxplot za natjecatelje iz Rijeke
plt.subplot(132)
plt.boxplot(dalj_ri["daljina"], vert=False)
plt.title("Rijeka")
plt.xlabel("Daljina")

# Crtamo boxplot za natjecatelje iz Varaždina
plt.subplot(133)
plt.boxplot(dalj_vz["daljina"], vert=False)
plt.title("Varaždin")
plt.xlabel("Daljina")

plt.show()
```

Isto tako, možemo prikazati i broj natjecatelja iz pojedinog grada

```
import Numpy
# Broj natjecatelja iz Zagreba, Rijeke i Varaždina
broj_natjecatelja_zg = dalj_zg.shape[0]
```

```
broj_natjecatelja_ri = dalj_ri.shape[0]
broj_natjecatelja_vz = dalj_vz.shape[0]

print(broj_natjecatelja_zg)
print(broj_natjecatelja_ri)
print(broj_natjecatelja_vz)
```

Možemo, zbog lakše manipulacije datumima, u naš dataframe dodati još 1 stupac koji ima samo numeričku godinu rođenja.

```
import pandas as pd

# Pretvorite stupac "rodjenje" u tip datuma (date)
dalj['rodjenje'] = pd.to_datetime(dalj['rodjenje'])

# Dodajte novi stupac "godina" s godinama rođenja
dalj['godina'] = dalj['rodjenje'].dt.year
```

Možemo i grupirati duljinu skoka po godinama rođenja i prikazati sažetak za svaku pojedinu grupu

```
import pandas as pd

# Grupiranje duljine skoka po godinama rođenja i prikaz sažetka za svaku grupu
grouped = dalj.groupby('godina')['daljina']
summary = grouped.describe()

print(summary)
```

Naravno, te je podatke moguće vizualizirati

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))

dalj_2004 = dalj[dalj['godina'] == 2004]
dalj_2005 = dalj[dalj['godina'] == 2005]

plt.subplot(1, 2, 1)
plt.hist(dalj_2004['daljina'], bins=10, color='skyblue')
plt.title('Godište 2004')
plt.xlabel('Daljina')
plt.ylabel('Frekvencija')

plt.subplot(1, 2, 2)
plt.hist(dalj_2005['daljina'], bins=10, color='salmon')
plt.title('Godište 2005')
plt.xlabel('Daljina')
plt.ylabel('Frekvencija')

plt.tight_layout()
plt.show()
```

U Pythonu, umjesto Tidyverse-a iz R-a, možemo koristiti Pandas, popularnu biblioteku za analizu i obradu podataka. Za čitanje podataka i izračunavanje sažetka za stupac "daljina" iz data frame-a možete koristiti sljedeći ekvivalentni Python kôd: import pandas as pd (ukoliko nije instaliran, instaliramo ga u našem terminalu pomoću pip install Pandas)

```
# Pretpostavljamo da postoji DataFrame "dalj" s odgovarajućim podacima

# Izračunajte sažetak za stupac "daljina"
summary = dalj["daljina"].describe()

# Ispis rezultata
print(summary)
```

Pandas nudi describe() funkciju koja daje sažetak za numeričke stupce u DataFrame-u, uključujući minimum, maksimum, srednju vrijednost, kvartile itd. Ovaj kôd će vam dati ekvivalentne rezultate kao i R kôd s Tidyverse-om

Kod Pythona čitanje i parsiranje HTML web stranica te dohvaćanje tablica može se postići korištenjem biblioteke BeautifulSoup i requests za dohvaćanje HTML sadržaja sa web stranice.

```
import requests
from bs4 import BeautifulSoup

# Adresa web stranice
wikistr = "https://en.wikipedia.org/wiki/"
listurban = "List_of_urban_areas_in_the_European_Union"
url = wikistr + listurban

# Dohvaćanje HTML sa web stranice
response = requests.get(url)
html_content = response.text

# Parsiranje HTML-a koristeći BeautifulSoup
soup = BeautifulSoup(html_content, 'html.parser')

# Dohvaćanje tablica
tables = soup.find_all('table', {'class': 'wikitable'})

# Pretvaranje tablica u DataFrame (ovdje pretpostavljamo da koristimo pandas)
import pandas as pd
dfs = [pd.read_html(str(table)) for table in tables]
```

```
# Iz naše html stranice možemo odabrati određene sadržaje i pohraniti ih u novi
DataFrame
import pandas as pd

# Odabir određenih stupaca iz DataFrame-a 'eu' i pretvaranje u novi DataFrame
eu_cities = eu.iloc[:, [1, 3, 4, 8]]

# Ispis strukture novog DataFrame-a
print(eu_cities.info())
```

```
# Možemo zamjeniti imena stupaca u eu_cities DataFrame-u
eu_cities.columns = ["Grad", "Drzava", "BrojSt", "Gustoca"]

# Nadalje, možemo konvertirati BrojSt i Gustoca u numeričke vektore
eu_cities["BrojSt"] = eu_cities["BrojSt"].str.replace(",", "").astype(float)
eu_cities["Gustoca"] = eu_cities["Gustoca"].str.replace(",", "").astype(float)

# Ispis strukture DataFrame-a s ograničenim brojem redova za prikaz
print(eu_cities.head(3))

# Ispis strukture eu_cities DataFrame-a
eu_cities.info()

# Možemo vidjeti gdje se Zagreb nalazi u listi gradova
zagreb_info = eu_cities[eu_cities["Grad"] == "Zagreb"]
print(zagreb_info)
```

```
# Sortiranje eu_cities po Gustoca u opadajućem redoslijedu
eu_sorted_gust = eu_cities.sort_values(by='Gustoca', ascending=False)

# Ispis prvih 10 gradova s najvećom gustoćom naseljenosti
print(eu_sorted_gust.head(10))
```

```
        Grad    Drzava   BrojSt Gustoca
67      Genoa     Italy   540000  695021
63     Vienna   Austria  1890000   56148
6     Athens    Greece  3450000  529046
57     Bilbao     Spain   775000  525051
62    Málaga     Spain   686000  509469
61 Santa Cruz     Spain   506000  465238
45 Sofia Pernik Bulgaria  947000  45713
48    Madrid     Spain  6211000  455122
22  Bucharest   Romania 1862000  45225
44 Barcelona     Spain 4800000  4477
```

Možemo sortirati gradove prema gustoći naseljenosti u opadajućem redoslijedu

```
import pandas as pd

# Sortiranje eu_cities prema gustoći naseljenosti u opadajućem redoslijedu
eu_sort_gust = eu_cities.sort_values(by='Gustoca', ascending=False)

# Ispis prvih 10 gradova s najvećom gustoćom naseljenosti
print(eu_sort_gust.head(10))
```

```
# Koristit ćemo matplotlib za kreiranje histograma s našim podacima
import matplotlib.pyplot as plt
import numpy as np

# Podaci za histogram (proučavanje BrojSt, podijeljeno s 1.000.000)
data = eu_cities['BrojSt'] / 1000000
```

```
# Postavljanje margina za grafikon
plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)

# Nacrtaj histogram
plt.hist(data, bins=11, range=(0.5, 11.5), color='aquamarine4')

# Postavke osi i naslova
plt.xlabel("Veličina urbanog područja (mil.)")
plt.ylabel("Broj gradova")
plt.title("Veličina EU gradova")

# Postavljanje oznaka na x-osi
plt.xticks(np.arange(0.5, 12, 1))

# Prikaži grafikon
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

# Postavljanje margina za grafikon
plt.subplots_adjust(left=0.15, right=0.95, top=0.9, bottom=0.2)

# Niz podataka (prilagođeni brojevi)
data = np.array(eu_cities['BrojSt']) / 1000000

# Nacrtaj histogram relativnih frekvencija
plt.hist(data, bins=11, density=True, color='darkmagenta')

# Postavljanje naslova i oznaka osi
plt.title("Veličina EU gradova")
plt.xlabel("Veličina urbanog područja (mil.)")
plt.ylabel("Relativna frekvencija")

# Postavljanje granica x-osi
plt.xlim(0.5, 11.5)

# Dodavanje oznaka na x-osi
plt.xticks(np.arange(0.5, 12, 1), rotation=0)

# Dodavanje "rug" (ticks) na x-osi
plt.scatter(data, np.zeros_like(data), marker='|', color='black', s=30)

# Prikaz grafa
plt.show()
```

Ovaj kôd crta histogram relativnih frekvencija na temelju podataka iz stupca eu_cities[broj_st], pri čemu se koristi density= TRUE. Oznake i stilovi grafikona također su postavljeni, a na x-osi se dodaju oznake s funkcijom xtics. Također je dodan "rug" na x-osi pomoću funkcije scatter.

Boxplot je koristan za prikazivanje raspodjele podataka i identificiranje potencijalnih outlier-a. U ovom slučaju, prikazat ćemo boxplot za veličinu EU gradova

```
import matplotlib.pyplot as plt

# Postavljanje margina za grafikon
plt.subplots_adjust(left=0.2)

# Crtanje horizontalnog boxplot-a za veličinu EU gradova
plt.boxplot(eu_cities["BrojSt"] / 1000000, vert=False)

# Postavljanje oznaka
plt.title("Veličina EU gradova")
plt.xlabel("Veličina urbanog područja (mil.)")
plt.ylabel("Broj gradova")

# Prikaz grafikona
plt.show()
```

Python

Python koristi pandas za ostvarivanje funkcionalnosti selektiranja podataka iz nekog dataframe-a. Python kod će grupirati podatke prema stupcu 'Drzava' i primijeniti funkciju sum na stupcu 'BrojSt' kako bi izračunao ukupnu sumu za svaku državu. Rezultat će biti DataFrame s dvjema kolonama, 'Drzava' i 'BrojSt', koji sadrži agregirane vrijednosti za svaku državu.

```
import pandas as pd

# Pretvaranje DataFrame-a eu_cities u Pandas DataFrame
eu_cities_df = pd.DataFrame(eu_cities)

# Agregacija podataka prema stupcu 'BrojSt' grupiranom po stupcu 'Drzava' i
primjenjujući funkciju sum
result = eu_cities_df.groupby('Drzava')['BrojSt'].sum().reset_index()

# Ispis rezultata
print(result)
```

U Pythonu možemo prilično jednostavno grupirati i sortirati podatke

```
import pandas as pd

# Grupiranje i agregacija
df1 = eu_cities.loc[1:20].groupby('Drzava')['BrojSt'].sum().reset_index()
df2 = eu_cities.loc[1:20].groupby('Drzava')['BrojSt'].count().reset_index()

# Spajanje data frame-ova
result_df = pd.merge(df1, df2, on='Drzava')

# Preimenovanje stupaca
result_df.columns = ['Drzava', 'Stanovnici', 'Broj_gradova']
```

```
# Sortiranje prema Broj_gradova u opadajućem redoslijedu
result_df = result_df.sort_values(by='Broj_gradova', ascending=False)

print(result_df)
```

Python

U Pythonu, postoje 2 načina da dohvatimo prvih 20 gradova po naseljenosti

```
import pandas as pd

# Učitavanje podataka iz DataFrame-a eu_cities
eu_cities = pd.DataFrame(eu_cities)  # Ovdje dodamo prave podatke

# Filtriranje prvih 20 redova
eu_cities = eu_cities.iloc[:20]

# Korištenje operatera za obradu podataka
result = (
    eu_cities
    .groupby('Drzava')
    .agg(Broj_gradova=pd.NamedAgg(column='BrojSt', aggfunc='count'),
         Stanovnici=pd.NamedAgg(column='BrojSt', aggfunc='sum'))
    .reset_index()
    .sort_values(by='Broj_gradova', ascending=False)
)

# Ispis rezultata
print(result)
```

```
import pandas as pd

# Učitavanje podataka iz DataFrame-a eu_cities
eu_cities = pd.DataFrame(eu_cities)

# Filtriranje prvih 20 redova
eu_cities = eu_cities.iloc[:20]

# Korištenje data.table za obradu podataka
import data_table as dt
dtdf = dt.Frame(eu_cities)
result = (
    dtdf[:, dt.sum(dt.f.BrojSt), dt.count(), dt.by(dt.f.Drzava)]
    .sort(-dt.f.Broj_gradova)
)

# Ispis rezultata
print(result)
```

U ovome dijelu skripte namjerno ostavljam i R i Python kod kako bi kontekst bio jasniji

R

Program "R" dolazi s raznovrsnim paketima koji su vrlo korisni za različite potrebe u analizi podataka. Popis dostupnih paketa može se pronaći korištenjem funkcije data(package = "ime_paketa"), koja prikazuje informacije o paketima i omogućuje pristup dokumentaciji i funkcionalnostima svakog od njih. Tako se, na primjer, u R-ovom paketu "datasets" nalazi skup podataka "faithful", koji sadrži podatke o erupcijama gejzira "Old Faithful" koji se nalazi u američkom Nacionalnom parku Yellowstone. Taj gejzir je poznat po relativno pravilnim erupcijama, gdje se događaju svakih 40 minuta do dva sata. U skupu podataka "faithful" nalaze se dvije kolone s 272 redaka. U prvoj koloni su navedeni vremena trajanja erupcija, a u drugoj koloni vremena čekanja između erupcija (u minutama). S obzirom na broj redaka, nećemo ispisati sve elemente tog skupa podataka, već ćemo odmah pogledati njihovu strukturu, sažetak podataka i boxplotove.

```r
library("datasets")

# Ispis strukture podataka 'faithful'
str(faithful)
# 'data.frame': 272 obs. of 2 variables:
#  $ eruptions: num 3.6 1.8 3.33 2.28 4.53 ...
#  $ waiting  : num 79 54 74 62 85 55 88 85 51 85 ...

# Sažetak podataka
summary(faithful)
#   eruptions        waiting
# Min.   :1.600   Min.   :43.0
# 1st Qu.:2.163   1st Qu.:58.0
# Median :4.000   Median :76.0
# Mean   :3.488   Mean   :70.9
# 3rd Qu.:4.454   3rd Qu.:82.0
# Max.   :5.100   Max.   :96.0

# Grafovi boxplot
par(mfrow = c(1, 2), mar = c(2, 2, 2, 2))
boxplot(faithful$eruptions, main = "Trajanje erupcije")
boxplot(faithful$waiting, main = "Vrijeme između")
```

U Pythonu ne postoji ekvivalentna funkcija koja prikazuje popis dostupnih paketa zajedno s informacijama o njima na način sličan funkciji data(package = "ime_paketa") u R-u.

U Pythonu, možete dobiti popis instaliranih paketa koristeći naredbu pip list ili pip freeze u naredbenom retku, ovisno o verziji Pythona koju koristite. Međutim, to će samo izlistati nazive paketa bez dodatnih informacija.

Da biste dobili više informacija o određenom paketu, možete koristiti pip show ime_paketa, gdje zamjenjujete "ime_paketa" stvarnim nazivom paketa. Ova naredba će prikazati informacije o verziji, autorima, opisu i drugim metapodacima za taj paket.

U Pythonu također postoji funkcija help(), koja vam omogućuje da dobijete dokumentaciju i informacije o modulima i funkcijama. Na primjer, ako želite dobiti informacije o paketu numpy, možete koristiti help(numpy). Ova funkcija će prikazati dokumentaciju za paket, modul ili funkciju koje specificirate kao argument.

R

Ovaj kod stvara dva jednodimenzionalna grafa (stripchart) za prikazivanje podataka iz skupa podataka "faithful". Prvi graf prikazuje trajanje erupcija, dok drugi graf prikazuje vrijeme između erupcija. Opcija jitter = TRUE dodaje malo nasumičnog pomaka podacima radi bolje preglednosti kad postoje višestruki zapisi s istim vrijednostima. Funkcija main koristi se za postavljanje naslova na grafove.

```
par(mfrow = c(2, 1), mar = c(2, 2, 2, 2))
stripchart(faithful$eruptions, jitter = TRUE, main = "Stripchart trajanja erupcija")
stripchart(faithful$waiting, jitter = TRUE, main = "Stripchart vremena između
erupcija")
```

Kako Python nema sadržan paket za gejzir, možemo uvesti dataframe sa tim podacima

```
import matplotlib.pyplot as plt
import pandas as pd

# Pretvaranje skupa podataka 'faithful' u DataFrame (pod pretpostavkom da je skup
podataka već dostupan)
faithful_df = pd.DataFrame({'eruptions': faithful.eruptions, 'waiting':
faithful.waiting})

# Postavljanje grafova u dva reda
plt.figure(figsize=(6, 8))

# Stripchart za trajanje erupcija
plt.subplot(2, 1, 1)
plt.stripplot(data=faithful_df, x='eruptions', jitter=True)
plt.title("Stripchart trajanja erupcija")

# Stripchart za vrijeme između erupcija
plt.subplot(2, 1, 2)
plt.stripplot(data=faithful_df, x='waiting', jitter=True)
plt.title("Stripchart vremena između erupcija")

plt.tight_layout()
plt.show()
```

```
# Postavljanje margina za grafikon
par(mar = c(4, 4, 2, 2))

# Izrada scatter-plot grafa
plot(
  faithful$waiting, faithful$eruptions,
  pch = 16, cex = 0.8,
  xlab = "Vremena između erupcija (min)",
```

```
  ylab = "Vrijeme erupcije (min)",
  main = "Scatter-plot grafikon za gejzir 'Old Faithful'"
)
```

```
import matplotlib.pyplot as plt

# Postavljanje margina za grafikon
plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)

# Izrada scatter-plot grafa
plt.scatter(faithful['waiting'], faithful['eruptions'], marker='o', s=40)
plt.xlabel("Vremena između erupcija (min)")
plt.ylabel("Vrijeme erupcije (min)")
plt.title("Scatter-plot grafikon za gejzir 'Old Faithful'")

# Prikaži grafikon
plt.show()
```

```
Poželjno je i sortirati podatke po veličini kako bi prikaz bio jasniji
# Postavljanje margina za grafikon
par(mar = c(2, 4, 1, 1))

# Ispis vremena erupcija u sortiranom redoslijedu
plot(
  sort(faithful$eruptions),
  pch = 1,
  cex = 0.7,
  cex.axis = 0.8,
  ylab = "Vrijeme erupcije (min)",
  cex.lab = 0.8
)
Možemo dodati i rug
# Postavljanje margina za grafikon
par(mar = c(2, 4, 1, 1))

# Ispis vremena erupcija u sortiranom redoslijedu
plot(
  sort(faithful$eruptions),
  pch = 1,
  cex = 0.7,
  cex.axis = 0.8,
  ylab = "Vrijeme erupcije (min)",
  cex.lab = 0.8
)

# Dodavanje "rug" (ticks) na y-osi
rug(faithful$eruptions, side = 2)

# Postavljanje margina za grafikon
par(mar = c(2, 4, 1, 1))

# Ispis vremena između erupcija u sortiranom redoslijedu
```

```r
plot(
  sort(faithful$waiting),
  pch = 1,
  cex = 0.7,
  cex.axis = 0.8,
  ylab = "Vrijeme između erupcija (min)",
  cex.lab = 0.8
)

# Dodavanje "rug" (ticks) na y-osi
rug(faithful$waiting, side = 2)

# Postavljanje margina za grafikon
par(mar = c(4, 4, 2, 2))

# Nacrtaj histogram vremena između erupcija
hist(
  faithful$eruptions,
  xlab = "Vrijeme između erupcija (min)",
  main = "Histogram frekvencija vremena između erupcija",
  cex.axis = 0.7,
  cex.main = 0.8,
  cex.lab = 0.8,
  ylab = "Frekvencija",
  col = "aquamarine4"
)
```

```python
import matplotlib.pyplot as plt
import numpy as np

# Postavljanje margina za grafikone
plt.figure(figsize=(8, 10))
plt.subplots_adjust(hspace=0.5)

# Ispis vremena erupcija u sortiranom redoslijedu
plt.subplot(3, 1, 1)
sorted_eruptions = np.sort(faithful['eruptions'])
plt.plot(sorted_eruptions, 'o', markersize=5)
plt.title('Vrijeme erupcija (min)')
plt.xlabel('Redoslijed')
plt.ylabel('Vrijeme erupcije (min)')

# Dodavanje "rug" (ticks) na y-osi
plt.subplot(3, 1, 2)
plt.plot(sorted_eruptions, np.zeros_like(sorted_eruptions), '|', markersize=20)
plt.title('Vrijeme erupcija s "rug" (ticks)')
plt.xlabel('Vrijeme erupcije (min)')

# Ispis vremena između erupcija u sortiranom redoslijedu
plt.subplot(3, 1, 3)
sorted_waiting = np.sort(faithful['waiting'])
plt.plot(sorted_waiting, 'o', markersize=5)
```

```
plt.title('Vrijeme između erupcija (min)')
plt.xlabel('Redoslijed')
plt.ylabel('Vrijeme između erupcija (min)')

plt.show()

# Nacrtaj histogram vremena između erupcija
plt.figure(figsize=(8, 5))
plt.hist(sorted_eruptions, bins=15, color='aquamarine', edgecolor='black')
plt.title('Histogram frekvencija vremena između erupcija')
plt.xlabel('Vrijeme između erupcija (min)')
plt.ylabel('Frekvencija')

plt.show()
```

```
# Postavljanje margina za grafikon
par(mar = c(5, 4, 2, 2))

# Nacrtaj histogram vremena između erupcija
hist(
  faithful$eruptions,
  main = "Histogram relativnih frekvencija",
  breaks = 20,
  probability = TRUE,
  cex.axis = 0.7,
  cex.main = 0.8,
  xlab = "Vrijeme između erupcija (min)",
  cex.lab = 0.8,
  ylab = "Frekvencija",
  col = "aquamarine4"
)

# Dodaj krivulju glatke gustoće
points(
  density(faithful$eruptions, bw = 0.1),
  type = 'l',
  col = 'red',
  lwd = 2
)
```

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import gaussian_kde

# Postavljanje margina za grafikon
plt.figure(figsize=(8, 6))
plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)

# Nacrtaj histogram vremena između erupcija
plt.hist(
    faithful['eruptions'],
    bins=20,
```

```
    density=True,
    color='aquamarine4',
    edgecolor='black'
)
plt.title('Histogram relativnih frekvencija')
plt.xlabel('Vrijeme između erupcija (min)')
plt.ylabel('Frekvencija')

# Izračunaj i nacrtaj glatku krivulju gustoće
density = gaussian_kde(faithful['eruptions'])
x = np.linspace(faithful['eruptions'].min(), faithful['eruptions'].max(), 1000)
plt.plot(x, density(x), 'r', linewidth=2)

plt.show()
```

Završit ćemo ovu malu analizu podataka o gejziru "Old Faithful" analizom korelacije između vremena trajanja erupcija i vremena između njih. Prikazani dijagram raspršenja sugerira da postoji visoka pozitivna korelacija između tih podataka. To isto potvrđuje i izračun korelacije:

```
cor(faithful$waiting, faithful$eruptions)
# [1] 0.9008112
```

Python ekvivalent:

```
import pandas as pd
import numpy as np

# Učitaj podatke
data = pd.read_csv("putanja/do/podataka.csv")  # Zamijenite "putanja/do/podataka.csv"
sa stvarnom putanjom do datoteke

# Izračunaj korelaciju
correlation = np.corrcoef(data['waiting'], data['eruptions'])[0, 1]
print(f"Korelacija: {correlation}")
```

Unutar treće cjeline nema programskih odsječaka sve do lekcije 3.3.16 (Pravila umnoška i zbroja se prirodno pojavljuju u programiranju). Pogledajmo sljedeća dva koda (u Python-u) u kojima se izvršava m for petlji s time da su u lijevo navedenom kodu one ugniježdene, dok su u desno navedenom kodu navedene jedna za drugom.

Prva petlja (ugnježdene)

```
k = 0
for i1 in range(1, n1 + 1):
    for i2 in range(1, n2 + 1):
        # ...
        for im in range(1, nm + 1):
            k += 1

##### Druga petlja (1 ispod druge)
k = 0
for i1 in range(1, n1 + 1):
```

```
    k += 1
for i2 in range(1, n2 + 1):
    k += 1
# ...
for im in range(1, nm + 1):
    k += 1
```

Vidimo da se svaka od for petlji izvršava ni puta, i {1, 2, . . . , m}, pri čemu se desne petlje izvršavaju neovisno jedna o drugoj. Stoga će vrijednost od k nakon izvršenja svih petlji biti jednaka n1 + n2 + . . . + nm, to je u skladu s pravilom zbroja. S druge strane, kod ugniježdenih petlji će se svaka unutrašnja petlja izvršiti onoliko puta koliko joj petlji prethodi pomnoženo s pripadnim ni, odnosno vrijednost od k nakon izvršenja svih petlji će biti jednaka n1 · n2 · . . . · nm, to je u skladu s pravilom umnoška.

Nadalje, programski se kodovi pojavljuju u lekciji 3.5. Kako su u kombinatorici veoma važni izračuni faktorijela i binomnih koeficijenata, logično je očekivati da u Pythonu postoji ugrađeni način da se izračunaju. Da bismo u Pythonu dobili mogućnost za računanje faktorijela i binomnog koeficijenta, možemo koristiti biblioteku math. Ovdje je Python kod za oba izračuna:

```
import math

# Računanje faktorijela broja 15
fact_15 = math.factorial(15)
print(fact_15)

# Izračun binomnog koeficijenta "10 nad 4"
binomial_coef = math.comb(10, 4)
print(binomial_coef)
```

Naravno, možemo izračunati razne vjerojatnosti i jesmo li izračunali ispravno možemo provjeriti pomoću generiranja slučajnih uzoraka. Ako želimo generirati uređeni uzorak od 5 brojeva od 10 dostupnih brojeva bez ponavljanja, koristimo sljedeći kod:

```
import random

# Generiraj uzorak od 5 brojeva od 1 do 10 bez ponavljanja
uzorak = random.sample(range(1, 11), 5)
print(uzorak)
```

Ovdje smo generirali uzorak 5 brojeva od 1 do 10. Sada možemo koristiti funkciju set.seed() za postavljanje početnog stanja generatora slučajnih brojeva kako bismo rezultate simulacije reproducirali na različite načine.

```
import random

# Postavi sjeme generatora slučajnih brojeva na 1
random.seed(1)

# Generiraj uzorak od 5 brojeva od 1 do 10 bez ponavljanja
uzorak1 = random.sample(range(1, 11), 5)
print(uzorak1)
```

```
# Postavi sjeme generatora slučajnih brojeva na 2022
random.seed(2022)

# Generiraj drugačiji uzorak od 5 brojeva od 1 do 10 bez ponavljanja
uzorak2 = random.sample(range(1, 11), 5)
print(uzorak2)
```

Ukoliko želimo kreirati veći broj slučajnih vrijednosti, možemo unutar naše sample funkcije uključiti for-petlju:

```
import random

# Generiranje 13 uzoraka od po 5 nasumičnih brojeva od 1 do 10 bez ponavljanja
uzorci = [random.sample(range(1, 11), 5) for _ in range(13)]
```

```
import random

# Generiranje 13 uzoraka od po 5 nasumičnih brojeva od 1 do 10 s ponavljanjem
uzorci_s_ponavljanjem = [random.choices(range(1, 11), k=5) for _ in range(13)]
```

Na sličan način možemo iz niza koji sadrži slova, generirati slučajan uzorak proizvoljne duljine. Primjera radi, odabrat ćemo duljinu 5.

```
import random

# Generiranje uzorka znakova duljine 5 iz niza znakova a, b, c, d, e, f, g, h, i, j
niz_znakova = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
uzorak_znakova = random.choices(niz_znakova, k=5)

# Ispis uzorka
print(uzorak_znakova)
```

Uz malo razmišljanja u Pythonu možemo izračunati i vjerojatnost da barem dvoje od n ljudi ima rođendan na isti dan. Uzet ćemo 23 osobe za naš primjer:

```
# Broj ljudi
k = 23

# Izračun vjerojatnosti da barem dvoje od 23 ljudi ima isti rođendan
p = 1 - (365 / 365)  # Početna vjerojatnost da nema zajedničkog rođendana

for i in range(k):
    p *= (365 - i) / 365  # Izračun vjerojatnosti da nema zajedničkog rođendana za svaku osobu

p = 1 - p  # Konačna vjerojatnost da barem dvoje ljudi imaju isti rođendan

# Ispis rezultata
print("Tražena vjerojatnost je jednaka", p)
```

U R-u postoje određene funkcije za izračun vjerojatnosti rođendana. Primjerice, kolike su šanse da od skupa odabranih osoba, bar 2 imaju rođendan na isti dan. U Pythonu one

nisu ugrađene, ali svejedno ih možemo korisnički definirati. (NAPOMENA: print f je Python funkcija koja korisniku omogućuje ispis formatiranog stringa koji će se mijenjati ovisno o vrijednosti pojedine varijable koju ispisujemo)

```
# Funkcija za izračun vjerojatnosti da će barem dvije osobe (od ukupno jako puno) imati rođendan istog dana.
def vjerojatnost_rođendana(k):
    p = 1
    for i in range(k):
        p *= (365 - i) / 365
    return 1 - p

# Funkcija za izračun minimalnog broja ljudi potrebnog za postizanje vjerojatnosti od 50% da barem dvije osobe imaju rođendan istog dana.
def minimalni_broj_osoba(vjerojatnost):
    k = 0
    p = 1
    while p > 1 - vjerojatnost:
        k += 1
        p *= (365 - k) / 365
    return k

# Izračun vjerojatnosti za 23 osobe
vjerojatnost_za_23_osobe = vjerojatnost_rođendana(23)
print(f"Vjerojatnost da će barem dvije osobe od 23 imati rođendan istog dana je: {vjerojatnost_za_23_osobe}")

# Izračun minimalnog broja osoba za vjerojatnost od 50%
minimalan_broj_osoba = minimalni_broj_osoba(0.5)
print(f"Minimalan broj osoba potreban za vjerojatnost od 50% da barem dvije osobe imaju rođendan istog dana je: {minimalan_broj_osoba}")
```

Također, moguće je vizualno prikazati dobivene podatke pomoću grafova. Python koristi biblioteku Matplotlib za izradu grafa i matematičke operacije za izračun vjerojatnosti. Rezultat je graf koji prikazuje promjenu vjerojatnosti za različite brojeve ljudi.

```
import matplotlib.pyplot as plt
import math

# Inicijalizacija vektora za pohranu vjerojatnosti
p = []

# Izračun vjerojatnosti za svaki broj ljudi od 1 do 100
for k in range(1, 101):
    vjerojatnost = 1 - math.prod([(365 - i) / 365 for i in range(k)])
    p.append(vjerojatnost)

# Kreiranje grafa
plt.figure(figsize=(8, 6))
plt.plot(range(1, 101), p, marker='o', markersize=4, linestyle='-')
plt.xlabel("Broj ljudi")
```

```
plt.ylabel("Vjerojatnost višestrukih rođendana")
plt.title("Vjerojatnost višestrukih rođendana u ovisnosti o broju ljudi")
plt.grid(True)
plt.show()
```

Nadalje, možemo i generirati slučajne rođendane za određeni broj ljudi. Konkretno, kreirat ćemo slučajne rođendane za 23 osobe. Kako se radi o rođendanima, nije zabranjeno da se vrijednosti ponavljaju.

```
import random

# Postavljanje početnog stanja generatora slučajnih brojeva (za reproduktivnost)
random.seed(42)

# Generiranje uzorka rođendana za 23 ljudi s ponavljanjem (moguće su iste vrijednosti)
rodjendani = [random.randint(1, 365) for _ in range(23)]

# Ispis uzorka rođendana
print(rodjendani)
```

Za kraj, možemo demonstrirati i rad s velikim brojem simulacija. Naime, radi postizanja boljeg uzorka, možemo izvesti 100000 simulacija i izračunati vjerojatnost da od 100000 ljudi, bar dvije dijele isti rođendan.

```
import random
import numpy as np

# Postavljanje početnog stanja generatora slučajnih brojeva (za reproduktivnost)
random.seed(42)

# Generiranje velikog broja simulacija
broj_simulacija = 10**5
simulacije = [max(np.bincount([random.randint(1, 365) for _ in range(23)])) for _ in
range(broj_simulacija)]

# Izračun postotka simulacija u kojima barem dvije osobe dijele isti rođendan
postotak_simulacija_s_dvostrukim_rodjendanom = sum(np.array(simulacije) >= 2) /
broj_simulacija

# Ispis rezultata simulacije
print(postotak_simulacija_s_dvostrukim_rodjendanom)

# Ispis vjerojatnosti izračunate pomoću pbirthday
vjerojatnost_rođendana = 1 - np.prod(1 - np.arange(1, 366) / 365)
print(vjerojatnost_rođendana)
```

Unutar četvrte cjeline nema programskih dijelova sve do naslova 4.6 -> Primjena R-a za računanje uvjetnih vjerojatnosti. Naravno, ovdje će se nalaziti Python ekvivalenti za računanje uvjetnih vjerojatnosti.

Za početak, simulirat ćemo bacanje kockica. Konkretno, 4 kockice ćemo bacati 12 puta.

```
import numpy as np

# Postavljanje početnog stanja generatora slučajnih brojeva
np.random.seed(42)

# Simulacija bacanja četiri kockice 12 puta
broj_bacanja = 12
broj_kockica = 4
DePom = np.random.choice(6, size=(broj_kockica, broj_bacanja), replace=True)

# Ispis rezultata simulacije
print(DePom)

# Dijeljenje svih elemenata matrice DePom s 6
DePom6 = DePom // 6

# Ispis rezultata dijeljenja
print(DePom6)

# Izračun maksimalnih vrijednosti u svakom stupcu
maksimalne_vrijednosti = np.max(DePom6, axis=0)

# Ispis maksimalnih vrijednosti
print(maksimalne_vrijednosti)

# Izračun prosječne vjerojatnosti kod višestrukog bacanja kockica
prosjecna_vjerojatnost = np.mean(maksimalne_vrijednosti)

# Ispis prosječne vjerojatnosti
print(prosjecna_vjerojatnost)
```

Zatim možemo pokušati napraviti isto, ali sa 100000 ponavljanja.

```
broj_simulacija = 100000
DeMere1 = np.random.choice(6, size=(broj_kockica, broj_simulacija), replace=True)

# Izračun maksimalnih vrijednosti u svakom nizu i prosječna vjerojatnost
maksimalne_vrijednosti_100k = np.max(DeMere1 // 6, axis=0)
prosjecna_vjerojatnost_100k = np.mean(maksimalne_vrijednosti_100k)

# Ispis prosječne vjerojatnosti za 100.000 ponavljanja
print(prosjecna_vjerojatnost_100k)
```

Ovaj kod simulira bacanje četiri kockice po 12 puta, izračunava maksimalne vrijednosti za svaku seriju bacanja i prosječnu vjerojatnost da će pasti određeni broj na kockici. Nakon toga, ista simulacija ponavlja se 100.000 puta i izračunava se prosječna vjerojatnost za taj veći broj simulacija. Ovo može poslužiti za procjenu vjerojatnosti u igrama s kockicama i druge statističke analize.

Uz malu modifikaciju prethodnog koda možemo izračunati i prosječnu vjerojatnost da će pasti dvije šestice nakon 24 bacanja:

```python
import numpy as np

# Postavljanje početnog stanja generatora slučajnih brojeva
np.random.seed(42)

# Broj simulacija
n_simulations = 100000

# Broj bacanja i broj kockica
n_bacanja = 24
n_kockica = 2

# Simulacija bacanja kockica
kockice = np.random.randint(1, 7, size=(n_simulations, n_bacanja, n_kockica))

# Provjerava jesu li oba rezultata u svakom bacanju jednaka 6 (šestice)
is_šestica = (kockice == 6).all(axis=2)

# Računanje maksimalnog broja šestica u 24 bacanja
maksimalne_šestice = is_šestica.sum(axis=1)

# Računanje prosječne vjerojatnosti
prosjecna_vjerojatnost = (maksimalne_šestice >= 2).mean()

print(f"Prosječna vjerojatnost da će pasti dvije šestice nakon 24 bacanja:
{prosjecna_vjerojatnost:.5f}") #.5f nam formatira output na 5 decimala
```

Možemo probati i s bacanjem četiri kockice 12 puta na sličan način.

```python
import random

# Postavljanje početnog stanja generatora slučajnih brojeva (za reproduktivnost)
random.seed(42)

# Simulacija bacanja četiri kockice 12 puta
DePom = [[random.randint(1, 6) for _ in range(4)] for _ in range(12)]

# Ispis rezultata simulacije
for row in DePom:
    print(row)
```

Možemo simulirati i bacanje kockice u 12 serija po 12 puta na sljedeći način:

```python
import numpy as np

# Generiranje uzorka podataka
np.random.seed(42)
DePom = np.random.randint(1, 7, size=(12, 12))

# Izračun maksimalnih vrijednosti u svakom stupcu
DePom2 = np.max(DePom, axis=0)
```

```
# Ispis maksimalnih vrijednosti
print(DePom2)

# Izračun postotka puta kada je maksimalna vrijednost 6
count_6 = np.sum(DePom2 == 6)
percentage_6 = count_6 / 12
print(percentage_6)
```

Zatim ispisujemo postotak slučajeva kada smo dobili 6.

Ukoliko želimo testirati velik broj slučajeva, možemo isprobati 100000 bacanja

```
import numpy as np

# Postavljamo širinu za bolju čitljivost
np.set_printoptions(linewidth=75)

# Generiramo simulaciju bacanja kockica
np.random.seed(42)
DePom = np.random.randint(1, 7, size=(12, 4))

# Ispisujemo logički vektor koji označava jesu li maksimalne vrijednosti u svakom
stupcu jednake 6
print(DePom == 6)

# Računamo broj puta kada je maksimalna vrijednost 6
print(np.sum(DePom == 6))

# Generiramo simulacije bacanja kockica
DeMere1 = np.random.randint(1, 7, size=(100000, 4))

# Računamo postotak puta kada je maksimalna vrijednost 6
print(np.sum(np.max(DeMere1, axis=1) == 6) / 100000)
```

Moguće je i simulirati 12 serija bacanja 4 kockice i izračunati vjerojatnost da je u barem jednom pala šestica.

```
import numpy as np

# Postavljamo početno stanje generatora slučajnih brojeva
np.random.seed(42)

# Postavljamo broj ponavljanja i broj bacanja
brpon = 12
brbac = 4
brojac = 0

# Generiramo simulaciju bacanja kockica
DePom = np.random.randint(1, 7, size=(brbac, brpon))

# Računamo procijenjenu vjerojatnost korištenjem petlji i if-ova
for i in range(brpon):
    j = 0
```

```
    while j < brbac:
        if DePom[j, i] == 6:
            brojac += 1
            j = brbac
        else:
            j += 1

# Ispisujemo procijenjenu vjerojatnost
print("Procijenjena vjerojatnost je", brojac / brpon)
```

Moguće je u Pythonu simulirati i rođenje djece. Ovaj kod simulira rođenje dvoje djece, gdje svako dijete može biti ili muško (označeno s 0) ili žensko (označeno s 1). Zatim računa uvjetnu vjerojatnost da će barem jedno od djece biti žensko, koristeći dva vektora: dijoba, koji označava jesu li oba djeteta ženskog spola, i dijjedno, koji označava je li barem jedno dijete žensko

```
# Python
import random

# Postavljanje početnog stanja generatora slučajnih brojeva
random.seed(42)

# Generiranje niza dij1 koji sadrži 20 nezavisnih bacanja prvog djeteta
dij1 = [random.choice([0, 1]) for _ in range(20)]

# Generiranje niza dij2 koji sadrži 20 nezavisnih bacanja drugog djeteta
dij2 = [random.choice([0, 1]) for _ in range(20)]

# Formiranje vektora dijoba koji označava je li barem jedno dijete žensko
dijoba = [d1 * d2 for d1, d2 in zip(dij1, dij2)]

# Formiranje podataka o djeci
dijpom = list(zip(dij1, dij2))

# Računanje vektora dijjedno koji označava imaju li oba djeteta barem jedno žensko
dijete
dijjedno = [max(d1, d2) for d1, d2 in dijpom]

# Izračun uvjetne vjerojatnosti
print("Tražena uvjetna vjerojatnost je", sum(dijoba) / sum(dijjedno))
```

Istu funkcionalnost možemo postići korištenjem NumPy-a, samo što će kod s NumPy-jem raditi brže i efikasnije.

```
import numpy as np

# Postavljanje početnog stanja generatora slučajnih brojeva
np.random.seed(42)

# Generiranje niza dij1 koji sadrži 20 nezavisnih bacanja prvog djeteta
dij1 = np.random.choice([0, 1], 20, replace=True)

# Generiranje niza dij2 koji sadrži 20 nezavisnih bacanja drugog djeteta
```

```
dij2 = np.random.choice([0, 1], 20, replace=True)

# Izračun broja slučajeva u kojima barem jedno dijete je žensko
nC = np.sum((dij1 == 1) | (dij2 == 1))

# Izračun broja slučajeva u kojima oba djeteta su ženska
nAC = np.sum((dij1 == 1) & (dij2 == 1))

# Izračun uvjetne vjerojatnosti
print("Tražena uvjetna vjerojatnost je", nAC / nC)
```

Na istu tematiku može se napraviti još sličnih simulacija, pomoću random.choice i zip.a(zip služi za povezivanje dvaju Python objekata -> u ovom slučaju listi):

```
import random

random.seed(42)

dij1 = [random.choice([0, 1]) for _ in range(100000)]
dij2 = [random.choice([0, 1]) for _ in range(100000)]

dijoba = [d1 * d2 for d1, d2 in zip(dij1, dij2)]
dijpom = list(zip(dij1, dij2))
dijjedno = [max(d1, d2) for d1, d2 in dijpom]

uvjetna_vjerojatnost = sum(dijoba) / sum(dijjedno)

print(f"Tražena uvjetna vjerojatnost je {uvjetna_vjerojatnost}")
```

```
import random

random.seed(42)

dij1 = [random.choice([0, 1]) for _ in range(100000)]
dij2 = [random.choice([0, 1]) for _ in range(100000)]

nC = sum(d1 == 1 or d2 == 1 for d1, d2 in zip(dij1, dij2))
nAC = sum(d1 == 1 and d2 == 1 for d1, d2 in zip(dij1, dij2))

uvjetna_vjerojatnost = nAC / nC
```

Za sam kraj, pozabavit ćemo se izvlačenjem kuglica crvene i plave boje. Simulirat ćemo situaciju s plavim (označeno s "p") i crvenim (označeno s "c") kuglicama te računati uvjetnu vjerojatnost da će, pod uvjetom da je donesena odluka o odabiru kuglice, kuglica zaista biti crvena.

```
import random

random.seed(42)
kug = [0] * 100000

nov = [random.choices([0, 1], [0.45, 0.55])[0] for _ in range(100000)]
```

```
for i in range(100000):
    if nov[i] == 0:
        kug[i] = random.choices(["p", "c"], [1/3, 2/3])[0]
    else:
        kug[i] = random.choices(["p", "c"], [2/3, 1/3])[0]

nA = kug.count("p")
nAH = sum(1 for i in range(100000) if nov[i] == 0 and kug[i] == "p")

print("Tražena uvjetna vjerojatnost je", nAH / nA)
```