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)
```